香港城市大學
City University of Hong Kong

CS3481 Assignment 3 Study on hierarchical clustering

Chan Tsz Fung (56228799)

Contents

Data Preparation

We load our **wine dataset** from the scikit learn library.

```
from sklearn import datasets
wine = datasets.load_wine()
print(wine)
```

```
{'data': array([[1.423e+01, 1.710e+00, 2.430e+00, ..., 1.040e+00, 3.920e+00,
        1.065e+03],
       [1.320e+01, 1.780e+00, 2.140e+00, ..., 1.050e+00, 3.400e+00,
        1.050e+03],
       [1.316e+01, 2.360e+00, 2.670e+00, ..., 1.030e+00, 3.170e+00,
        1.185e+03],
       ...,
       [1.327e+01, 4.280e+00, 2.260e+00, ..., 5.900e-01, 1.560e+00,
        8.350e+02],
       [1.317e+01, 2.590e+00, 2.370e+00, ..., 6.000e-01, 1.620e+00,
        8.400e+02],
       [1.413e+01, 4.100e+00, 2.740e+00, ..., 6.100e-01, 1.600e+00,
        5.600e+02]]), 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 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,
```

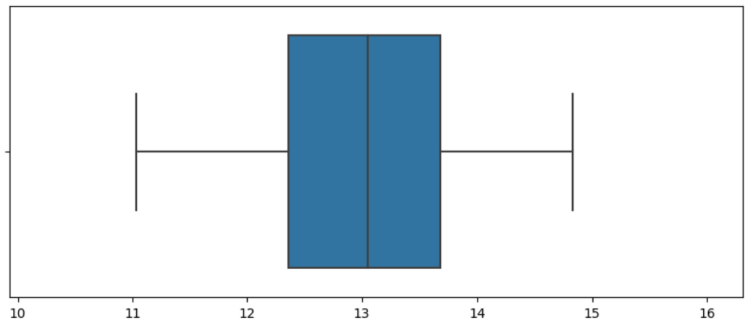
This dataset contains 13 attributes and 1 classification label. There are a total of 178 records. In order to improve clustering results, we have **removed the outliers** and **normalized** the list of values for every attribute.

Removing Outliers

To detect the outliers, we draw box plot using seaborn library. A point is considered as outliers if its value is smaller than Q1-1.5*IQR or its value is larger than Q3+1.5*IQR. We check every attribute one by one as below:

```
import seaborn as sns

plt.figure(figsize=(10,4))
print(str(0)+wine.feature_names[0])
plt.xlim(X[:,0].min()*0.9,X[:,0].max()*1.1)
sns.boxplot(x=X[:,0])
plt.show()
```

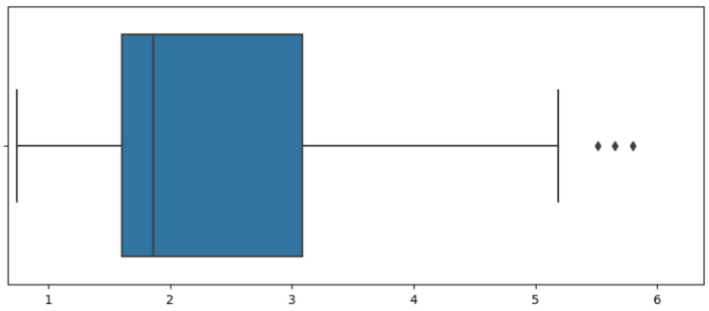
```
0alcohol
```



This attribute has no outliers, we would then look for the next attribute.

```
plt.figure(figsize=(10,4))
print(str(1)+wine.feature_names[1])
plt.xlim(X[:,1].min()*0.9,X[:,1].max()*1.1)
sns.boxplot(x=X[:,1])
plt.show()
```

```
1malic_acid
```



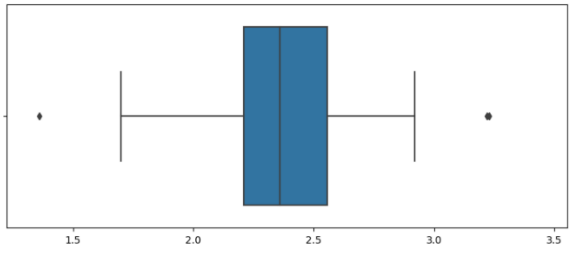
This attribute has three outliers, we would just delete the corresponding three records from the dataset. The number of records became 175.

```
k=sorted(X[:,1])

a=k[-3]
arr_bool=X[:,1]<a
X=X[arr_bool]
Y=Y[arr_bool]
```

Similar deletion of records are performed based on the outliers of other attributes. The results are shown below:

```
plt.figure(figsize=(10,4))
print(str(2)+wine.feature_names[2])
plt.xlim(X[:,2].min()*0.9,X[:,2].max()*1.1)
sns.boxplot(x=X[:,2])
plt.show()
```

2ash

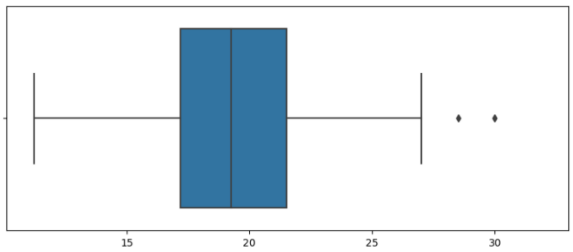



```
k=sorted(X[:,2])

a=k[-2]
b=k[0]
arr_bool1=X[:,2]<a

X=X[arr_bool1]
Y=Y[arr_bool1]
arr_bool2=X[:,2]>b
X=X[arr_bool2]
Y=Y[arr_bool2]
len(X)
```

172

```
plt.figure(figsize=(10,4))
print(str(3)+wine.feature_names[3])
plt.xlim(X[:,3].min()*0.9,X[:,3].max()*1.1)
sns.boxplot(x=X[:,3])
plt.show()
```

3alcalinity_of_ash

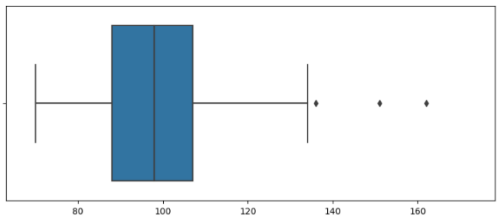



```
k=sorted(X[:,3])

a=k[-2]
arr_bool=X[:,3]<a
X=X[arr_bool]
Y=Y[arr_bool]
len(X)
```

170

```
plt.figure(figsize=(10,4))
print(str(4)+wine.feature_names[4])
plt.xlim(X[:,4].min()*0.9,X[:,4].max()*1.1)
sns.boxplot(x=X[:,4])
plt.show()
```

4magnesium

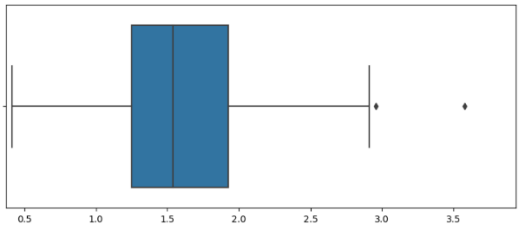



```
k=sorted(X[:,4])

a=k[-3]
arr_bool=X[:,4]<a
X=X[arr_bool]
Y=Y[arr_bool]

len(X)
```

167

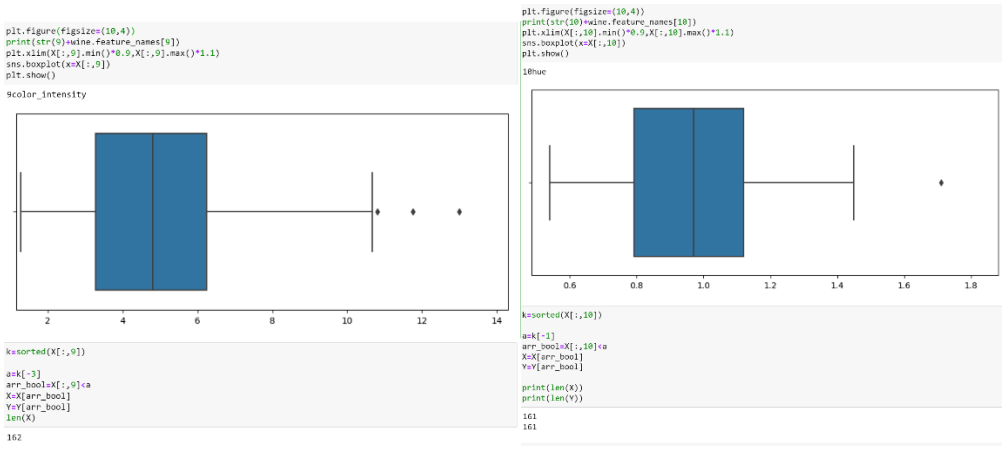
```
plt.figure(figsize=(10,4))
print(str(8)+wine.feature_names[8])
plt.xlim(X[:,8].min()*0.9,X[:,8].max()*1.1)
sns.boxplot(x=X[:,8])
plt.show()
```

8proanthocyanins




```
k=sorted(X[:,8])

a=k[-2]
arr_bool=X[:,8]<a
X=X[arr_bool]
Y=Y[arr_bool]
len(X)
```

165



After one round of detection and removal, 161 records remained. We would use these 161 records for performing our clustering task.

Normalization

Next, we normalize the value in every attribute by using the formula X'= (X-min)/(max-min), where max, min, X, X' are the maximum value, minimum value, original value, and normalized value for the attribute respectively.

```
for j in range(13): #feature
    min=X[:,j].min()
    max=X[:,j].max()
    for i in range(161):#record
        X[i,j]=((X[i,j]-min)/(max-min))
```

The sample normalized result for the first attribute is shown below:

```
X[:,0]
```

```
array([0.8245614 , 0.52339181, 0.51169591, 0.86549708, 0.53508772,
       0.81578947, 0.87134503, 0.7748538 , 1.        , 0.71637427,
       0.78654971, 0.79239766, 0.68421053, 0.97660819, 0.64912281,
       0.84502924, 0.70760234, 0.8128655 , 0.65204678, 0.7748538 ,
       0.44444444, 0.67251462, 0.42105263, 0.61111111, 0.57894737,
       0.55263158, 0.71929825, 0.76315789, 0.67836257, 0.63450292,
       0.66374269, 0.6871345 , 0.61403509, 0.60526316, 0.54678363,
       0.47953216, 0.48538012, 0.82163743, 0.62865497, 0.58479532,
       0.72222222, 0.53508772, 0.47953216, 0.81871345, 0.86842105,
       0.72807018, 0.78654971, 0.73976608, 0.47953216, 0.70760234,
       0.70467836, 0.69005848, 0.68128655, 0.62865497, 0.82163743,
       0.5497076 , 0.6754386 , 0.26900585, 0.35964912, 0.66081871,
       0.28070175, 0.22222222, 0.28070175, 0.49707602, 0.28070175,
       0.56432749, 0.25730994, 0.71637427, 0.60818713, 0.16081871,
       0.07309942, 0.47368421, 0.12573099, 0.37719298, 0.17251462,
       0.38304094, 0.19590643, 0.47953216, 0.12573099, 0.36842105,
       0.21929825, 0.07017544, 0.06725146, 0.19590643, 0.19590643,
       0.17251462, 0.37426901, 0.25730994, 0.06140351, 0.11695906,
       0.25730994, 0.28070175, 0.25730994, 0.19590643, 0.34795322,
       0.27192982, 0.11988304, 0.32163743, 0.29532164, 0.24561404,
       0.38304094, 0.23684211, 0.05847953, 0.3245614 , 0.10233918,
       0.        , 0.19590643, 0.11988304, 0.29532164, 0.39766082,
       0.17251462, 0.01169591, 0.29532164, 0.13450292, 0.19298246,
       0.29824561, 0.28070175, 0.18421053, 0.42397661, 0.42982456,
       0.40935673, 0.37719298, 0.32163743, 0.34795322, 0.24561404,
       0.60818713, 0.41812865, 0.44444444, 0.57017544, 0.61695906,
       0.64619883, 0.24561404, 0.51169591, 0.72222222, 0.42690058,
       0.55847953, 0.48830409, 0.61111111, 0.49707602, 0.53216374,
       0.34210526, 0.51461988, 0.71052632, 0.30409357, 0.27777778,
       0.66666667, 0.42105263, 0.45321637, 0.69298246, 0.67836257,
       0.59649123, 0.4122807 , 0.63450292, 0.58187135, 0.23099415,
       0.39766082, 0.80409357, 0.58187135, 0.54385965, 0.51461988,
       0.79532164])
```

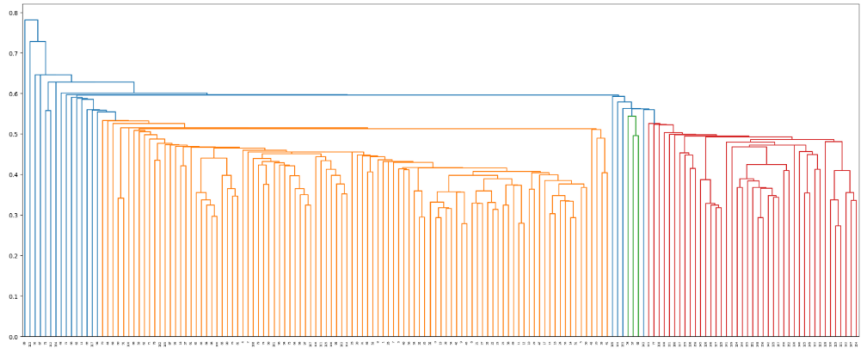
The maximum value is updated as 1 and the minimum value is updated as 0.

After normalization, clustering is performed.

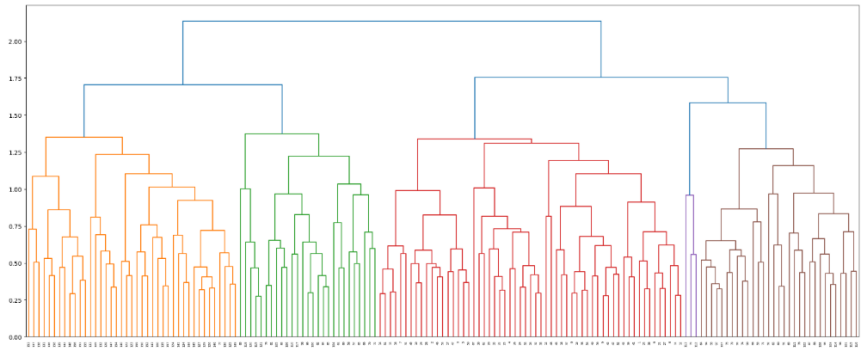
(a) Compare the hierarchical structures generated using single link, complete link and group average for the Wine data set. (30%)

The three constructed hierarchical structures generated using single link, complete link and group average are shown as follows:

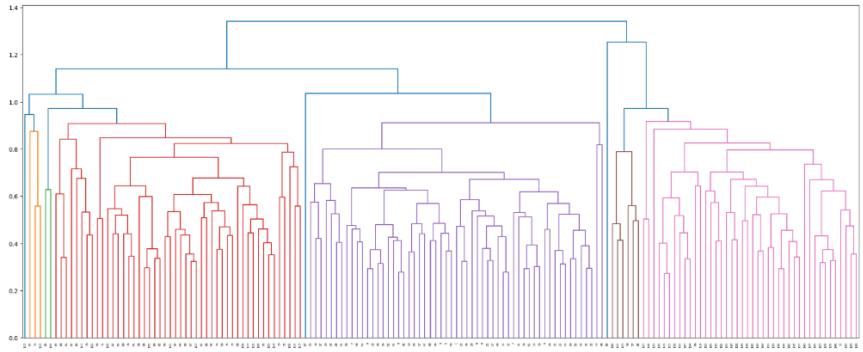
```
Z1 = linkage(X,'single')
plt.figure(figsize=(25, 10))
dendrogram(Z1)
plt.show()
```



```
Z2 = linkage(X,'complete')
plt.figure(figsize=(25, 10))
dendrogram(Z2)
plt.show()
```



```
Z3 = linkage(X,'average')
plt.figure(figsize=(25, 10))
dendrogram(Z3)
plt.show()
```



In general, for the preprocessed 161 records, 160 merges (iterations) should be found in Z, which means len (Z)=160. After the i-th merge, the solution should consist of (161-i) clusters.

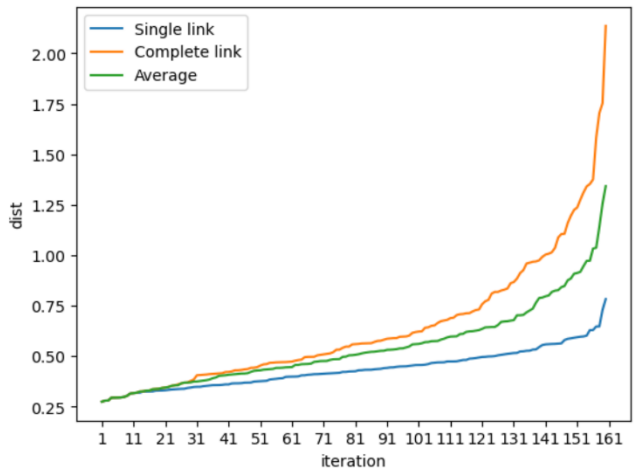
We would compare and analyze the three hierarchical structures in the following ways, they are:

1. distance between two points/clusters when merged
2. size of clusters when being merged

DISTANCE OVERVIEW

First, we would look into the trend of distance increase for each of the approaches. We plot all the lists of distances for a clear visualization:

```
import matplotlib.pyplot as plt
import numpy as np
li_x=[]
for i in range(160):
    li_x.append(i+1)

li_y_dist1 = Z1[:,2].tolist()   #Z1[:,2] just mean all the distances
li_y_dist2 = Z2[:,2].tolist()
li_y_dist3 = Z3[:,2].tolist()


plt.plot(li_x, li_y_dist1,label='Single link')  # Plot the chart
plt.plot(li_x, li_y_dist2,label='Complete link')  # Plot the chart
plt.plot(li_x, li_y_dist3,label='Average')  # Plot the chart
plt.xlabel('iteration')
plt.ylabel('dist')
plt.xticks(range(1,170,10 ))
plt.legend()
plt.show()
```



Obviously, we can see that the rate of distance increase is different for every approach. In the first 20 iterations, the rate of distance increase is still very similar. However, the difference in the rate of distance increase starts to appear when the number of iterations gets larger. This is the time when clusters start to merge with points or cluster starts merging with cluster. When this is the case, the complete link method always selects the maximum distance, therefore reaching a much

higher rate in distance increase. In opposite, the single link method is selecting the minimum value as inter-cluster distance. It results in a lower rate in distance increase compared to the complete link approach. Regarding the rate of distance increase for average method, it is somewhere between the single link and complete link.

Another thing we can look at is the range of merge distance. It is shown that single link method has the smallest range, the complete link method has the largest range, and average method is somewhere in the middle. This also make sense for the different metrics on selecting the inter-cluster distance. If we use a larger distance for inter-cluster distance, the distance we used for merging in the later iterations would increase.

DISTANCE – FROM each point of view

In the following, we would try to find out the merge distance for every record under Single Link, Complete Link and Group Average. We write the result to a csv file.

```
#find distance of merge for each point
li_dist1=[]
for i in range(161): #total 161 points
    li=Z1[:,0].tolist() + Z1[:,1].tolist()
    idx=li.index(i)
    if(idx>=160): #idx 0-159 is pt1
        idx=idx-160
    dist=Z1[:,2].tolist()[idx]
    li_dist1.append(dist)


li_dist2=[]
for i in range(161): #total 161 points
    li=Z2[:,0].tolist() + Z2[:,1].tolist()
    idx=li.index(i)
    if(idx>=160): #idx 0-159 is pt1
        idx=idx-160
    dist=Z2[:,2].tolist()[idx]
    li_dist2.append(dist)


li_dist3=[]
for i in range(161): #total 161 points
    li=Z3[:,0].tolist() + Z3[:,1].tolist()
    idx=li.index(i)
    if(idx>=160): #idx 0-159 is pt1
        idx=idx-160
    dist=Z3[:,2].tolist()[idx]
    li_dist3.append(dist)
print(li_dist1)
print(li_dist2)
print(li_dist3)
import csv

header = ['Point Idx','Single link', 'complete link', 'average']
data=[]
for i in range(161):
    data .append([i,li_dist1[i],li_dist2[i],li_dist3[i]])

with open('distForpts.csv', 'w', encoding='UTF8',newline='') as f:
    writer = csv.writer(f)

    # write the header
    writer.writerow(header)
    writer.writerows(data)
```

The file result is displayed below:

For every point, we rank the 3 distance value, largest distance among the three would be highlighted in red, second largest would be highlighted in green, and the smallest distance would be highlighted in blue. If there are two largest distances, both distance would be highlighted as red and the remaining one would be highlighted as blue. Furthermore, we would not do any highlighting if 3 distances are equal.

Point Idx	Single link	complete link	average
0	0.293638386	0.293638386	0.293638386
1	0.431889329	0.512728315	0.472308822
2	0.455366431	0.469060038	0.462213234
3	0.414047321	0.452478212	0.443521691
4	0.459963508	0.459963508	0.459963508
5	0.368291762	0.368291762	0.368291762
6	0.329016437	0.430799256	0.379907847
7	0.429477466	0.56328036	0.53043768
8	0.436423905	0.479742198	0.516080491
9	0.278463057	0.278463057	0.278463057
10	0.363716073	0.363716073	0.363716073
11	0.278890796	0.278890796	0.278890796
12	0.278890796	0.278890796	0.278890796
13	0.584850611	0.817418732	0.817418732
14	0.293148928	0.293148928	0.293148928
15	0.302857244	0.302857244	0.302857244
16	0.302857244	0.302857244	0.302857244
17	0.384752728	0.441650285	0.441650285
18	0.420984344	0.420984344	0.420984344
19	0.293638386	0.293638386	0.293638386
20	0.489661739	0.596115863	0.581179006
21	0.328066383	0.328066383	0.328066383
22	0.313926332	0.313926332	0.313926332
23	0.313926332	0.313926332	0.313926332
24	0.334958099	0.334958099	0.334958099
25	0.324500236	0.324500236	0.324500236
26	0.431574291	0.459963508	0.459963508
27	0.328066383	0.328066383	0.328066383
28	0.45488062	0.495998433	0.475340661
29	0.363716073	0.363716073	0.363716073
30	0.447870802	0.563548486	0.537813367
31	0.446090461	0.56328036	0.512744115
32	0.294320022	0.294320022	0.294320022
33	0.330118732	0.407872396	0.368995564
34	0.359078142	0.420336286	0.389707214

35	0.294320022	0.294320022	0.294320022
36	0.324500236	0.324500236	0.324500236
37	0.420984344	0.420984344	0.420984344
38	0.315669516	0.315669516	0.315669516
39	0.404285071	0.404285071	0.404285071
40	0.411095372	0.411095372	0.411095372
41	0.404285071	0.404285071	0.404285071
42	0.355750594	0.423565197	0.412703503
43	0.510497925	0.540378925	0.525438425
44	0.53159928	0.588569214	0.57434218
45	0.278463057	0.278463057	0.278463057
46	0.406701089	0.406701089	0.406701089
47	0.398826187	0.441650285	0.441650285
48	0.601001281	0.817418732	0.817418732
49	0.372886263	0.495542221	0.427952428
50	0.368291762	0.368291762	0.368291762
51	0.293148928	0.293148928	0.293148928
52	0.332036236	0.423565197	0.426263079
53	0.406701089	0.406701089	0.406701089
54	0.315669516	0.315669516	0.315669516
55	0.333912061	0.334958099	0.334958099
56	0.411095372	0.411095372	0.411095372
57	0.495745757	0.495745757	0.495745757
58	0.5440353	0.575583317	0.559809308
59	0.505689287	0.505689287	0.505689287
60	0.341173739	0.341173739	0.341173739
61	0.466809581	0.466809581	0.628093796
62	0.497574237	0.563548486	0.715923012
63	0.587755066	0.622201928	0.609823103
64	0.364561958	0.518566268	0.435973342
65	0.495745757	0.495745757	0.495745757
66	0.591157357	0.706896178	0.643609769
67	0.646080789	1.00693017	1.036194381
68	0.526131518	0.778284502	0.641654177
69	0.560022332	0.702993137	0.675237812
70	0.532658516	0.596943001	0.596943001
71	0.487612863	0.505689287	0.505689287
72	0.596943001	0.596943001	0.596943001
73	0.55746423	0.55746423	0.55746423
74	0.37551805	0.37551805	0.37551805
75	0.396360832	0.43353554	0.43353554
76	0.346273066	0.346273066	0.346273066
77	0.523078404	0.569483041	0.542288323
78	0.43514818	0.43514818	0.43514818
79	0.423047785	0.43353554	0.43353554
80	0.364457095	0.466809581	0.440818564

81	0.346273066	0.346273066	0.346273066
82	0.355360983	0.412803968	0.44075908
83	0.397928297	0.46935223	0.44075908
84	0.324608083	0.337331489	0.337331489
85	0.337331489	0.337331489	0.337331489
86	0.296856552	0.296856552	0.296856552
87	0.472701616	0.508594786	0.508594786
88	0.508594786	0.508594786	0.508594786
89	0.781616995	1.002492122	1.254227677
90	0.37551805	0.37551805	0.37551805
91	0.341173739	0.341173739	0.341173739
92	0.646559074	0.96863077	0.875046619
93	0.468733322	0.54636563	0.537357199
94	0.42095524	0.471979422	0.459194523
95	0.376905088	0.428840997	0.402873043
96	0.423681717	0.428939594	0.428939594
97	0.32381391	0.32381391	0.32381391
98	0.554218986	0.639922984	0.597656771
99	0.350170116	0.3592306	0.354700358
100	0.296856552	0.296856552	0.296856552
101	0.428939594	0.428939594	0.428939594
102	0.43514818	0.43514818	0.43514818
103	0.352015568	0.352015568	0.352015568
104	0.628093796	0.77106967	0.628093796
105	0.473413135	0.601039687	0.54691227
106	0.397568697	0.46935223	0.460498161
107	0.32381391	0.32381391	0.32381391
108	0.439523156	0.559895037	0.563226579
109	0.593027614	0.689881017	0.725341192
110	0.514463199	0.558672715	0.558672715
111	0.433949861	0.587070262	0.532053613
112	0.55746423	0.55746423	0.55746423
113	0.72825923	0.957912869	0.945570442
114	0.352015568	0.352015568	0.352015568
115	0.412792634	0.44437035	0.44437035
116	0.443063824	0.44437035	0.44437035
117	0.558672715	0.558672715	0.558672715
118	0.52220491	0.642053703	0.643609769
119	0.273295224	0.273295224	0.273295224
120	0.336941448	0.466205163	0.401573305
121	0.273295224	0.273295224	0.273295224
122	0.412970137	0.412970137	0.412970137
123	0.412970137	0.412970137	0.412970137
124	0.578921456	0.687650401	0.669036924
125	0.354397131	0.354397131	0.354397131
126	0.328397087	0.328397087	0.328397087

127	0.318182818	0.318182818	0.318182818
128	0.480391785	0.494011322	0.518314711
129	0.318182818	0.318182818	0.318182818
130	0.472662934	0.476486836	0.474574885
131	0.499440412	0.531164701	0.530915159
132	0.35530186	0.580541025	0.444846087
133	0.525568998	0.809113558	0.88400766
134	0.367586148	0.367586148	0.367586148
135	0.343273348	0.343273348	0.343273348
136	0.414709712	0.414709712	0.414709712
137	0.45334023	0.503113653	0.503113653
138	0.503113653	0.503113653	0.503113653
139	0.46834416	0.46834416	0.496724753
140	0.456740098	0.510367462	0.48355378
141	0.480142254	0.562042803	0.521092528
142	0.347265836	0.46834416	0.408298875
143	0.559523471	0.672478221	0.734909262
144	0.367586148	0.367586148	0.367586148
145	0.386763638	0.471005535	0.432162346
146	0.323746613	0.328397087	0.328397087
147	0.33594554	0.33594554	0.33594554
148	0.292950127	0.292950127	0.292950127
149	0.354397131	0.354397131	0.354397131
150	0.409859768	0.409859768	0.409859768
151	0.368015895	0.383355272	0.383355272
152	0.409859768	0.409859768	0.409859768
153	0.562499269	0.727623053	0.62169751
154	0.33594554	0.33594554	0.33594554
155	0.383355272	0.383355272	0.383355272
156	0.292950127	0.292950127	0.292950127
157	0.343273348	0.343273348	0.343273348
158	0.414709712	0.414709712	0.414709712
159	0.448295657	0.574431593	0.496724753
160	0.497996659	0.531290806	0.573069838
AVG	0.415976	0.46007	0.454181
MAX	0.781617	1.00693	1.254228
MIN	0.273295	0.273295	0.273295
RANGE	0.508322	0.733635	0.980932

After highlighting, we have noticed serval patterns in the resulted merge distance when using different approaches in calculating inter-cluster distance. Most of the distance for single linked tends to be blue. It means that for each of the records, **the merging distance using single link tends to be smaller compared to those distances using complete link and group average**. The same conclusion can also

be drawn from average merging distance, MAX merging distance and RANGE of merging distance (which is the last 4 row in the table).

On the other hand, majority of the distances using complete link is highlighted as red. There are 64 records (out of 161 records) highlighted as red in complete link, while there are only 33 records highlighted as red in group average. It means that over the 3 different approaches in calculating the inter-cluster distance, **complete link tends to give the largest distance for each of the merge**. The same conclusion can also be drawn from the average of the merging distances. (The average merge distance for complete link is 0.46 while the average merge distance for group average is about 0.45.)

Similarly, most of the **merge distance of records using group average sits between the distance using Single link and complete link.** Their values are highlighted as green in the table. The average merge distance for group average also shows this pattern.

However, the largest merge distance from the table belongs to group average method, instead of complete link method. The reason behind is that we are just displaying the merge distance for single point data (with index 0 – 160) in table, instead of any cluster-and-cluster merging. The distance for cluster-cluster merging cannot be effectively compared as the clusters being merged are just different for different methods starting from index larger than 160. As a result, the max distance and range of distance displayed in this table is different from the overview result drawn from the broken line graph in the last section. In other words, it means that if we consider all the merging including cluster-cluster merging, the MAX distance and the largest range of distance should lie in the Compete link approach. The distance constructed by complete link is generally larger if we consider the full picture, just like the average distance represented for point merges in the table.

After talking about the average distance and max distance, min distances are the same among the three hierarchical structures. They all have the same starting point. This is because they all sharing the same smallest distance when doing their first merge, which is merging point 119 and point 121:

```
print(Z1[0:10])
print(Z2[0:10])
print(Z3[0:10])
```

```
[[119.        121.          0.27329522   2.        ]
 [  9.         45.          0.27846306   2.        ]
 [ 11.         12.          0.2788908    2.        ]
 [148.        156.          0.29295013   2.        ]
 [ 14.         51.          0.29314893   2.        ]
 [  0.         19.          0.29363839   2.        ]
 [ 32.         35.          0.29432002   2.        ]
 [ 86.        100.          0.29685655   2.        ]
 [ 15.         16.          0.30285724   2.        ]
 [ 22.         23.          0.31392633   2.        ]]
[[119.        121.          0.27329522   2.        ]
 [  9.         45.          0.27846306   2.        ]
 [ 11.         12.          0.2788908    2.        ]
 [148.        156.          0.29295013   2.        ]
 [ 14.         51.          0.29314893   2.        ]
 [  0.         19.          0.29363839   2.        ]
 [ 32.         35.          0.29432002   2.        ]
 [ 86.        100.          0.29685655   2.        ]
 [ 15.         16.          0.30285724   2.        ]
 [ 22.         23.          0.31392633   2.        ]]
[[119.        121.          0.27329522   2.        ]
 [  9.         45.          0.27846306   2.        ]
 [ 11.         12.          0.2788908    2.        ]
 [148.        156.          0.29295013   2.        ]
 [ 14.         51.          0.29314893   2.        ]
 [  0.         19.          0.29363839   2.        ]
 [ 32.         35.          0.29432002   2.        ]
 [ 86.        100.          0.29685655   2.        ]
 [ 15.         16.          0.30285724   2.        ]
 [ 22.         23.          0.31392633   2.        ]]
```

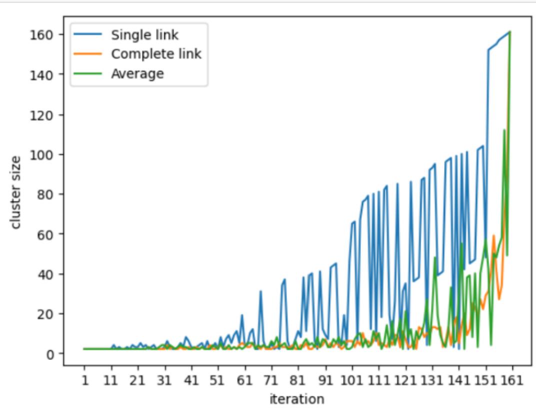
In the next section, we will compare the merge size of the 3 structures.

Size of merge- Overview

To cluster 161 records, there are always 160 merges need to be done, no matter which metrics we use for calculating inter-cluster distance. After any merge step i, the number of clusters being formed would be (161-i) clusters. After 160 iterations, all the records would converge into one cluster.

We first illustrate the size of cluster being formed in each iteration and plot all of them onto a plot:

```
li_y_pts1 =Z1[:,3].tolist()
li_y_pts2 =Z2[:,3].tolist()
li_y_pts3 =Z3[:,3].tolist()


plt.plot(li_x, li_y_pts1,label='Single link')  # Plot the chart
plt.plot(li_x, li_y_pts2,label='Complete link')  # Plot the chart
plt.plot(li_x, li_y_pts3,label='Average')  # Plot the chart
plt.xlabel('iteration')
plt.ylabel('cluster size')
plt.xticks(range(1,170,10 ))
plt.legend()
plt.show()
```



We can see that the merge size of the cluster formed in Single Link fluatate most. It means that a relatively larger cluster is often merge with small clusters **throughout all the iterations**. This make sense as single link always choose smallest value when calculating intercluster distance. As a result, cluster merging **would occur earlier** compared to the other two methods. This can be represented by the peaks of the blue line in the graph starting from iteration 60. At that point, Complete link and group average are still quite stable. It means that the point and point / small cluster to small cluster merging are still occuring continuosly. There are not much difference in the size of the two merging size. There will be no large cluster merging with small cluster at that point.

The peaks of the green line start appearing about iteration 115. It means that at this point, larger cluster starts to merge with some smaller cluster. For complete link, the peak (larger cluster merge with smaller cluster) starts even later. It only occurs at the last part of the 160 iterations (about iteration 125).

Other than the timing of the peaks occurs, the size of the peaks are also different for the three methods. Single link tends to give much larger peaks compared to the other two methods. It means that during the process of merging, Single link continously to merge small cluster with small cluster. But whenever there exist cluster to point distance lower than the point to point distance, the point would be merged into the cluster. This is a frequent case for single link as we always choose the smallest value for claculating intercluster distance. As a result, a larger size cluster are more likely to form in early stage, resulting in a big peak in Single link method.

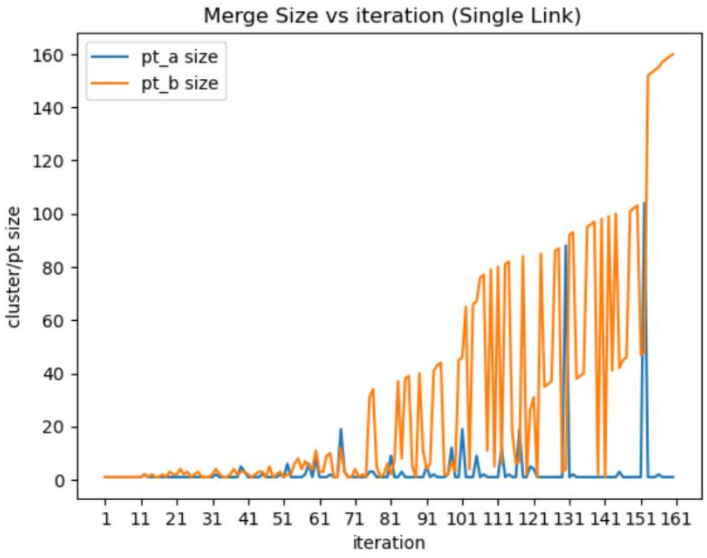
In contrast, the stability of complete link and group average are just the opposite. As we don't prefer cluster-cluster merging so often, Point to point merging would be much more often in the early iterations. This would form many small clusters throughout the process, instead of one large cluster like single link does. Those many clusters would only merge with each other when all the point to pont/ point to cluster merging have been finished, resulting a realtively stable line in most iteration, and big peaks only occur in late iterations (which represents merging of those clusters).

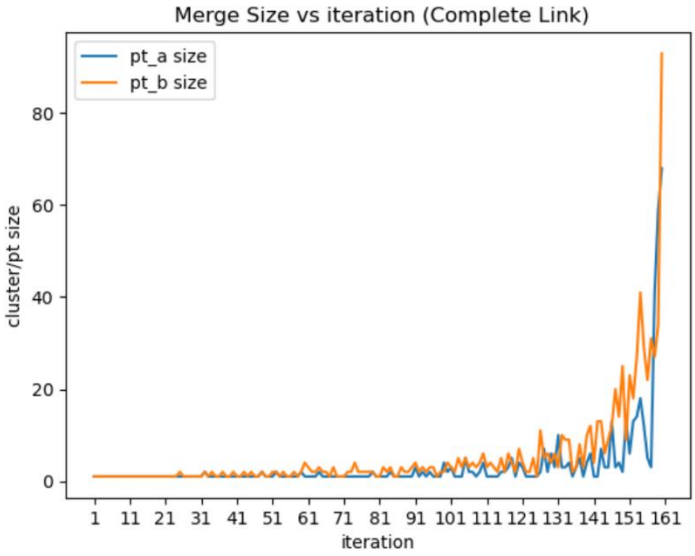
Meanwhile, group average is somewhere between the two methods. The merging between large cluster and small cluster would not occur too early or too late. The size of the peak would also in the middle between the one produced by the two

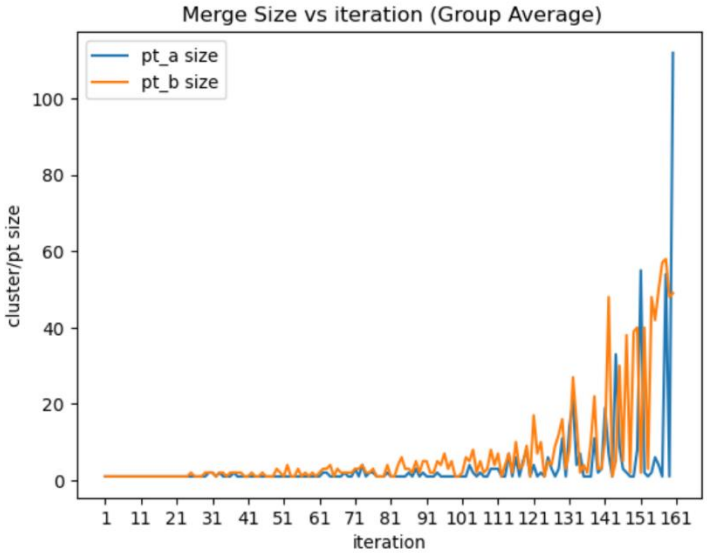
methods. It means that we would not accumulate points into one large cluster like single link does, we would also not produce large number of small clusters like complete link does. The point to cluster or cluster to cluster merging just make the point more likely to be distributed evenly to all the clusters formed, in a smoother pace.

Other than the plot illustrated above, I have also prepared three additional plot for showing the same conclusion. The only difference is that we show the size of cluster A and size of cluster B separately into two lines. These 3 plots share the same obserbvation compared with the previous plot.

```
li_x=[]
for i in range(160):
    li_x.append(i+1)

li_pt1_Z1=Z1[:,0].tolist()
li_pt2_Z1=Z1[:,1].tolist()
li_mergeSize_a=[]
li_mergeSize_b=[]
for i in range(160):##total 161 points,160merges
    a=1
    b=1
    if li_pt1_Z1[i] >=161:
        idx=li_pt1_Z1[i]
        a=Z1[int(idx-161),3]
    if li_pt2_Z1[i] >=161:
        idx=li_pt2_Z1[i]
        b=Z1[int(idx-161),3]
    li_mergeSize_a.append(a)
    li_mergeSize_b.append(b)

plt.plot(li_x, li_mergeSize_a,label='pt_a size')  # Plot the chart
plt.plot(li_x, li_mergeSize_b,label='pt_b size')  # Plot the chart
plt.title(' Merge Size vs iteration (Single Link)')
plt.xlabel('iteration')
plt.ylabel('cluster/pt size')
plt.xticks(range(1,170,10 ))
plt.legend()
plt.show()


li_x=[]
li_pt1_Z2=Z2[:,0].tolist()
li_pt2_Z2=Z2[:,1].tolist()
li_mergeSize_a=[]
li_mergeSize_b=[]
for i in range(160):##total 161 points,160merges
    a=1
    b=1
    if li_pt1_Z2[i] >=161:
        idx=li_pt1_Z2[i]
        a=Z2[int(idx-161),3]
    if li_pt2_Z2[i] >=161:
        idx=li_pt2_Z2[i]
        b=Z2[int(idx-161),3]
    li_mergeSize_a.append(a)
    li_mergeSize_b.append(b)
for i in range(160):
    li_x.append(i+1)
plt.plot(li_x, li_mergeSize_a,label='pt_a size')  # Plot the chart
plt.plot(li_x, li_mergeSize_b,label='pt_b size')  # Plot the chart
plt.title(' Merge Size vs iteration (Complete Link)')
plt.xlabel('iteration')
plt.ylabel('cluster/pt size')
plt.xticks(range(1,170,10 ))
plt.legend()
plt.show()

li_x=[]
li_pt1_Z3=Z3[:,0].tolist()
li_pt2_Z3=Z3[:,1].tolist()
li_mergeSize_a=[]
li_mergeSize_b=[]


for i in range(160):##total 161 points,160merges
    a=1
    b=1
    if li_pt1_Z3[i] >=161:
        idx=li_pt1_Z3[i]
        a=Z3[int(idx-161),3]
    if li_pt2_Z3[i] >=161:
        idx=li_pt2_Z3[i]
        b=Z3[int(idx-161),3]
    li_mergeSize_a.append(a)
    li_mergeSize_b.append(b)
for i in range(160):
    li_x.append(i+1)
plt.plot(li_x, li_mergeSize_a,label='pt_a size')  # Plot the chart
plt.plot(li_x, li_mergeSize_b,label='pt_b size')  # Plot the chart
plt.title(' Merge Size vs iteration (Group Average)')
plt.xlabel('iteration')
plt.ylabel('cluster/pt size')
plt.xticks(range(1,170,10 ))
plt.legend()
plt.show()
```


Merge Size vs iteration (Single Link)
pt_a size
pt_b size
cluster/pt size
iteration
0
20
40
60
80
100
120
140
160
1 11 21 31 41 51 61 71 81 91 101 111 121 131 141 151 161


Merge Size vs iteration (Complete Link)
pt_a size
pt_b size
cluster/pt size
iteration
0
20
40
60
80
1 11 21 31 41 51 61 71 81 91 101 111 121 131 141 151 161


Merge Size vs iteration (Group Average)
pt_a size
pt_b size
cluster/pt size
iteration
0
20
40
60
80
100
1 11 21 31 41 51 61 71 81 91 101 111 121 131 141 151 161

Merge Size- Looking into each cluster solution

In the following, we would look into each cluster solution in detail for supporting the argument we have stated previously.

We would use the following code for analyzing the merge size of the corresponding two clusters in each merge step.

```
li_pt1_Z1=Z1[:,0].tolist()
li_pt2_Z1=Z1[:,1].tolist()
li_pt1_Z2=Z2[:,0].tolist()
li_pt2_Z2=Z2[:,1].tolist()
li_pt1_Z3=Z3[:,0].tolist()
li_pt2_Z3=Z3[:,1].tolist()
li_mergeSize=[]
for i in range(160):
    a1=1
    b1=1
    a2=1
    b2=1
    a3=1
    b3=1
    if li_pt1_Z1[i] >=161:
        idx=li_pt1_Z1[i]
        a1=Z1[int(idx-161),3]
    if li_pt2_Z1[i] >=161:
        idx=li_pt2_Z1[i]
        b1=Z1[int(idx-161),3]
    if li_pt1_Z2[i] >=161:
        idx=li_pt1_Z2[i]
        a2=Z2[int(idx-161),3]
    if li_pt2_Z2[i] >=161:
        idx=li_pt2_Z2[i]
        b2=Z2[int(idx-161),3]
    if li_pt1_Z3[i] >=161:
        idx=li_pt1_Z3[i]
        a3=Z3[int(idx-161),3]
    if li_pt2_Z3[i] >=161:
        idx=li_pt2_Z3[i]
        b3=Z3[int(idx-161),3]
    li_mergeSize.append([i+1,a1,b1,a2,b2,a3,b3])
import csv

header = ['Merge Step','ClusterA MergeSize(Single Link)',
          'ClusterB MergeSize(Single Link)',
          'ClusterA MergeSize(Complete Link)',
          'ClusterB MergeSize(Complete Link)',
          'ClusterA MergeSize(Group Average)',
          'ClusterB MergeSize(Group Average)']


with open('SizeForMerges.csv', 'w', encoding='UTF8',newline='') as f:
    writer = csv.writer(f)

    # write the header
    writer.writerow(header)
    writer.writerows(li_mergeSize)
```

The results are written in csv and shown as below:

Since we have 161 points, Lets define cluster size X as follows:

If X<10, small cluster

If 50>X>=10, medium cluster

If X>=50, large cluster

We would highlight the size with Red, Green and Blue according to the size.

Merge Step	ClusterA MergeSize(Single Link)	ClusterB MergeSize(Single Link)	ClusterA MergeSize(Complete Link)	ClusterB MergeSize(Complete Link)	ClusterA MergeSize(Group Average)	ClusterB MergeSize(Group Average)
1	1	1	1	1	1	1
2	1	1	1	1	1	1
3	1	1	1	1	1	1

4	1	1	1	1	1	1
5	1	1	1	1	1	1
6	1	1	1	1	1	1
7	1	1	1	1	1	1
8	1	1	1	1	1	1
9	1	1	1	1	1	1
10	1	1	1	1	1	1
11	1	1	1	1	1	1
12	2	2	1	1	1	1
13	1	1	1	1	1	1
14	1	2	1	1	1	1
15	1	1	1	1	1	1
16	1	1	1	1	1	1
17	1	2	1	1	1	1
18	1	1	1	1	1	1
19	1	3	1	1	1	1
20	1	2	1	1	1	1
21	1	2	1	1	1	1
22	1	4	1	1	1	1
23	1	2	1	1	1	1
24	1	3	1	1	1	1
25	1	1	1	2	1	2
26	1	2	1	1	1	1
27	1	3	1	1	1	1
28	1	1	1	1	1	1
29	1	1	1	1	1	2
30	1	1	1	1	2	2
31	1	2	1	1	2	2
32	2	4	2	2	1	1
33	1	2	1	1	2	2
34	1	1	1	2	1	2
35	1	1	1	1	1	1
36	1	2	1	1	1	2
37	1	4	1	2	2	2
38	1	2	1	1	1	2
39	5	3	1	1	1	2
40	3	3	1	2	1	1
41	1	2	1	1	1	1
42	1	1	1	1	1	2
43	1	2	1	2	1	1
44	1	3	1	1	1	1
45	2	3	1	2	1	2
46	1	1	1	1	1	1
47	1	5	1	1	1	1
48	1	1	2	2	1	1
49	1	2	1	1	1	3

50	2	3	1	1	1	2
51	1	1	1	2	1	1
52	6	2	2	2	1	4
53	1	2	1	1	1	1
54	1	6	1	2	1	1
55	1	8	1	1	1	3
56	1	4	1	1	1	1
57	2	7	1	2	1	2
58	6	5	1	1	1	1
59	1	4	2	2	1	2
60	8	11	1	4	1	1
61	1	3	1	3	1	2
62	1	3	1	2	2	3
63	1	9	1	2	2	3
64	2	10	2	3	1	4
65	1	1	1	2	1	1
66	1	1	1	2	1	3
67	19	12	1	1	1	2
68	3	3	1	3	2	2
69	1	1	1	1	1	2
70	1	1	1	1	1	2
71	1	4	1	1	3	3
72	1	1	1	2	1	3
73	1	2	1	2	4	4
74	1	1	1	4	1	2
75	3	31	1	2	2	2
76	3	34	1	2	2	3
77	1	5	1	2	1	1
78	1	1	1	2	1	1
79	1	2	2	2	1	1
80	1	6	1	1	2	4
81	9	2	1	1	1	1
82	1	7	1	3	1	1
83	1	37	1	2	1	4
84	3	8	2	3	1	6
85	1	38	1	1	1	3
86	1	39	1	1	2	3
87	1	5	1	3	1	2
88	1	1	1	2	3	5
89	1	40	1	2	1	2
90	1	11	1	3	2	5
91	5	4	3	4	1	5
92	1	6	1	2	1	2
93	2	41	2	3	1	2
94	1	43	1	2	2	5
95	1	44	2	3	1	4

96	1	2	1	3	1	7
97	2	2	1	1	1	3
98	12	7	1	2	1	5
99	1	3	4	2	1	1
100	1	45	2	4	1	1
101	19	46	3	3	1	2
102	1	65	1	2	1	6
103	1	4	1	5	4	5
104	1	66	1	3	2	8
105	9	67	5	5	1	2
106	1	76	2	3	2	5
107	2	77	2	4	1	2
108	1	11	1	3	1	3
109	1	79	2	4	3	8
110	1	5	4	6	3	4
111	1	80	1	3	3	7
112	12	6	1	4	1	1
113	1	81	1	3	1	4
114	2	82	1	2	7	7
115	1	18	2	5	1	2
116	1	6	2	2	6	10
117	19	7	3	6	1	3
118	1	84	5	4	5	4
119	1	2	1	2	8	9
120	5	26	4	7	1	1
121	4	31	3	4	4	17
122	1	1	1	2	1	7
123	1	85	1	2	2	10
124	1	35	1	5	1	1
125	1	36	1	1	6	5
126	1	37	2	11	3	4
127	1	86	7	5	1	9
128	1	87	2	6	3	12
129	1	3	6	4	11	16
130	88	4	3	6	1	3
131	1	92	10	3	14	8
132	2	93	3	10	21	27
133	1	38	3	9	4	15
134	1	39	4	3	7	2
135	1	40	1	2	1	4
136	1	95	3	3	1	2
137	1	96	5	3	1	10
138	1	97	1	3	11	22
139	1	2	4	10	2	3
140	1	98	6	12	3	3
141	1	1	1	4	19	11

142	1	99	1	13	7	48
143	1	41	7	13	1	1
144	1	100	3	6	33	5
145	3	42	3	9	9	30
146	1	45	13	12	3	5
147	1	46	3	20	2	38
148	1	101	4	14	1	2
149	1	102	2	25	1	39
150	1	103	13	9	8	40
151	1	47	6	23	55	2
152	104	48	13	18	2	40
153	1	152	14	27	1	3
154	1	153	18	41	2	48
155	1	154	12	29	6	42
156	2	155	5	22	4	50
157	1	157	3	31	1	57
158	1	158	41	27	54	58
159	1	159	59	34	1	48
160	1	160	68	93	112	49
# of red	34		3		6	
# of green	36		27		26	
# of blue	250		290		288	
Earliest red	#step102		#step159		#step151	
Earliest green	#step60		#step126		#step116	
Earliest blue	#step1		#step1		#step1	

After highlighting all the data, we can support our claims stated in the previous section. From the table, we can see that **medium size cluster and large size cluster appears earlier when using single link approach**. This is because we use a smaller distance in calculating inter-cluster distance. Point-to-cluster merging and cluster-to-cluster merging would be more preferrable over point-to-point merging compared to the other two approaches. This would result in **a larger cluster being formed in earlier iterations**. These large clusters are later merge with the small

cluster and single point for the single link approach. As a result, **the cluster formed using single link would be larger in the later iterations.**

In contrast, the Complete link approach is an opposite situation. It does not prefer cluster merging like single link does. Therefore, it would only perform cluster to cluster merging or cluster to point merging in the later iteration. This would result in **many small clusters being formed in earlier iterations**, instead of one large cluster and many small clusters like single link does. This explains why the **number of large size cluster is smaller in Complete link** compared to Single link (3<34). Meanwhile, this also explains why the **number of small and medium size clusters is greater than** that generated by Single links. (290+27>250+36 for medium and small clusters).

To justify this claim, we can also look at the cluster solution in different steps.

We can first look at a larger k in early iteration. We look at step 10, where k is 161-10=151 clusters being formed in this stage.

```
kclusters1 = fcluster(Z1, 151, criterion='maxclust')
kclusters2 = fcluster(Z2, 151, criterion='maxclust')
kclusters3 = fcluster(Z3, 151, criterion='maxclust')
print(kclusters1)
print(kclusters2)
print(kclusters3)
```

```
[ 43  70  75  40  76  64  51  68  71  47  58  56  56  96  61  60  60  66
   7  43  87  49  52  52  63  54  69  50  74  59  73  72  41  53  42  41
  55   8  44  85  38  86  48  88  91  47  36  67 146  57  65  61  46  37
  45  62  39  99 101  83   3  77  82  97  23 100  98 148  90  95  92  81
 145   1  17  24  13 139   5  19  15  14  12  16  10  11   9  79  84 151
  18   4 149  78  25  31  26  20  93  22   9  27   6  29 147  80  32  21
  28 144  89  34   2 150  30  33  35  94 138 129 130 129 124 125 143 122
 109 106 134 107 127 136 133 140 111 114 102 105 137 121 126 128 116 141
 112 110 108 131 113 123 119 117 120 142 132 118 113 115 103 104 135]
[ 99 112  75  81  87  79 115  68 117 104  70 116 116  94  66  67  67  77
  96  99  86 113  84  84  89 110  88 114  76  71  82  69  91  85  92  91
 111  97 100 107 102 108 105 109  98 104  73  78  95  72  80  66 106  74
 101  90 103  60  62 131 134  57  83 136 125  61  65  93 133 141  63 132
  64 118 128 126  45  40 138 127  58  46  55  47  53  54  52 142 143  44
 129 135 137 130 124 146 148 121  56 123  52 149 139 144  59  49  48 122
 147  16  50 140 119 120 145 150 151  51  43  41  42  41  18  19  31  37
  34  32  14  33  39   6  15  17  28  24   4   1   2   7  20  30   8  27
  29  36  35  12   9  38  21  10  22   3  13  11   9  25   5  23  26]
[ 65  82  63  79  94  77  85  93  88  68  71  86  86 102  90  91  91  73
  54  65  60  83  96  96  98  80  95  84  64  72  89  92 100  97 101 100
  81  55  66  57  75  58  69  59  56  68  61  74 103  87  78  90  70  62
  67  99  76 108 110  15   7   5  14   9  32 109 119 104  48  13  49  16
  50   1  36  38  19 147  10  39  21  20  17  18  24  25  23  34  35 151
  37   8   3  40  33  45  27  29  26  31  23  28  11  43   6  22  46  30
  47  53  51  12   2   4  44  41  42  52 120 113 114 113 105 106 148 144
 141 139 118 140 146 127 117 150 130 136 125 111 112 128 107 132 138 149
 131 143 142 115 135 145 121 133 122 124 116 134 135 137 126 129 123]
```

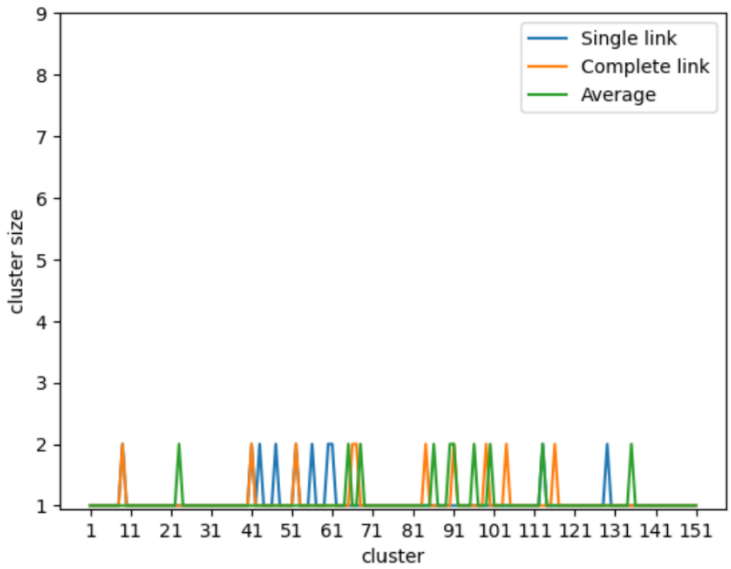
The distribution is extracted in following way:

```
li=[] #store num of pts in each cluster [numpointsincluster1,numpointsincluster2,...]
for i in range(151):#loop the cluster
    cnt=0
    for j in range(161): #loop the list to see if an element belongs to that cluster
        if kclusters1[j]==i+1:
             cnt+=1
    li.append(cnt)

li2=[] #store num of pts in each cluster [numpointsincluster1,numpointsincluster2,...]
for i in range(151):#loop the cluster
    cnt=0
    for j in range(161): #loop the list to see if an element belongs to that cluster
        if kclusters2[j]==i+1:
             cnt+=1
    li2.append(cnt)

li3=[] #store num of pts in each cluster [numpointsincluster1,numpointsincluster2,...]
for i in range(151):#loop the cluster
    cnt=0
    for j in range(161): #loop the list to see if an element belongs to that cluster
        if kclusters3[j]==i+1:
             cnt+=1

    li3.append(cnt)

print(li)
print(li2)
print(li3)
```

```
[1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1,
1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1,
2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

We can then plot the graph:

```
li_x=[]
for i in range(151):#Loop the cluster
    li_x.append(i+1)

plt.plot(li_x, li,label='Single link')  # Plot the chart
plt.plot(li_x, li2,label='Complete link')  # Plot the chart
plt.plot(li_x, li3,label='Average')  # Plot the chart
plt.xlabel('cluster')
plt.ylabel('cluster size')
plt.xticks(range(1,160,10))
plt.yticks(range(1,10,1))
plt.legend()
plt.show()
```



We can see that for each of the clusters, it only consists of one point or two points. No matter which method we use for calculating inter-cluster distance, the distribution of points in different clusters is similar when k is large (early iterations). This is because there are still no point-to-cluster/cluster-to-cluster merging being done at this stage, resulting in no difference by using different methods in calculating inter-cluster distance.

Then we can look at the situation for a smaller k under different approaches. We would take step 102 as an example, which means the number of clusters k=161-102=59.

```
kclusters1 = fcluster(Z1, 59, criterion='maxclust')
kclusters2 = fcluster(Z2, 59, criterion='maxclust')
kclusters3 = fcluster(Z3, 59, criterion='maxclust')
print(kclusters1)
print(kclusters2)
print(kclusters3)
```

```
[ 7  7  7  7  8  7  7  7  7  7  7  7  7 27  7  7  7  7  5  7 18  7  7  7
  7  7  7  7  7  7  7  7  7  7  7  7  7  5  7 17  7 17  7 19 22  7  7  7
 54  7  7  7  7  7  7  7  7 30 32 15  3  9 14 28  7 31 29 56 21 26 23 13
 53  1  7  7  6 47  4  7  6  6  6  6  6  6 11 16 59  7  3 57 10  7  7
  7  7 24  7  6  7  4  7 55 12  7  7  7 52 20  7  2 58  7  7  7 25 46 41
 41 41 37 37 51 37 34 34 42 34 39 44 41 48 35 35 33 33 45 36 38 40 35 49
 35 34 34 41 35 37 35 35 35 50 41 35 35 35 33 33 43]
[41 44 31 32 35 32 45 30 45 42 31 45 45 38 30 30 30 32 40 41 34 45 34 34
 36 44 35 45 31 31 33 30 36 34 36 36 44 40 41 43 42 43 42 43 40 42 31 32
 39 31 32 30 42 31 41 36 42 27 27 50 52 25 33 52 48 27 29 37 51 55 28 50
 28 46 49 49 20 16 54 49 25 20 23 21 23 23 23 56 56 19 49 52 53 49 48 57
 58 48 24 48 23 58 54 57 26 21 21 48 57  7 22 54 46 47 57 59 59 22 18 17
 17 17  9  9 14 16 15 15  6 15 16  3  6  8 13 11  3  1  1  4  9 13  4 12
 13 15 15  6  5 16 10  5 10  2  6  5  5 11  3 10 11]
[29 32 28 31 35 31 33 34 33 29 30 33 33 37 34 34 34 30 24 29 27 33 36 36
 36 32 35 33 28 30 33 34 36 36 36 36 32 24 29 26 31 26 29 26 25 29 28 30
 38 33 31 34 29 28 29 36 31 41 41 11  6  4 10  7 15 41 45 39 19  9 20 11
 21  1 17 17 12 55  8 17 12 12 12 12 13 13 13 16 16 59 17  6  2 17 15 18
 15 15 14 15 13 15  8 18  5 12 18 15 18 23 22  8  1  3 18 18 18 22 46 43
 43 43 40 40 56 55 54 54 44 54 55 50 44 58 52 53 50 42 42 51 40 52 53 57
 52 54 54 44 53 55 47 53 47 49 44 53 53 53 50 51 48]
```

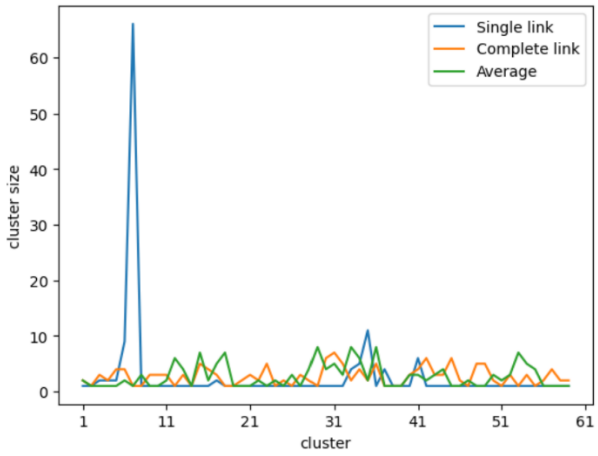
Then we further look at number of points in each cluster:

```
li=[] #store num of pts in each cluster [numpointsincluster1,numpointsincluster2,...]
for i in range(59):#loop the cluster
    cnt=0
    for j in range(161): #loop the list to see if an element belongs to that cluster
        if kclusters1[j]==i+1:
            cnt+=1
    li.append(cnt)

li2=[] #store num of pts in each cluster [numpointsincluster1,numpointsincluster2,...]
for i in range(59):#loop the cluster
    cnt=0
    for j in range(161): #loop the list to see if an element belongs to that cluster
        if kclusters2[j]==i+1:
            cnt+=1
    li2.append(cnt)

li3=[] #store num of pts in each cluster [numpointsincluster1,numpointsincluster2,...]
for i in range(59):#loop the cluster
    cnt=0
    for j in range(161): #loop the list to see if an element belongs to that cluster
        if kclusters3[j]==i+1:
            cnt+=1
    li3.append(cnt)

print(li)
print(li2)
print(li3)
```

```
[1, 1, 2, 2, 2, 9, 66, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4,
5, 11, 1, 4, 1, 1, 1, 6, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[2, 1, 3, 2, 4, 4, 1, 1, 3, 3, 3, 1, 3, 1, 5, 4, 3, 1, 1, 2, 3, 2, 5, 1, 2, 1, 3, 2, 1, 6, 7, 5, 2,
4, 2, 5, 1, 1, 1, 3, 4, 6, 3, 3, 6, 2, 1, 5, 5, 2, 1, 3, 1, 3, 1, 2, 4, 2, 2]
[2, 1, 1, 1, 1, 2, 1, 3, 1, 1, 2, 6, 4, 1, 7, 2, 5, 7, 1, 1, 1, 2, 1, 2, 1, 3, 1, 4, 8, 4, 5, 3, 8,
6, 2, 8, 1, 1, 1, 3, 3, 2, 3, 4, 1, 1, 2, 1, 1, 3, 2, 3, 7, 5, 4, 1, 1, 1, 1]
```

We plot the distribution in broken line graph:

```
li_x=[]
for i in range(59):#Loop the cluster
    li_x.append(i+1)

plt.plot(li_x, li,label='Single link')  # Plot the chart
plt.plot(li_x, li2,label='Complete link')  # Plot the chart
plt.plot(li_x, li3,label='Average')  # Plot the chart
plt.xlabel('cluster')
plt.ylabel('cluster size')
plt.xticks(range(1,65,10))
plt.legend()
plt.show()
```

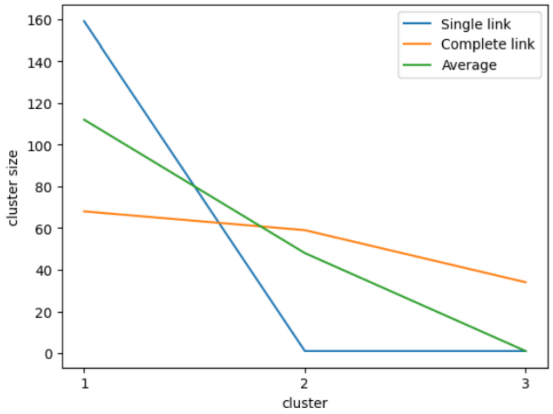


We can see that when k=59 (step102) in single link approach, most of the points (66 out of 161) are packed into one cluster (cluster 7). The points are not evenly distributed. This is the result of preferring cluster merging. Large size clusters are more likely to be formed.

In contrast, the distribution of points for complete link and group averages is more even. It means that by NOT preferring cluster merging, we can still have the 59 clusters with similar proportions of points at this stage.

We can further study the distribution of points for a smaller number of clusters (formed in later iterations). Let's see the case for k=3.

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1]
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 3 3 1 2 3 3 1 1 2 3 3 1 3 1 3
 3 3 1 1 3 3 1 1 1 1 1 1 3 3 1 3 3 3 3 3 3 3 3 1 3 1 3 3 3 1 1 1 3 3 1 1
 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1]
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1
 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2]
[159, 1, 1]
[68, 59, 34]
[112, 48, 1]
```



We can see that the situation for k=3 is similar to the situation for k=59. It is shown that the most uneven distribution of points among clusters is obtained through Single link method. This method gives 3 clusters with corresponding number of points being [159,1,1]. This is the cluster solution by preferring cluster-point/cluster-to-cluster merging. The range of the distribution is 158 and it is very large for the distribution of 161 data. The distribution of points in these clusters can be considered as unevenly distributed, with one cluster being very large and other clusters being very small.

The opposite is the complete link method. By using this method, we can produce 3 clusters with each of the clusters having similar number of items. For the example here, the 3 clusters contain 68,59,34 items respectively. The range of distribution is only 34, which can be considered as an even distribution. The reason we can have such even distribution is that we don't prefer cluster merging compared to point to point merging. In that way, we could possibly form many small clusters without expanding a particular cluster too much.

Again, the Group average method sits somewhere between Single link and Complete link. The distribution of points under this method is [112,48,1] when k=3. It means that there are one large cluster, one medium size cluster and one smaller

size cluster. The range of distribution is 111. It is not evenly distributed compared to complete link, but it is also not unevenly distributed like Single-link does.

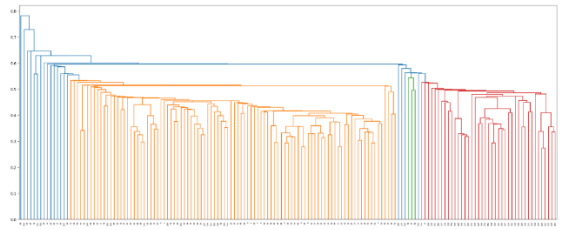
Small Summary for PART a

The hierarchical structure of clusters generated by three different methods are different in both merge distance and merge size.

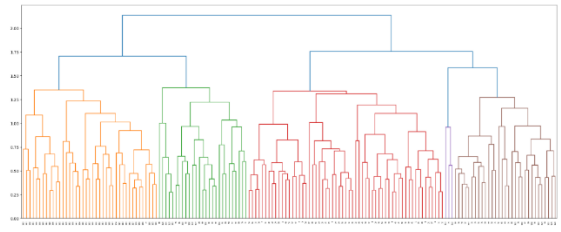
The merge distance of Single links tends to be smaller, while merge distance of Complete link tends to be larger. The group average just sits between the two.

The merge size of cluster in Single link tends to be larger (One cluster would include more points), and the distribution of points in different clusters tends to be more uneven. In opposite, the merge size of cluster in complete link tends to be smaller larger (One cluster would include fewer points), and the distribution of points in different clusters tends to be more even. The group average just sits between the two.

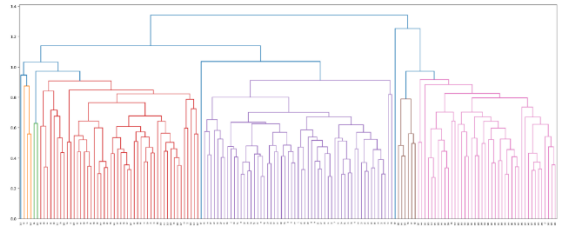
```
Z1 = linkage(X,'single')
plt.figure(figsize=(25, 10))
dendrogram(Z1)
plt.show()
```



```
Z2 = linkage(X,'complete')
plt.figure(figsize=(25, 10))
dendrogram(Z2)
plt.show()
```



```
Z3 = linkage(X,'average')
plt.figure(figsize=(25, 10))
dendrogram(Z3)
plt.show()
```

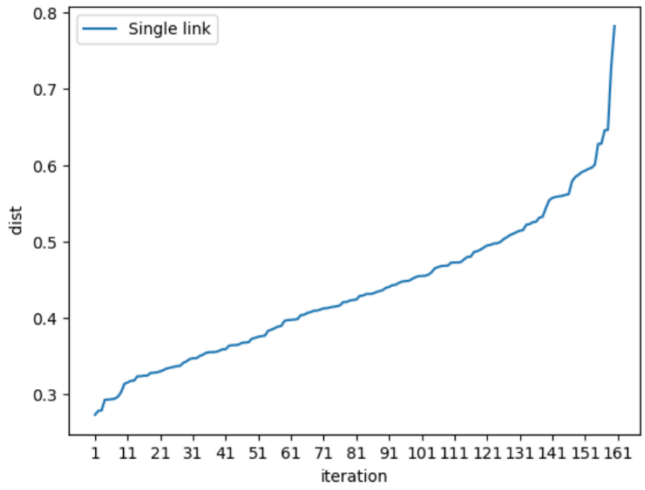


(b) For some of these hierarchical structures, observe the set of distance values at which cluster merge occurs, and identify possible patterns from these values. (20%)

Single Link

We first consider the set of distance values for single link. We plot all the distance values with respect to the number of iterations:

```
#extract all the merge distance
li_dist=Z1[:,2].tolist()
li_x=[]
for i in range(160):#160 merges
    li_x.append(i+1)

plt.plot(li_x, li_dist,label='Single link')  # Plot the chart
plt.xlabel('iteration')
plt.ylabel(' dist')
plt.xticks(range(1,162,10))
plt.legend()
plt.show()
```



It is shown that the distance values increase at a stable pace. The big jump of merge distances only occurs in the last 20 iterations.

We can further derive the distance jump by taking the merge distance between every two iterations:

```
#reformat the list by taking difference
li_dist_diff=[]
for i in range(159):
    li_dist_diff.append(li_dist[i+1]-li_dist[i])
formatted_li_dist_diff = [ '%.4f' % elem for elem in li_dist_diff]
print(formatted_li_dist_diff)
```

```
['0.0052', '0.0004', '0.0141', '0.0002', '0.0005', '0.0007', '0.0025', '0.0060', '0.0111', '0.0017',
'0.0020', '0.0005', '0.0056', '0.0001', '0.0007', '0.0001', '0.0035', '0.0003', '0.0006', '0.0011',
'0.0019', '0.0019', '0.0010', '0.0010', '0.0010', '0.0004', '0.0038', '0.0021', '0.0030', '0.0010',
'0.0001', '0.0028', '0.0018', '0.0024', '0.0009', '0.0001', '0.0004', '0.0012', '0.0021', '0.0000',
'0.0046', '0.0007', '0.0001', '0.0007', '0.0023', '0.0004', '0.0003', '0.0046', '0.0011', '0.0015',
'0.0009', '0.0005', '0.0065', '0.0014', '0.0020', '0.0020', '0.0009', '0.0067', '0.0009', '0.0003',
'0.0004', '0.0009', '0.0050', '0.0004', '0.0024', '0.0011', '0.0018', '0.0003', '0.0012', '0.0017',
'0.0002', '0.0011', '0.0007', '0.0007', '0.0013', '0.0043', '0.0000', '0.0021', '0.0006', '0.0006',
'0.0047', '0.0005', '0.0020', '0.0001', '0.0003', '0.0021', '0.0012', '0.0013', '0.0031', '0.0012',
'0.0023', '0.0005', '0.0026', '0.0018', '0.0004', '0.0005', '0.0022', '0.0023', '0.0015', '0.0003',
'0.0002', '0.0014', '0.0032', '0.0051', '0.0018', '0.0013', '0.0002', '0.0004', '0.0039', '0.0000',
'0.0001', '0.0006', '0.0036', '0.0031', '0.0002', '0.0061', '0.0011', '0.0020', '0.0028', '0.0026',
'0.0006', '0.0018', '0.0004', '0.0014', '0.0037', '0.0026', '0.0029', '0.0019', '0.0021', '0.0019',
'0.0007', '0.0071', '0.0009', '0.0025', '0.0006', '0.0055', '0.0011', '0.0114', '0.0102', '0.0032',
'0.0012', '0.0009', '0.0005', '0.0014', '0.0011', '0.0164', '0.0059', '0.0029', '0.0034', '0.0019',
'0.0024', '0.0015', '0.0041', '0.0271', '0.0004', '0.0176', '0.0005', '0.0817', '0.0534']
```

We can see that most of the jump distances are similar and lie between 0.0001 and 0.01. However, there are some big jumps (with jump values>0.01) discovered in the list. Most of them lie in the last part of the list:

```
for idx,e in enumerate(formatted_li_dist_diff):
     if float(e)>0.01:
        print([idx,e])
```

```
[2, '0.0141']
[8, '0.0111']
[137, '0.0114']
[138, '0.0102']
[145, '0.0164']
[153, '0.0271']
[155, '0.0176']
[157, '0.0817']
[158, '0.0534']
```

These are the set of thresholds we could consider for selecting an optimal clustering solution. The selection criteria is that we want to avoid having big jumps and also the number of clusters should not be too large.

For this purpose, we can sort all the jump distances, and select the top 4 largest jump from it:

```
largest=sorted(formatted_li_dist_diff)[-1]
largest_idx=formatted_li_dist_diff.index(largest)
print('\nLargest jump of merge distance is'+str(largest)
     +'\nIt is found between %s merge and %s merge (from k=%s to k= %s)\n'
      %(largest_idx+1,largest_idx+2,161-(largest_idx+1),161-(largest_idx+2)))


largest=sorted(formatted_li_dist_diff)[-2]
largest_idx=formatted_li_dist_diff.index(largest)
print('Second largest jump of merge distance is'+str(largest)
     +'\nIt is found between %s merge and %s merge (from k=%s to k= %s)\n'
      %(largest_idx+1,largest_idx+2,161-(largest_idx+1),161-(largest_idx+2)))

largest=sorted(formatted_li_dist_diff)[-3]
largest_idx=formatted_li_dist_diff.index(largest)
print('Third largest jump of merge distance is'+str(largest)
     +'\nIt is found between %s merge and %s merge (from k=%s to k= %s)\n'
      %(largest_idx+1,largest_idx+2,161-(largest_idx+1),161-(largest_idx+2)))

largest=sorted(formatted_li_dist_diff)[-4]
largest_idx=formatted_li_dist_diff.index(largest)
print('Fourth largest jump of merge distance is'+str(largest)
     +'\nIt is found between %s merge and %s merge (from k=%s to k= %s)\n'
      %(largest_idx+1,largest_idx+2,161-(largest_idx+1),161-(largest_idx+2)))
```

```
Largest jump of merge distance is0.0817
It is found between 158 merge and 159 merge (from k=3 to k= 2)

Second largest jump of merge distance is0.0534
It is found between 159 merge and 160 merge (from k=2 to k= 1)

Third largest jump of merge distance is0.0271
It is found between 154 merge and 155 merge (from k=7 to k= 6)

Fourth largest jump of merge distance is0.0176
It is found between 156 merge and 157 merge (from k=5 to k= 4)
```

In this case, k=7 is one of the optimal clustering solutions we can select. It is shown that after 154th merges, the jump of the merge distances starts exceeding the usual jump values (0.0271). This implies that we are merging clusters that should not be merged. If we select k=7, the 4 largest distance jumps can be avoided. K=7 produced in merge step 154 is one of the optimal solutions. It is better not to proceed further.

The corresponding clustering solution should be:

```
kclusters1 = fcluster(Z1, 7, criterion='maxclust')
kclusters1
```

```
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, 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, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 7, 2, 2, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, 2, 2, 2, 2,
       2, 2, 1, 6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2], dtype=int32)
```

Apart from this solution, another solution we can select is k=15, this is derived from the fifth largest jump. In that way, we can avoid more large distance jumps. On the other hand, we are also having more clusters as our solution:

```
largest=sorted(formatted_li_dist_diff)[-5]
largest_idx=formatted_li_dist_diff.index(largest)
print('Fifth largest jump of merge distance is'+str(largest)
     +'\nIt is found between %s merge and %s merge (from k=%s to k= %s)\n'
      %(largest_idx+1,largest_idx+2,161-(largest_idx+1),161-(largest_idx+2)))
```

```
Fifth largest jump of merge distance is0.0164
It is found between 146 merge and 147 merge (from k=15 to k= 14)
```

```
kclusters1 = fcluster(Z1, 15, criterion='maxclust')
kclusters1
```

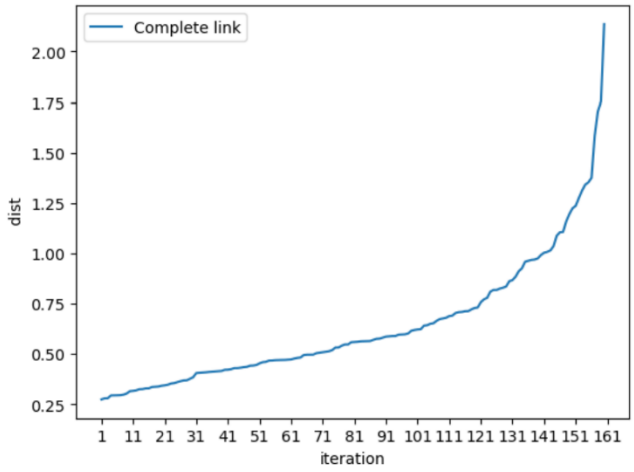
```
array([ 2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  3,  2,  2,  2,
        2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2, 10,  2,  2,
        2,  2,  2,  2,  2,  2,  6,  6,  2,  2,  2,  2,  4,  2,  6,  5, 12,
        2,  2,  2,  2,  9,  1,  2,  2,  2,  6,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2, 15,  2,  2, 13,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2, 11,  2,  2,  2,  2,  8,  2,  2,  1, 14,  2,  2,  2,  2,  6,
        6,  6,  6,  6,  6,  7,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,
        6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,
        6,  6,  6,  6,  6,  6,  6,  6], dtype=int32)
```

Complete Link

We can also analyze the distance jump for the complete link case.

Similar, we obtain the merge distance trend:

```
#extract all the merge distance
li_dist=Z2[:,2].tolist()
li_x=[]
for i in range(160):#160 merges
    li_x.append(i+1)

plt.plot(li_x, li_dist,label='Complete link')  # Plot the chart
plt.xlabel('iteration')
plt.ylabel(' dist')
plt.xticks(range(1,162,10))
plt.legend()
plt.show()
```



We can see that the situation is the same as the situation of single link. The distance increased in a predictable pace for most of the iterations. There is only a sudden increase in the distance jump starting from last 20 iterations.

We can derived the list of jump distances by taking the difference of every two merge differences:

```
reformat the list by taking difference
i_dist_diff=[]
or i in range(159):
    li_dist_diff.append(li_dist[i+1]-li_dist[i])
ormatted_li_dist_diff = [ '%.4f' % elem for elem in li_dist_diff]
rint(formatted_li_dist_diff)
```

```
'0.0052', '0.0004', '0.0141', '0.0002', '0.0005', '0.0007', '0.0025', '0.0060', '0.0111', '0.0017',
0.0025', '0.0056', '0.0007', '0.0036', '0.0003', '0.0066', '0.0010', '0.0014', '0.0038', '0.0021',
0.0030', '0.0057', '0.0024', '0.0048', '0.0045', '0.0039', '0.0007', '0.0072', '0.0078', '0.0209',
0.0010', '0.0014', '0.0012', '0.0020', '0.0012', '0.0017', '0.0002', '0.0017', '0.0056', '0.0006',
0.0026', '0.0053', '0.0001', '0.0019', '0.0027', '0.0016', '0.0059', '0.0006', '0.0027', '0.0081',
0.0058', '0.0017', '0.0062', '0.0006', '0.0015', '0.0007', '0.0003', '0.0003', '0.0013', '0.0010',
0.0045', '0.0033', '0.0021', '0.0122', '0.0015', '0.0002', '0.0003', '0.0071', '0.0026', '0.0029',
0.0018', '0.0024', '0.0058', '0.0126', '0.0001', '0.0091', '0.0060', '0.0002', '0.0109', '0.0012',
0.0012', '0.0021', '0.0005', '0.0008', '0.0003', '0.0059', '0.0049', '0.0012', '0.0050', '0.0048',
0.0017', '0.0013', '0.0002', '0.0067', '0.0009', '0.0008', '0.0041', '0.0134', '0.0041', '0.0025',
0.0012', '0.0177', '0.0021', '0.0072', '0.0025', '0.0120', '0.0086', '0.0036', '0.0030', '0.0086',
0.0022', '0.0131', '0.0039', '0.0013', '0.0034', '0.0006', '0.0085', '0.0070', '0.0026', '0.0264',
0.0144', '0.0072', '0.0308', '0.0083', '0.0001', '0.0069', '0.0043', '0.0058', '0.0263', '0.0055',
0.0173', '0.0271', '0.0143', '0.0327', '0.0045', '0.0045', '0.0017', '0.0057', '0.0168', '0.0114',
0.0044', '0.0072', '0.0216', '0.0514', '0.0171', '0.0015', '0.0526', '0.0364', '0.0289', '0.0121',
0.0390', '0.0362', '0.0297', '0.0118', '0.0224', '0.2073', '0.1222', '0.0486', '0.3833']
```

We can see that most of the value lies between 0.001 and 0.04. For value over 0.05, we can defined that as unusual big jump. Their corresponding index and distance jump are:

```
for idx,e in enumerate(formatted_li_dist_diff):
     if float(e)>0.05:
        print([idx,e])
```

```
[143, '0.0514']
[146, '0.0526']
[155, '0.2073']
[156, '0.1222']
[158, '0.3833']
```

Their corresponding merge step and k values are:

```
largest=sorted(formatted_li_dist_diff)[-1]
largest_idx=formatted_li_dist_diff.index(largest)
print('\nLargest jump of merge distance is'+str(largest)
     +'\nIt is found between %s merge and %s merge (from k=%s to k= %s)\n'
      %(largest_idx+1,largest_idx+2,161-(largest_idx+1),161-(largest_idx+2)))


largest=sorted(formatted_li_dist_diff)[-2]
largest_idx=formatted_li_dist_diff.index(largest)
print('Second largest jump of merge distance is'+str(largest)
     +'\nIt is found between %s merge and %s merge (from k=%s to k= %s)\n'
      %(largest_idx+1,largest_idx+2,161-(largest_idx+1),161-(largest_idx+2)))

largest=sorted(formatted_li_dist_diff)[-3]
largest_idx=formatted_li_dist_diff.index(largest)
print('Third largest jump of merge distance is'+str(largest)
     +'\nIt is found between %s merge and %s merge (from k=%s to k= %s)\n'
      %(largest_idx+1,largest_idx+2,161-(largest_idx+1),161-(largest_idx+2)))

largest=sorted(formatted_li_dist_diff)[-4]
largest_idx=formatted_li_dist_diff.index(largest)
print('Fourth largest jump of merge distance is'+str(largest)
     +'\nIt is found between %s merge and %s merge (from k=%s to k= %s)\n'
      %(largest_idx+1,largest_idx+2,161-(largest_idx+1),161-(largest_idx+2)))

largest=sorted(formatted_li_dist_diff)[-5]
largest_idx=formatted_li_dist_diff.index(largest)
print('Fifth largest jump of merge distance is'+str(largest)
     +'\nIt is found between %s merge and %s merge (from k=%s to k= %s)\n'
      %(largest_idx+1,largest_idx+2,161-(largest_idx+1),161-(largest_idx+2)))
```

```
Largest jump of merge distance is0.3833
It is found between 159 merge and 160 merge (from k=2 to k= 1)

Second largest jump of merge distance is0.2073
It is found between 156 merge and 157 merge (from k=5 to k= 4)

Third largest jump of merge distance is0.1222
It is found between 157 merge and 158 merge (from k=4 to k= 3)

Fourth largest jump of merge distance is0.0526
It is found between 147 merge and 148 merge (from k=14 to k= 13)

Fifth largest jump of merge distance is0.0514
It is found between 144 merge and 145 merge (from k=17 to k= 16)
```

Therefore, k=2,k=5,k=4,k=14,k=17 are all possible solutions. We can choose k=5 such that the cluster solution can avoid 3 big jumps, or we can choose k=14 such that the cluster solution can avoid 4 big jumps, or we can choose k=17 such that the cluster solution can avoid 5 big jumps.

(c)Select different clustering solutions from the hierarchical structures, and compare the cluster groupings with the corresponding K-means clustering solutions (using the method KMeans from the module sklearn.cluster), in terms of the extent to which the clusters can capture the class structure of the data set. (30%)

To start with, lets recap the **ground truth** for the 161 records:

```
print(len(Y)) #161 truth
Y
```

```
161

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

There are :

57 samples belong to class 1,

61 samples belong to class 2,

43 samples belong to class 3.

K=3

Since there are 3 classes in the original data, we consider doing a comparison on k=3 first:

K-Means clustering

```
from sklearn.cluster import KMeans
km = KMeans(n_clusters=3)
km.fit(X)
kmeansol=km.labels_
kmeansol
```

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

<u>By converting to original class:</u>

```
cluster_1_ori_class=[]
cluster_2_ori_class=[]
cluster_3_ori_class=[]
original_classli=Y.tolist()

for i in range (161):
    if kmeansol[i]==0:
        cluster_1_ori_class.append(original_classli[i])
    if kmeansol[i]==1:
        cluster_2_ori_class.append(original_classli[i])
    if kmeansol[i]==2:
        cluster_3_ori_class.append(original_classli[i])
print(cluster_1_ori_class)
print(cluster_2_ori_class)
print(cluster_3_ori_class)
#------
```

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

Hierarchical clustering (Single link)

```
kclusters1 = fcluster(Z1, 3, criterion='maxclust')
print(kclusters1)
```

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

By converting to original class:

```
cluster_1_ori_class=[]
cluster_2_ori_class=[]
cluster_3_ori_class=[]
original_classli=Y.tolist()
for i in range (161):
    if kclusters1[i]==1:
        cluster_1_ori_class.append(original_classli[i])
    if kclusters1[i]==2:
        cluster_2_ori_class.append(original_classli[i])
    if kclusters1[i]==3:
        cluster_3_ori_class.append(original_classli[i])
print(cluster_1_ori_class)
print(cluster_2_ori_class)
print(cluster_3_ori_class)
#------
```

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

Hierarchical clustering (Complete Link)

```
kclusters2 = fcluster(Z2, 3, criterion='maxclust')
print(kclusters2)
```

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

```
cluster_1_ori_class=[]
cluster_2_ori_class=[]
cluster_3_ori_class=[]
original_classli=Y.tolist()
for i in range (161):
    if kclusters2[i]==1:
        cluster_1_ori_class.append(original_classli[i])
    if kclusters2[i]==2:
        cluster_2_ori_class.append(original_classli[i])
    if kclusters2[i]==3:
        cluster_3_ori_class.append(original_classli[i])
print(cluster_1_ori_class)
print(cluster_2_ori_class)
print(cluster_3_ori_class)
```

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

Hierarchical clustering (Group average)

```
kclusters3 = fcluster(Z3, 3, criterion='maxclust')
print(kclusters3)
```

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

```
cluster_1_ori_class=[]
cluster_2_ori_class=[]
cluster_3_ori_class=[]
original_classli=Y.tolist()
for i in range (161):
    if kclusters3[i]==1:
        cluster_1_ori_class.append(original_classli[i])
    if kclusters3[i]==2:
        cluster_2_ori_class.append(original_classli[i])
    if kclusters3[i]==3:
        cluster_3_ori_class.append(original_classli[i])
print(cluster_1_ori_class)
print(cluster_2_ori_class)
print(cluster_3_ori_class)
```

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

The result can be interpreted in the table below (majority class within each cluster are highlighted in yellow):

K=3	Original class	K-Means clustering solution	Hierarchical clustering (Single link) solution	Hierarchical clustering (Complete link) solution	Hierarchical clustering (Group Average) solution
Cluster 1	Class 0	45/45	57/159	0/68	57/112
	Class 1	0/45	59/159	25/68	55/112
	Class 2	0/45	43/159	43/68	0/112
Cluster 2	Class 0	0/60	0/1	57/59	0/48
	Class 1	44/60	1/1	2/59	5/48
	Class 2	16/60	0/1	0/59	43/48
Cluster 3	Class 0	12/56	0/1	0/34	0/1
	Class 1	17/56	1/1	34/34	1/1
	Class 2	27/56	0/1	0/34	0/1
Misclassified samples		45	100	27	60

The performance of cluster grouping can be evaluated in two dimensions. First, to evaluate whether a cluster only consists of a particular class of points. Second, to evaluate whether there exists a cluster for representing each of the classes. If these two criteria are matched, the cluster solution can be considered as having good performance in capturing the class structure of the dataset.

We can see that the cluster groupings obtained by k-means have good performance in capturing the class structure of the dataset. Cluster 1 consists of a majority of points belong to class 0, Cluster 2 consists of a majority of points belong to class 1, Cluster 3 consists of a majority of points belong to class 2. The role of each cluster is distinguishable, and every cluster is representative enough for one of the classes.

This is not the case for hierarchical clustering (single link). We can see that for single link solution, there are no cluster responsible for representing class 0 and class 2. All the three clusters consist of a majority of points belong to class 1. All the points coming from class 0 and all the points coming from class 2 are grouped into cluster 1. That is an uneven distribution of points toward the three clusters. **The role of every cluster is less distinguishable compared to k-means.**

Hierarchical cluster (complete link) provides a better solution compared to the single link. **Same as the k-means solution, every cluster is consisting of a majority of points belonging to one of the classes.** The role of every cluster is clearly distinguishable. Furthermore**, the cluster solution can be said to be performing better compared to k-means**. From the result, we can see that 25 points are being mislabeled in cluster 1, 2 points are being mislabeled in cluster 2, and 0 point are being mislabeled in cluster 3. In total, there are **only 27 points being mislabeled, which is better than k-mean** where there are 45 points being mislabeled in total.

In terms of capturing the class structure, the group average method has also done a good job. We can see that for every cluster, there are a majority of points belong to one of the classes. **This is the same as k-means.** However**, this solution is considered worse** compared to the one provided by k-means. In total, there are 60 points being misclassified. The reason for that is that most of the points belong to class 2 are grouped into cluster 1, which is the representative cluster for class0. Even though the role of every cluster is distinguishable, **the grouping of the points is not desirable compared to k-means.**

In summary, for k=3, the performance of the cluster solution in capturing the class structure is ranked as follows:

Complete link>K-means>Group Average>Single Link

K=10

We perform the same experiment for k=10 and obtain the following table:

K=10	Original class	K-Means clustering solution	Hierarchical clustering (Single link) solution	Hierarchical clustering (Complete link) solution	Hierarchical clustering (Group Average) solution
Cluster 1	Class 0	0/28	0/2	0/12	0/3
	Class 1	15/28	2/2	0/12	3/3
	Class 2	13/28	0/2	12/12	0/3
Cluster 2	Class 0	10/10	56/104	0/29	0/1
	Class 1	0/10	48/104	2/29	1/1
	Class 2	0/10	0/104	27/29	0/1
Cluster 3	Class 0	1/30	0/1	0/5	0/2
	Class 1	12/30	1/1	1/5	2/2
	Class 2	17/30	0/1	4/5	0/2
Cluster 4	Class 0	5/5	1/1	0/22	0/48
	Class 1	0/5	0/1	22/22	48/48
	Class 2	0/5	0/1	0/22	0/48
Cluster 5	Class 0	5/13	0/1	18/18	57/57
	Class 1	3/13	1/1	0/18	0/57
	Class 2	5/13	0/1	0/18	0/57
Cluster 6	Class 0	14/14	0/1	12/14	0/1
	Class 1	0/14	1/1	2/14	1/1
	Class 2	0/14	0/1	0/14	0/1
Cluster 7	Class 0	0/17	0/1	27/27	0/6
	Class 1	14/17	1/1	0/27	3/6
	Class 2	3/17	0/1	0/27	3/6
Cluster 8	Class 0	16/16	0/1	0/3	0/2
	Class 1	0/16	1/1	3/3	0/2
	Class 2	0/16	0/1	0/3	2/2
Cluster 9	Class 0	6/13	0/1	0/13	0/40
	Class 1	2/13	1/1	13/13	2/40
	Class 2	5/13	0/1	0/13	38/40
Cluster 10	Class 0	0/15	0/1	0/18	0/1
	Class 1	15/15	1/1	18/18	1/1
	Class 2	0/15	0/1	0/18	0/1
# cluster representing class 0		6	2	3	1
# cluster representing class 1		3	8	4	7
# cluster representing class 2		1	0	3	2
Total Mis-labelled points		44	48	5	5

For k =10, k means can construct clusters responsible for each class. For example, 14 out of 14 points in cluster 6 belong to class 0. 15/15 points in cluster 10 belong to class 1. 17/30 points in cluster 3 belongs to class 2. Every cluster is composing of a majority of points belong to one particular class. The roles of most clusters are clear and distinguishable.

However, this is not the case for the single link hierarchical clustering solution. In this solution, there is no cluster responsible for representing class 2. Also, there are many class 1 points being grouped into cluster 2, which represents class 0. As a result, there are more mislabeled points under this solution. Furthermore, a lot more clusters are representing class 1, but the size of cluster is only 1. It shows that the distribution of points is uneven compared to k-means and the role of each cluster is also less representative compared to k-means.

On the other hand, the solution constructed using complete link hierarchical clustering is capturing more class structures compared to the k-means solution. First, it consists of clusters representing different classes, and the number of clusters representing each of the classes is quite even. There are 3 clusters representing class 0, 4 clusters representing class 1 and 3 clusters representing class 2. Also, there are fewer number of misclassified samples under this solution. These make the cluster solution have a more representative structure compared to k-means.

Finally, the solution constructed using group average hierarchical clustering is also better than that constructed using k-means. In this solution, there are also clusters representing each of the classes. For example, cluster 4 contains 48/48points from class1, cluster 5 contains 57/57 points from class 0 and cluster 9 contains 38/40 points from class 2. Most of the clusters are also representative and contain only a few numbers of mis-labelled points. To be precise, the number of mis-labelled points in group average hierarchical clustering is only 5, which is much fewer than the number of mis-labelled points in K-means. We can see that this solution outperforms k-means in capturing the class structure.

In summary, for k=10, the performance of the cluster solution in capturing the class structure is ranked as follows:

Complete link = Group Average >K-means >Single Link

(d) Select different subsets of attributes from the data sets and re-perform hierarchical clustering. Compare the resulting hierarchical structures based on the selected attribute subsets with the original hierarchical structures. (20%)

Preparation

To improve the clustering solution of hierarchical clustering, we select attributes that are most likely to be affecting the classification result. The way we do this is to construct a random forest classifier. We let the number of subtrees to be five and fit X and Y to the classifier. Then, we check the importance of every feature for every sub tree. The importance of an attribute is estimated based on the total importance of that attribute in all the subtrees.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
clf1=RandomForestClassifier(n_estimators=5)
clf1=clf1.fit(X,Y)
clf1_1=clf1.estimators_[0]
clf1_2=clf1.estimators_[1]
clf1_3=clf1.estimators_[2]
clf1_4=clf1.estimators_[3]
clf1_5=clf1.estimators_[4]

pred=clf1.predict(X)
print(accuracy_score(pred,Y))
li_importances=[]
for i in range(13):
    importances=clf1_1.feature_importances_[i]+clf1_2.feature_importances_[i]
    +clf1_3.feature_importances_[i]+clf1_4.feature_importances_[i]
    +clf1_5.feature_importances_[i]
    li_importances.append(importances)
print(li_importances)
#idx list according to val(smalltolarge)
ranked_idx=sorted(range(len(li_importances)), key=lambda k: li_importances[k])
ranked_idx.reverse()
print(ranked_idx[:6])
```

```
1.0
[0.4519423414254201, 0.0, 0.014047231270358305, 0.017180205415499535, 0.1076514
8404638727, 0.3719549022439164, 0.5190867055082284, 0.0, 0.0, 0.441880731765387
6, 0.0, 0.0762563983248022, 0.0]
[6, 0, 9, 5, 4, 11]
```

After calculating the importance values for every attribute, we take the top six and top three attributes for performing hierarchical clustering.

```
sub6_X=X[:,[0,4,5,6,9,11]]
print(sub6_X)
```

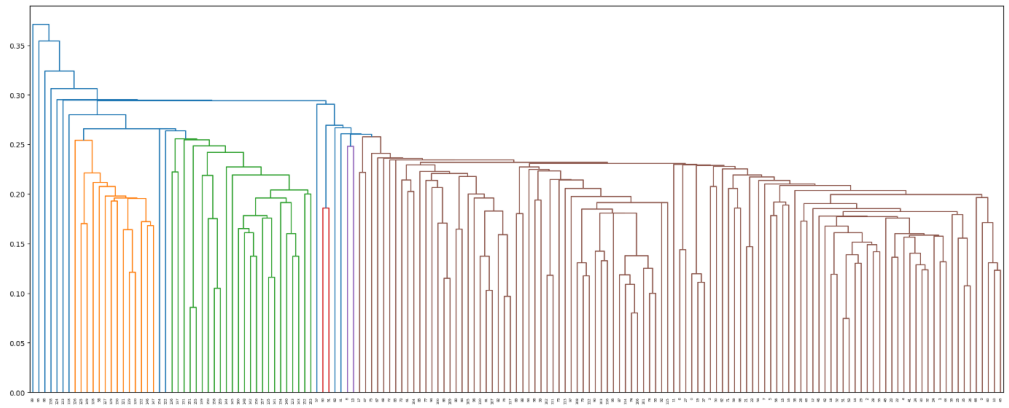
```
[[0.8245614  0.890625   0.62758621 0.75766017 0.46382979 0.97069597]
 [0.52339181 0.46875    0.57586207 0.67409471 0.32978723 0.78021978]
 [0.51169591 0.484375   0.62758621 0.80779944 0.46808511 0.6959707 ]
 [0.86549708 0.671875   0.98965517 0.87743733 0.69361702 0.7985348 ]
 [0.53508772 0.75       0.62758621 0.6545961  0.32340426 0.60805861]
 [0.81578947 0.65625    0.78965517 0.84958217 0.58191489 0.57875458]
 [0.87134503 0.40625    0.52413793 0.60724234 0.42234043 0.84615385]
 [0.7748538  0.796875   0.55862069 0.60445682 0.40106383 0.84615385]
 [1.         0.421875   0.62758621 0.73537604 0.41702128 0.57875458]
 [0.71637427 0.4375     0.68965517 0.78272981 0.63191489 0.83516484]
 [0.78654971 0.546875   0.67931034 0.83008357 0.47553191 0.6959707 ]
 [0.79239766 0.390625   0.42068966 0.5821727  0.39574468 0.56776557]
 [0.68421053 0.296875   0.55862069 0.67409471 0.45957447 0.5970696 ]
 [0.97660819 0.328125   0.73103448 0.93314763 0.43829787 0.53479853]
 [0.64912281 0.65625    0.64482759 0.71587744 0.64042553 0.58974359]
 [0.84502924 0.78125    0.62758621 0.77994429 0.52340426 0.50549451]
 [0.70760234 0.703125   0.67931034 0.85236769 0.56595745 0.47619048]
 [0.8128655  0.59375    0.8        1.         0.7893617  0.56776557]
 [0.65204678 0.71875    0.59310345 0.74930362 0.40638298 0.76556777]
```

```
sub3_X=X[:,[0,6,9]]
print(sub3_X)
```

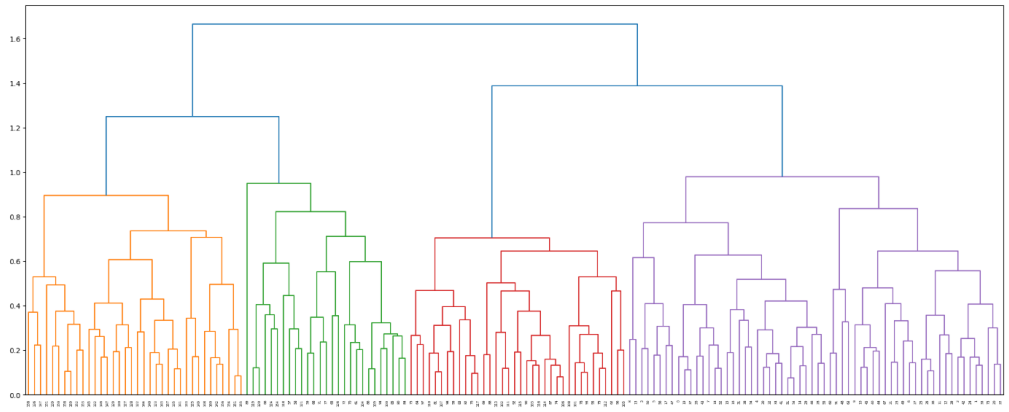
```
[[0.8245614  0.75766017 0.46382979]
 [0.52339181 0.67409471 0.32978723]
 [0.51169591 0.80779944 0.46808511]
 [0.86549708 0.87743733 0.69361702]
 [0.53508772 0.6545961  0.32340426]
 [0.81578947 0.84958217 0.58191489]
 [0.87134503 0.60724234 0.42234043]
 [0.7748538  0.60445682 0.40106383]
 [1.         0.73537604 0.41702128]
 [0.71637427 0.78272981 0.63191489]
 [0.78654971 0.83008357 0.47553191]
 [0.79239766 0.5821727  0.39574468]
 [0.68421053 0.67409471 0.45957447]
 [0.97660819 0.93314763 0.43829787]
 [0.64912281 0.71587744 0.64042553]
 [0.84502924 0.77994429 0.52340426]
 [0.70760234 0.85236769 0.56595745]
 [0.8128655  1.         0.7893617 ]
 [0.65204678 0.74930362 0.40638298]
```

Then we construct the dendrogram using sub6_X and sub3_X

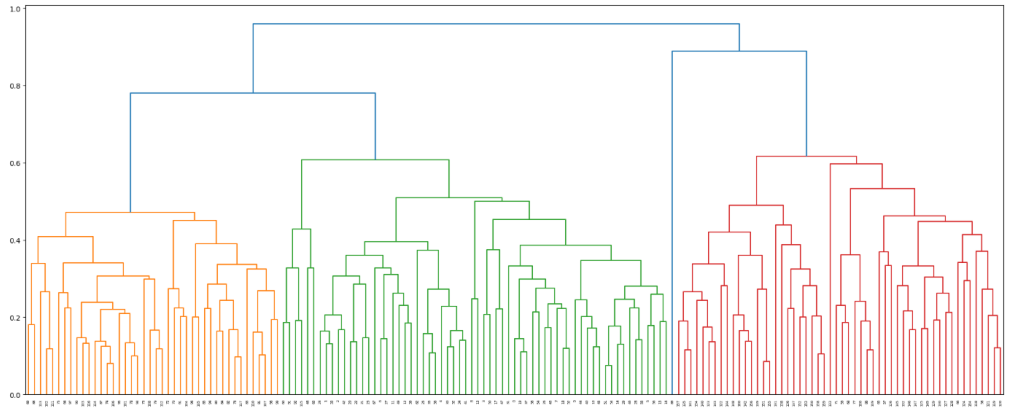
```
Z1_new = linkage(sub6_X,'single')
plt.figure(figsize=(25, 10))
dendrogram(Z1_new)
plt.show()
```



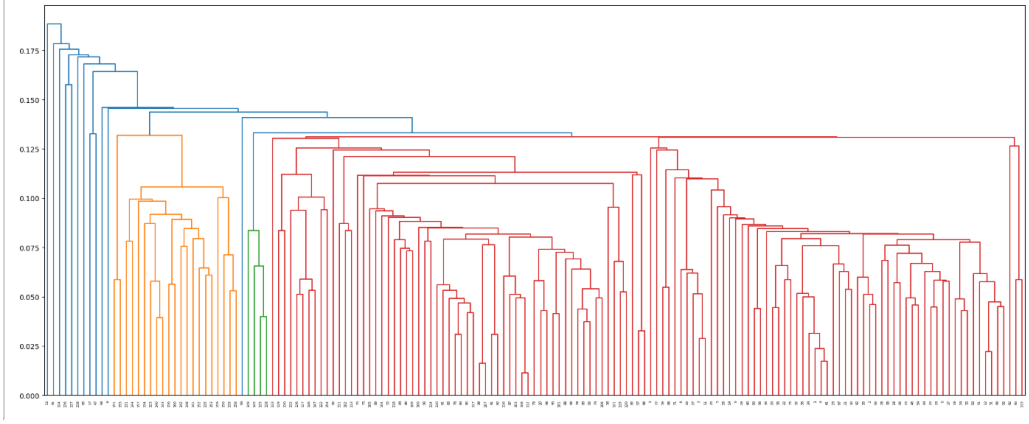
```
Z2_new = linkage(sub6_X,'complete')
plt.figure(figsize=(25, 10))
dendrogram(Z2_new)
plt.show()
```



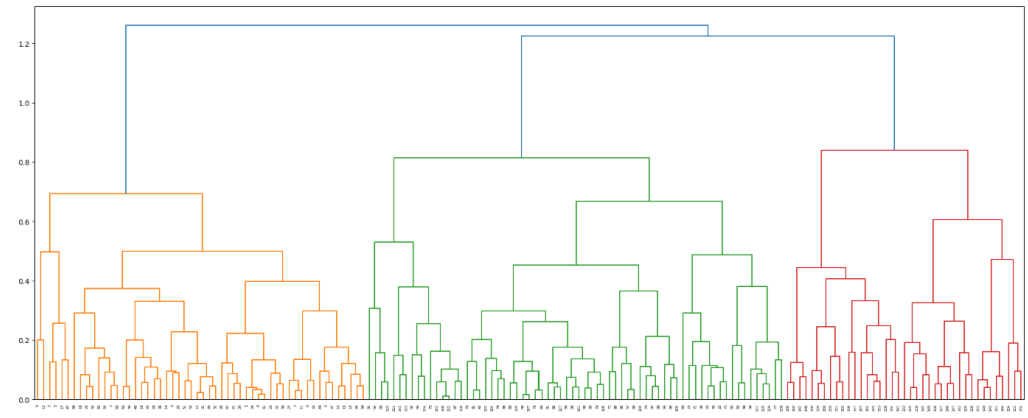
```
Z3_new = linkage(sub6_X,'average')
plt.figure(figsize=(25, 10))
dendrogram(Z3_new)
plt.show()
```



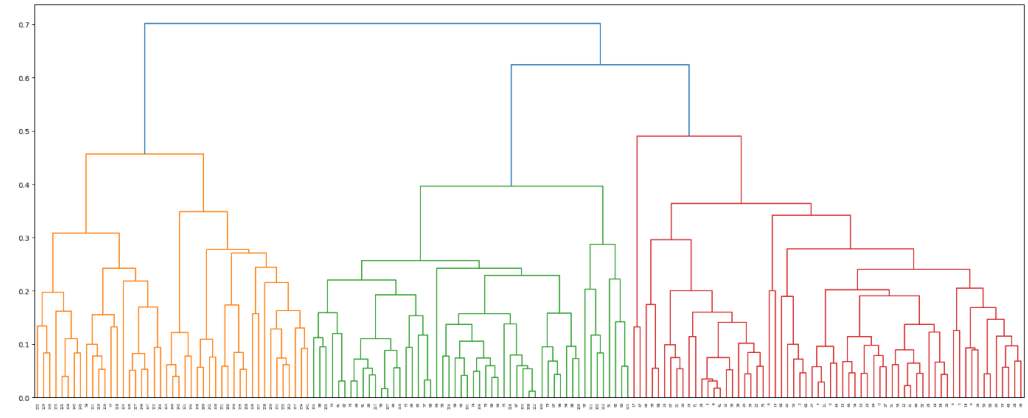
```
Z1_new2 = linkage(sub3_X,'single')
plt.figure(figsize=(25, 10))
dendrogram(Z1_new2)
plt.show()
```



```
Z2_new2 = linkage(sub3_X,'complete')
plt.figure(figsize=(25, 10))
dendrogram(Z2_new2)
plt.show()
```

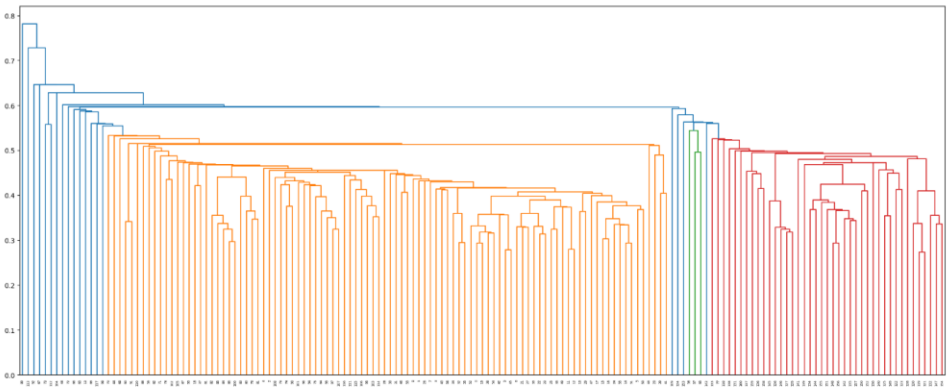


```
Z3_new2 = linkage(sub3_X,'average')
plt.figure(figsize=(25, 10))
dendrogram(Z3_new2)
plt.show()
```

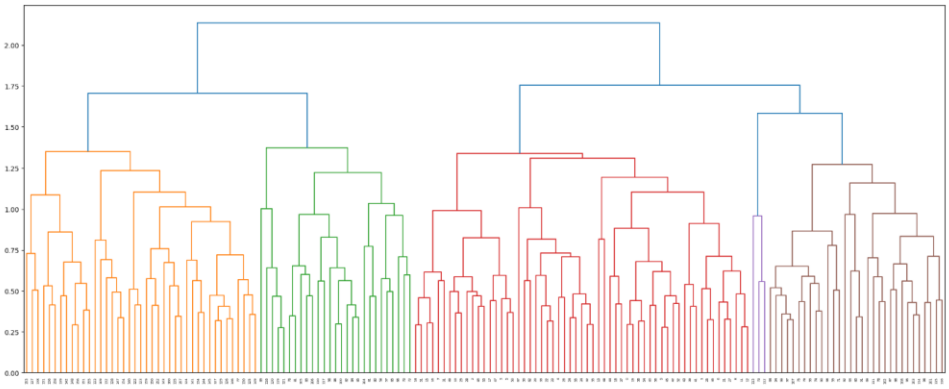


We can put the original dendrogram together for comparison:

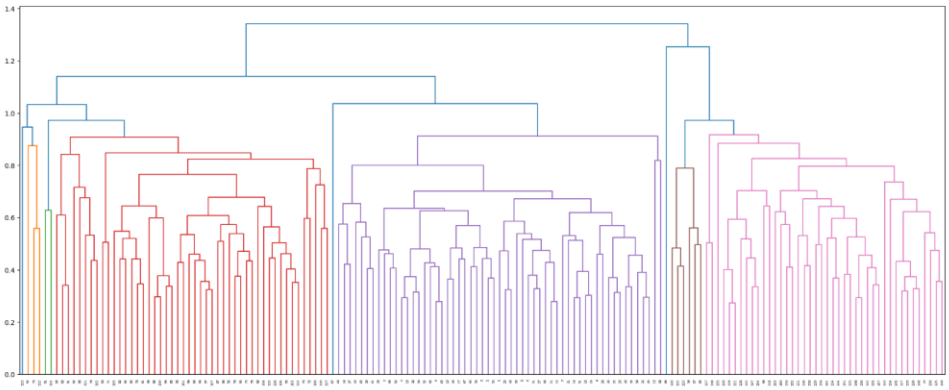
```
Z1 = linkage(X,'single')
plt.figure(figsize=(25, 10))
dendrogram(Z1)
plt.show()
```



```
Z2= linkage(X,'complete')
plt.figure(figsize=(25, 10))
dendrogram(Z2)
plt.show()
```



```
Z3 = linkage(X,'average')
plt.figure(figsize=(25, 10))
dendrogram(Z3)
plt.show()
```

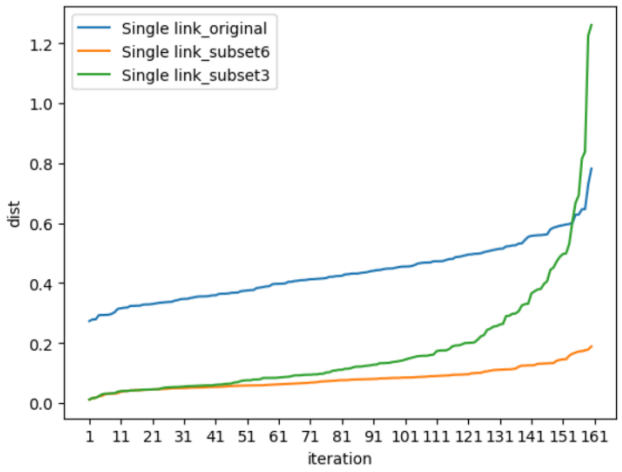


Just by observing the y-axis on dendrogram, we can see that the merge distance is shorter compared to the original one. The smaller the subset of attributes, the shorter the merge distance. Considering the merge size, we can see that the merge size of the clusters for the new dataset is smaller. We can further validate these claims by looking at the real numerical data.

Effect on Merge Distance

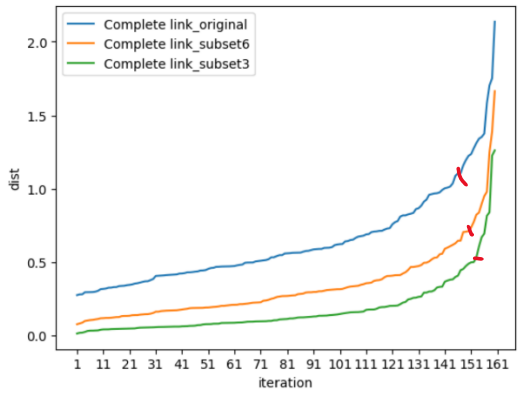
We plot all the merge distances obtained by the recent hierarchical clustering. We then compare with those obtained from the original hierarchical clustering.

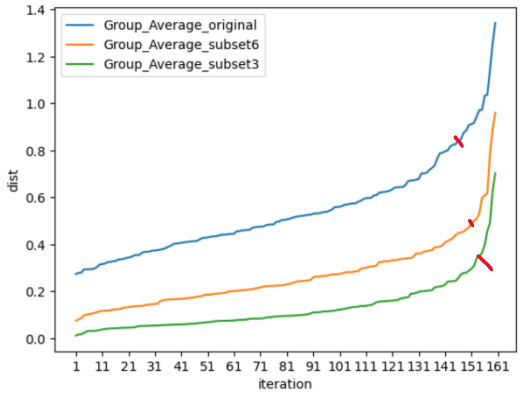
```
import matplotlib.pyplot as plt
import numpy as np
li_x=[]
for i in range(160):
    li_x.append(i+1)

li_y_dist1 = Z1[:,2].tolist()   #Z1[:,2] just mean all the distances
li_y_dist2 = Z1_new[:,2].tolist()
li_y_dist2 = Z1_new2[:,2].tolist()
plt.plot(li_x, li_y_dist1,label='Single link_original')  # Plot the chart
plt.plot(li_x, li_y_dist2,label='Single link_subset6')
plt.plot(li_x, li_y_dist3,label='Single link_subset3')
plt.xlabel('iteration')
plt.ylabel('dist')
plt.xticks(range(1,170,10 ))
plt.legend()
plt.show()
```



```python
import matplotlib.pyplot as plt
import numpy as np
li_x=[]
for i in range(160):
    li_x.append(i+1)

li_y_dist1 = Z2[:,2].tolist()   #Z1[:,2] just mean all the distances
li_y_dist2 = Z2_new[:,2].tolist()
li_y_dist3 = Z2_new2[:,2].tolist()
plt.plot(li_x, li_y_dist1,label='Complete link_original')  # Plot the chart
plt.plot(li_x, li_y_dist2,label='Complete link_subset6')
plt.plot(li_x, li_y_dist3,label='Complete link_subset3')
plt.xlabel('iteration')
plt.ylabel('dist')
plt.xticks(range(1,170,10 ))
plt.legend()
plt.show()
```

```python
import matplotlib.pyplot as plt
import numpy as np
li_x=[]
for i in range(160):
    li_x.append(i+1)

li_y_dist1 = Z3[:,2].tolist()   #Z1[:,2] just mean all the distances
li_y_dist2 = Z3_new[:,2].tolist()
li_y_dist3 = Z3_new2[:,2].tolist()
plt.plot(li_x, li_y_dist1,label='Group_Average_original')  # Plot the chart
plt.plot(li_x, li_y_dist2,label='Group_Average_subset6')
plt.plot(li_x, li_y_dist3,label='Group_Average_subset3')
plt.xlabel('iteration')
plt.ylabel('dist')
plt.xticks(range(1,170,10 ))
plt.legend()
plt.show()
```





Obviously, we can see that all the **merge distances are dropped**. **The smaller the subset of attributes, the greater the drop in merge distance.** Also, for complete link and group average, the slope of those distance obtained using subset attributes are flatter compared to distance obtained using full set of attributes. The turning point of the line exists in a later iteration (annotated in red). The smaller the subset, the later the turning point is. It implies that **the increases in merge distances is slower** when we are extracting useful attributes. The clustering algorithm can find points with shorter distances, and therefore, able to perform a **smoother merging with fewer big jumps in merge distance**. This is also shown on the graph: the big jumps are found after 140th iterations if we use the original dataset, it is only found after 150th iterations if we use the subset_6. The turning point is delayed. It means that the optimal clustering solution can be achieved in a later iteration.

We can validate these claims by looking at the merging distance for every point:

```
#find distance of merge for each point
li_dist1=[]
for i in range(161): #total 161 points
    li=Z1[:,0].tolist() + Z1[:,1].tolist()
    idx=li.index(i)
    if(idx>=160): #idx 0-159 is pt1
        idx=idx-160
    dist=Z1[:,2].tolist()[idx]
    li_dist1.append(dist)


li_dist2=[]
for i in range(161): #total 161 points
    li=Z1_new[:,0].tolist() + Z1_new[:,1].tolist()
    idx=li.index(i)
    if(idx>=160): #idx 0-159 is pt1
        idx=idx-160
    dist=Z1_new[:,2].tolist()[idx]
    li_dist2.append(dist)

li_dist2_5=[]
for i in range(161): #total 161 points
    li=Z1_new2[:,0].tolist() + Z1_new2[:,1].tolist()
    idx=li.index(i)
    if(idx>=160): #idx 0-159 is pt1
        idx=idx-160
    dist=Z1_new2[:,2].tolist()[idx]
    li_dist2_5.append(dist)

li_dist3=[]
for i in range(161): #total 161 points
    li=Z2[:,0].tolist() + Z2[:,1].tolist()
    idx=li.index(i)
    if(idx>=160): #idx 0-159 is pt1
        idx=idx-160
    dist=Z2[:,2].tolist()[idx]
    li_dist3.append(dist)

li_dist4=[]
for i in range(161): #total 161 points
    li=Z2_new[:,0].tolist() + Z2_new[:,1].tolist()
    idx=li.index(i)
    if(idx>=160): #idx 0-159 is pt1
        idx=idx-160
    dist=Z2_new[:,2].tolist()[idx]
    li_dist4.append(dist)

li_dist4_5=[]
for i in range(161): #total 161 points
    li=Z2_new2[:,0].tolist() + Z2_new2[:,1].tolist()
    idx=li.index(i)
    if(idx>=160): #idx 0-159 is pt1
        idx=idx-160
    dist=Z2_new2[:,2].tolist()[idx]
    li_dist4_5.append(dist)

li_dist5=[]
for i in range(161): #total 161 points
    li=Z3[:,0].tolist() + Z3[:,1].tolist()
    idx=li.index(i)
    if(idx>=160): #idx 0-159 is pt1
        idx=idx-160
    dist=Z3[:,2].tolist()[idx]
    li_dist5.append(dist)

li_dist6=[]
for i in range(161): #total 161 points
    li=Z3_new[:,0].tolist() + Z3_new[:,1].tolist()
    idx=li.index(i)
    if(idx>=160): #idx 0-159 is pt1
        idx=idx-160
    dist=Z3_new[:,2].tolist()[idx]
    li_dist6.append(dist)

li_dist7=[]
for i in range(161): #total 161 points
    li=Z3_new2[:,0].tolist() + Z3_new2[:,1].tolist()
    idx=li.index(i)
    if(idx>=160): #idx 0-159 is pt1
        idx=idx-160
    dist=Z3_new2[:,2].tolist()[idx]
    li_dist7.append(dist)
import csv

header = ['Point Idx','Single Link (original)', 'Single Link (Subset6)', 'Single Link (Subset3)'
         ,'Complete Link (original)', 'Complete Link (Subset6)','Complete Link (Subset3)'
         ,'Group Average (original)', 'Group Average  (Subset6)','Group Average  (Subset3)']
data=[]
for i in range(161):
    data .append([i,li_dist1[i],li_dist2[i],li_dist2_5[i],li_dist3[i],li_dist4[i],
                  li_dist4_5[i],li_dist5[i],li_dist6[i],li_dist7[i]])

with open('distForpts_subset.csv', 'w', encoding='UTF8',newline='') as f:
    writer = csv.writer(f)

    # write the header
    writer.writerow(header)
    writer.writerows(data)
```

The result are shown below:

Point Idx	Single Link (original)	Single Link (Subset6)	Single Link (Subset3)	Complete Link (original)	Complete Link (Subset6)	Complete Link (Subset3)	Group Average (original)	Group Average (Subset6)	Group Average (Subset3)
0	0.293638	0.119701	0.05779	0.293638	0.169553	0.05779	0.293638	0.144627	0.05779
1	0.431889	0.131453	0.023616	0.512728	0.131453	0.039894	0.472309	0.131453	0.030865
2	0.455366	0.149076	0.046123	0.46906	0.167427	0.046123	0.462213	0.167427	0.046123
3	0.414047	0.207377	0.125396	0.452478	0.207377	0.125396	0.443522	0.207377	0.125396
4	0.459964	0.156307	0.017246	0.459964	0.290971	0.017246	0.459964	0.227604	0.017246
5	0.368292	0.178267	0.09554	0.368292	0.178267	0.125396	0.368292	0.178267	0.125396
6	0.329016	0.143589	0.063894	0.430799	0.143589	0.063894	0.379908	0.143589	0.063894
7	0.429477	0.209964	0.028856	0.56328	0.223332	0.028856	0.530438	0.222765	0.028856
8	0.436424	0.248065	0.145371	0.479742	0.248065	0.200283	0.51608	0.248065	0.200283
9	0.278463	0.170898	0.089364	0.278463	0.313467	0.089364	0.278463	0.245223	0.089364
10	0.363716	0.1233	0.044674	0.363716	0.1233	0.044674	0.363716	0.1233	0.044674
11	0.278891	0.230002	0.028856	0.278891	0.267764	0.028856	0.278891	0.262451	0.028856
12	0.278891	0.185366	0.022187	0.278891	0.185366	0.022187	0.278891	0.185366	0.022187
13	0.584851	0.248065	0.188453	0.817419	0.248065	0.200283	0.817419	0.248065	0.200283
14	0.293149	0.130033	0.090042	0.293149	0.130033	0.095207	0.293149	0.130033	0.092625
15	0.302857	0.188626	0.049342	0.302857	0.188626	0.084707	0.302857	0.188626	0.067024
16	0.302857	0.188626	0.057062	0.302857	0.188626	0.057062	0.302857	0.188626	0.057062
17	0.384753	0.221568	0.132657	0.44165	0.221568	0.132657	0.44165	0.221568	0.132657
18	0.420984	0.118881	0.048813	0.420984	0.118881	0.08361	0.420984	0.118881	0.066211
19	0.293638	0.110668	0.044674	0.293638	0.110668	0.044674	0.293638	0.110668	0.044674

20	0.489662	0.136079	0.044379	0.596116	0.136079	0.044379	0.581179	0.136079	0.044379
21	0.328066	0.146717	0.053815	0.328066	0.146717	0.053815	0.328066	0.146717	0.053815
22	0.313926	0.136079	0.058663	0.313926	0.136079	0.058663	0.313926	0.136079	0.058663
23	0.313926	0.146717	0.066525	0.313926	0.146717	0.087233	0.313926	0.146717	0.098391
24	0.334958	0.157669	0.04284	0.334958	0.169302	0.04284	0.334958	0.163485	0.04284
25	0.3245	0.155258	0.104074	0.3245	0.160335	0.11402	0.3245	0.157797	0.11402
26	0.431574	0.172565	0.091448	0.459964	0.172565	0.12306	0.459964	0.172565	0.123035
27	0.328066	0.143589	0.051473	0.328066	0.143589	0.063538	0.328066	0.143589	0.057505
28	0.454881	0.141822	0.068466	0.495998	0.141822	0.068466	0.475341	0.141822	0.068466
29	0.363716	0.130033	0.086719	0.363716	0.130033	0.089364	0.363716	0.130033	0.089364
30	0.447871	0.123675	0.053815	0.563548	0.123675	0.053815	0.537813	0.123675	0.053815
31	0.44609	0.260742	0.022187	0.56328	0.334642	0.022187	0.512744	0.332681	0.022187
32	0.29432	0.123675	0.051703	0.29432	0.123675	0.051703	0.29432	0.123675	0.051703
33	0.330119	0.131453	0.04284	0.407872	0.131453	0.04284	0.368996	0.131453	0.04284
34	0.359078	0.128832	0.031347	0.420336	0.140107	0.032122	0.389707	0.140107	0.034454
35	0.29432	0.10735	0.044379	0.29432	0.10735	0.044379	0.29432	0.10735	0.044379
36	0.3245	0.10735	0.082942	0.3245	0.10735	0.11412	0.3245	0.10735	0.101984
37	0.420984	0.110668	0.05779	0.420984	0.110668	0.05779	0.420984	0.110668	0.05779
38	0.31567	0.203267	0.068466	0.31567	0.213714	0.068466	0.31567	0.213714	0.068466
39	0.404285	0.179438	0.049967	0.404285	0.185366	0.051703	0.404285	0.185366	0.051703
40	0.411095	0.130882	0.085773	0.411095	0.196293	0.141268	0.411095	0.17125	0.115269
41	0.404285	0.140107	0.017246	0.404285	0.140107	0.017246	0.404285	0.140107	0.017246
42	0.355751	0.167427	0.05989	0.423565	0.167427	0.139496	0.412704	0.167427	0.100027

43	0.510498	0.172565	0.062073	0.540379	0.172565	0.063894	0.525438	0.172565	0.063894
44	0.531599	0.196293	0.06287	0.588569	0.196293	0.095183	0.574342	0.20167	0.079027
45	0.278463	0.1233	0.057062	0.278463	0.1233	0.057062	0.278463	0.1233	0.057062
46	0.406701	0.177782	0.045765	0.406701	0.269752	0.045765	0.406701	0.223767	0.045765
47	0.398826	0.221568	0.132657	0.44165	0.221568	0.132657	0.44165	0.221568	0.132657
48	0.601001	0.236092	0.146102	0.817419	0.327815	0.290069	0.817419	0.327815	0.189349
49	0.372886	0.175707	0.045037	0.495542	0.24141	0.045037	0.427952	0.230783	0.045037
50	0.368292	0.207377	0.044074	0.368292	0.207377	0.044074	0.368292	0.207377	0.044074
51	0.293149	0.074717	0.057511	0.293149	0.074717	0.061696	0.293149	0.074717	0.061696
52	0.332036	0.118881	0.045037	0.423565	0.118881	0.045037	0.426263	0.118881	0.045037
53	0.406701	0.074717	0.061696	0.406701	0.074717	0.061696	0.406701	0.074717	0.061696
54	0.31567	0.213714	0.045765	0.31567	0.213714	0.045765	0.31567	0.213714	0.045765
55	0.333912	0.141822	0.051103	0.334958	0.141822	0.092247	0.334958	0.141822	0.071675
56	0.411095	0.178267	0.044074	0.411095	0.178267	0.044074	0.411095	0.178267	0.044074
57	0.495746	0.290629	0.03263	0.495746	0.29492	0.03263	0.495746	0.333558	0.03263
58	0.544035	0.207693	0.095457	0.575583	0.207693	0.102198	0.559809	0.272453	0.099377
59	0.505689	0.18564	0.054816	0.505689	0.18564	0.054816	0.505689	0.18564	0.054816
60	0.341174	0.185591	0.058698	0.341174	0.185591	0.058698	0.341174	0.185591	0.058698
61	0.46681	0.202455	0.05321	0.46681	0.202455	0.055832	0.628094	0.202455	0.054521
62	0.497574	0.225165	0.12636	0.563548	0.465376	0.156957	0.715923	0.372709	0.141658
63	0.587755	0.266753	0.046123	0.622202	0.327815	0.046123	0.609823	0.327815	0.046123
64	0.364562	0.22461	0.140866	0.518566	0.22461	0.290764	0.435973	0.22461	0.242343
65	0.495746	0.354353	0.168007	0.495746	0.354353	0.180089	0.495746	0.368874	0.174048

66	0.591157	0.324148	0.03263	0.706896	0.358627	0.03263	0.64361	0.341387	0.03263
67	0.646081	0.236301	0.062858	1.00693	0.409027	0.086065	1.036194	0.327891	0.074461
68	0.526132	0.18564	0.054816	0.778285	0.18564	0.054816	0.641654	0.18564	0.054816
69	0.560022	0.180289	0.037318	0.702993	0.180289	0.037318	0.675238	0.180289	0.037318
70	0.532659	0.213967	0.111664	0.596943	0.23516	0.128178	0.596943	0.224564	0.119921
71	0.487613	0.205341	0.110328	0.505689	0.236841	0.11402	0.505689	0.23023	0.11402
72	0.596943	0.234566	0.090254	0.596943	0.31358	0.093708	0.596943	0.273737	0.093708
73	0.557464	0.214365	0.111257	0.557464	0.299711	0.161446	0.557464	0.298493	0.137893
74	0.375518	0.079952	0.049273	0.375518	0.079952	0.049273	0.375518	0.079952	0.049273
75	0.396361	0.240941	0.058663	0.433536	0.266932	0.058663	0.433536	0.26397	0.058663
76	0.346273	0.096822	0.03081	0.346273	0.096822	0.03081	0.346273	0.096822	0.03081
77	0.523078	0.21361	0.124533	0.569483	0.236841	0.13267	0.542288	0.24154	0.13267
78	0.435148	0.099962	0.043601	0.435148	0.099962	0.069171	0.435148	0.099962	0.059735
79	0.423048	0.117714	0.058126	0.433536	0.117714	0.078688	0.433536	0.117714	0.068407
80	0.364457	0.164371	0.042013	0.46681	0.164371	0.042013	0.440819	0.164371	0.042013
81	0.346273	0.102941	0.030949	0.346273	0.102941	0.030949	0.346273	0.102941	0.030949
82	0.355361	0.159187	0.030949	0.412804	0.17542	0.030949	0.440759	0.167303	0.030949
83	0.397928	0.23452	0.049015	0.469352	0.337222	0.055999	0.440759	0.324783	0.055999
84	0.324608	0.164371	0.03081	0.337331	0.164371	0.03081	0.337331	0.164371	0.03081
85	0.337331	0.222696	0.111935	0.337331	0.263516	0.121319	0.337331	0.222696	0.116627
86	0.296857	0.115195	0.07318	0.296857	0.115195	0.07318	0.296857	0.115195	0.07318
87	0.472702	0.118386	0.051163	0.508595	0.13126	0.060506	0.508595	0.124823	0.059953
88	0.508595	0.180289	0.069171	0.508595	0.180289	0.069171	0.508595	0.180289	0.094588

89	0.781617	0.371117	0.093708	1.002492	0.948513	0.093708	1.254228	0.88973	0.093708
90	0.375518	0.142149	0.078139	0.375518	0.151616	0.078139	0.375518	0.146883	0.078139
91	0.341174	0.185591	0.178297	0.341174	0.185591	0.306591	0.341174	0.185591	0.222008
92	0.646559	0.191128	0.124508	0.968631	0.191128	0.150678	0.875047	0.191128	0.202964
93	0.468733	0.099962	0.037318	0.546366	0.099962	0.037318	0.537357	0.099962	0.037318
94	0.420955	0.206756	0.074449	0.471979	0.206756	0.11672	0.459195	0.222696	0.095585
95	0.376905	0.180718	0.041527	0.428841	0.186633	0.041527	0.402873	0.209175	0.041527
96	0.423682	0.196109	0.016158	0.42894	0.200151	0.016158	0.42894	0.200151	0.016158
97	0.323814	0.197639	0.042907	0.323814	0.22461	0.042907	0.323814	0.22461	0.042907
98	0.554219	0.193923	0.042907	0.639923	0.193923	0.042907	0.597657	0.193923	0.042907
99	0.35017	0.193923	0.064069	0.359231	0.193923	0.075035	0.3547	0.193923	0.073792
100	0.296857	0.170479	0.084991	0.296857	0.206756	0.11075	0.296857	0.190027	0.094988
101	0.42894	0.125129	0.041527	0.42894	0.143101	0.041527	0.42894	0.134115	0.041527
102	0.435148	0.117739	0.083505	0.435148	0.117739	0.083505	0.435148	0.117739	0.083505
103	0.352016	0.132474	0.049479	0.352016	0.132474	0.058392	0.352016	0.132474	0.053936
104	0.628094	0.202455	0.091095	0.77107	0.202455	0.097093	0.628094	0.202455	0.111943
105	0.473413	0.200151	0.094588	0.60104	0.200151	0.137558	0.546912	0.200151	0.094588
106	0.397569	0.079952	0.049273	0.469352	0.079952	0.049273	0.460498	0.079952	0.049273
107	0.323814	0.102941	0.016158	0.323814	0.102941	0.016158	0.323814	0.102941	0.016158
108	0.439523	0.130801	0.011343	0.559895	0.309048	0.011343	0.563227	0.16607	0.011343
109	0.593028	0.115195	0.07318	0.689881	0.115195	0.07318	0.725341	0.115195	0.07318
110	0.514463	0.137061	0.055999	0.558673	0.186286	0.055999	0.558673	0.161673	0.055999
111	0.43395	0.117739	0.08703	0.58707	0.117739	0.147792	0.532054	0.117739	0.117411

112	0.557464	0.117714	0.011343	0.557464	0.117714	0.011343	0.557464	0.117714	0.011343
113	0.728259	0.205588	0.083505	0.957913	0.279768	0.083505	0.94557	0.265599	0.083505
114	0.352016	0.108662	0.078139	0.352016	0.160281	0.078139	0.352016	0.137394	0.078139
115	0.412793	0.191128	0.058698	0.44437	0.191128	0.058698	0.44437	0.191128	0.058698
116	0.443064	0.132474	0.060506	0.44437	0.132474	0.060506	0.44437	0.132474	0.080075
117	0.558673	0.096822	0.042013	0.558673	0.096822	0.042013	0.558673	0.096822	0.042013
118	0.522205	0.279897	0.078981	0.642054	0.445088	0.13267	0.64361	0.370939	0.13267
119	0.273295	0.121228	0.052429	0.273295	0.121228	0.052429	0.273295	0.121228	0.052429
120	0.336941	0.121228	0.052429	0.466205	0.121228	0.052429	0.401573	0.121228	0.052429
121	0.273295	0.163954	0.067911	0.273295	0.207693	0.086711	0.273295	0.204217	0.077311
122	0.41297	0.263825	0.094197	0.41297	0.28147	0.094197	0.41297	0.28147	0.094197
123	0.41297	0.137002	0.057967	0.41297	0.137002	0.065046	0.41297	0.137002	0.061506
124	0.578921	0.295138	0.175645	0.68765	0.295138	0.18892	0.669037	0.295138	0.218574
125	0.354397	0.170042	0.039864	0.354397	0.170042	0.039864	0.354397	0.170042	0.039864
126	0.328397	0.254231	0.051257	0.328397	0.354353	0.051257	0.328397	0.333558	0.051257
127	0.318183	0.197843	0.051257	0.318183	0.211788	0.051257	0.318183	0.211788	0.051257
128	0.480392	0.211788	0.039864	0.494011	0.211788	0.039864	0.518315	0.211788	0.039864
129	0.318183	0.192774	0.083465	0.318183	0.192774	0.083465	0.318183	0.192774	0.083465
130	0.472663	0.192774	0.083465	0.476487	0.192774	0.083465	0.474575	0.192774	0.083465
131	0.49944	0.254481	0.078131	0.531165	0.493698	0.078131	0.530915	0.438841	0.078131
132	0.355302	0.172367	0.09376	0.580541	0.260871	0.157325	0.444846	0.216619	0.134646
133	0.525569	0.294468	0.130452	0.809114	0.343966	0.191615	0.884008	0.597692	0.161827
134	0.367586	0.191231	0.087171	0.367586	0.292697	0.091832	0.367586	0.243672	0.091832

135	0.343273	0.116024	0.061018	0.343273	0.116024	0.061018	0.343273	0.116024	0.061018
136	0.41471	0.222228	0.157603	0.41471	0.222228	0.157603	0.41471	0.222228	0.157603
137	0.45334	0.222228	0.157603	0.503114	0.222228	0.157603	0.503114	0.222228	0.157603
138	0.503114	0.306081	0.171576	0.503114	0.370528	0.200831	0.503114	0.388105	0.215761
139	0.468344	0.218725	0.05307	0.468344	0.218725	0.05307	0.496725	0.272066	0.05307
140	0.45674	0.159959	0.039472	0.510367	0.188864	0.039472	0.483554	0.174411	0.039472
141	0.480142	0.116024	0.079342	0.562043	0.116024	0.091832	0.521093	0.116024	0.091832
142	0.347266	0.137115	0.075701	0.468344	0.137115	0.075701	0.408299	0.137115	0.075701
143	0.559523	0.137002	0.039472	0.672478	0.137002	0.039472	0.734909	0.137002	0.039472
144	0.367586	0.227455	0.078131	0.367586	0.28147	0.078131	0.367586	0.28147	0.078131
145	0.386764	0.2193	0.065446	0.471006	0.292756	0.083443	0.432162	0.281326	0.083443
146	0.323747	0.168096	0.053188	0.328397	0.168096	0.053188	0.328397	0.168096	0.053188
147	0.335946	0.168096	0.053188	0.335946	0.168096	0.053188	0.335946	0.168096	0.053188
148	0.29295	0.16079	0.075701	0.29295	0.283753	0.075701	0.29295	0.206173	0.075701
149	0.354397	0.170042	0.083443	0.354397	0.170042	0.083443	0.354397	0.170042	0.083443
150	0.40986	0.175204	0.071107	0.40986	0.218725	0.097376	0.40986	0.203015	0.084242
151	0.368016	0.085429	0.058534	0.383355	0.085429	0.058534	0.383355	0.085429	0.058534
152	0.40986	0.200145	0.064631	0.40986	0.200145	0.083235	0.40986	0.200145	0.073933
153	0.562499	0.200145	0.061018	0.727623	0.200145	0.061018	0.621698	0.200145	0.061018
154	0.335946	0.265584	0.094197	0.335946	0.295138	0.094197	0.335946	0.295138	0.094197
155	0.383355	0.085429	0.058534	0.383355	0.085429	0.058534	0.383355	0.085429	0.058534
156	0.29295	0.137115	0.056441	0.29295	0.137115	0.056441	0.29295	0.137115	0.056441
157	0.343273	0.175432	0.098561	0.343273	0.204103	0.155984	0.343273	0.189768	0.130179

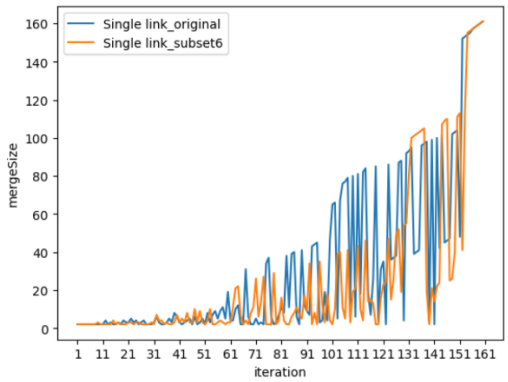
158	0.41471	0.104599	0.05307	0.41471	0.104599	0.05307	0.41471	0.104599	0.05307
159	0.448296	0.104599	0.100149	0.574432	0.104599	0.144687	0.496725	0.104599	0.128083
160	0.497997	0.164897	0.056441	0.531291	0.166097	0.056441	0.57307	0.165497	0.056441
AVERAGE	0.415976	0.173963	0.068821	0.46007	0.199296	0.080109	0.454181	0.194589	0.076565

We can see that on average, the merge distance for every single point dropped after taking subset of attributes. For example, in the single link approach, each point has a merged distance of 0.42 on average. It drops to 0.17 after we take the 6 sub-attributes. It further drops to 0.07 after we take the 3 sub-attributes. **The fewer the number of attributes taken, the greater the drop in merge distance.** This is also the same case for complete link and group average. The cluster algorithm can be said to be running more efficiently.

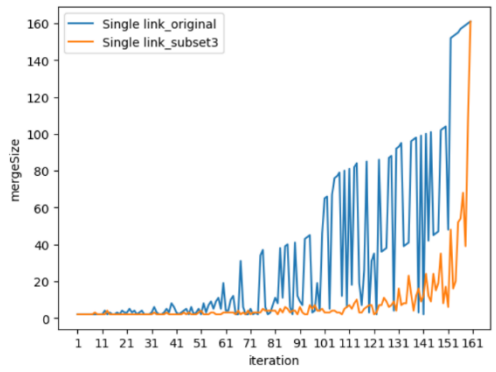
Effect on Merge Size

Next, we are going to compare the two hierarchical structures in term of merge size. We directly plot the two list of merge size for every approach:

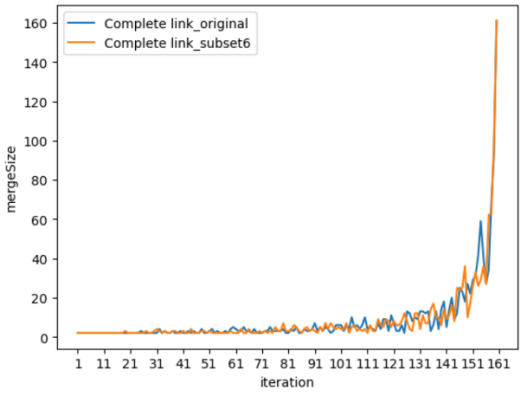
```
import matplotlib.pyplot as plt
import numpy as np
li_x=[]
for i in range(160):
    li_x.append(i+1)

li_y_ms1 = Z1[:,3].tolist()   #Z1[:,2] just mean all the distances
li_y_ms2 = Z1_new[:,3].tolist()

plt.plot(li_x, li_y_ms1,label='Single link_original')  # Plot the chart
plt.plot(li_x, li_y_ms2,label='Single link_subset6')
plt.xlabel('iteration')
plt.ylabel('mergeSize')
plt.xticks(range(1,170,10 ))
plt.legend()
plt.show()
```



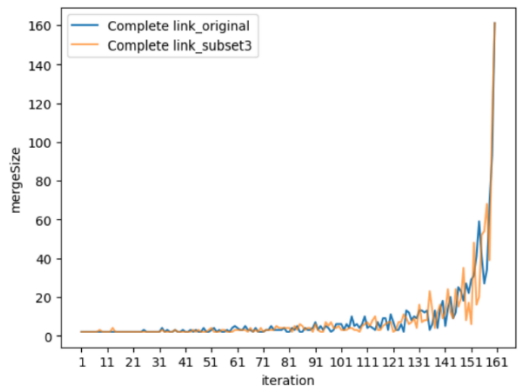

```
import matplotlib.pyplot as plt
import numpy as np
li_x=[]
for i in range(160):
    li_x.append(i+1)

li_y_ms1 = Z1[:,3].tolist()   #Z1[:,2] just mean all the distances
li_y_ms2 = Z1_new[:,3].tolist()
li_y_ms2 = Z1_new2[:,3].tolist()
plt.plot(li_x, li_y_ms1,label='Single link_original')  # Plot the chart
#plt.plot(li_x, li_y_ms2,label='Single link_subset6')
plt.plot(li_x, li_y_ms3,label='Single link_subset3')
plt.xlabel('iteration')
plt.ylabel('mergeSize')
plt.xticks(range(1,170,10 ))
plt.legend()
plt.show()
```



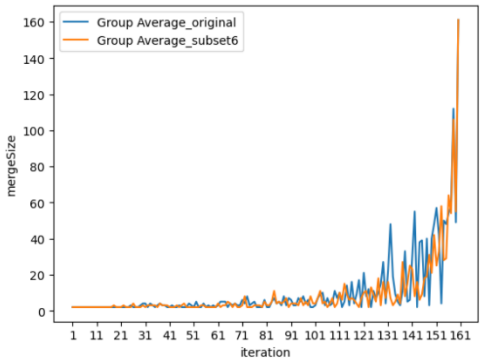

```
import matplotlib.pyplot as plt
import numpy as np
li_x=[]
for i in range(160):
    li_x.append(i+1)

li_y_ms1 = Z2[:,3].tolist()   #Z1[:,2] just mean all the distances
li_y_ms2 = Z2_new[:,3].tolist()

plt.plot(li_x, li_y_ms1,label='Complete link_original')  # Plot the chart
plt.plot(li_x, li_y_ms2,label='Complete link_subset6')
plt.xlabel('iteration')
plt.ylabel('mergeSize')
plt.xticks(range(1,170,10 ))
plt.legend()
plt.show()
```



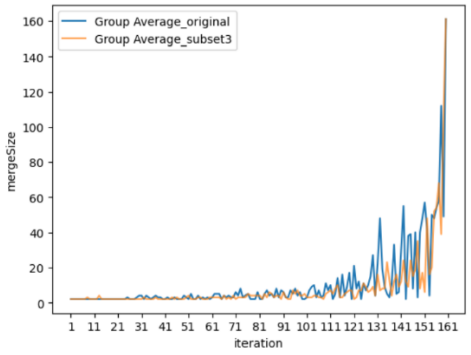

```
import matplotlib.pyplot as plt
import numpy as np
li_x=[]
for i in range(160):
    li_x.append(i+1)

li_y_ms1 = Z2[:,3].tolist()   #Z1[:,2] just mean all the distances
li_y_ms2 = Z2_new[:,3].tolist()
li_y_ms3 = Z2_new2[:,3].tolist()
plt.plot(li_x, li_y_ms1,label='Complete link_original')  # Plot the chart
#plt.plot(li_x, li_y_ms2,label='Complete link_subset6')
plt.plot(li_x, li_y_ms3,label='Complete link_subset3',alpha=0.7)
plt.xlabel('iteration')
plt.ylabel('mergeSize')
plt.xticks(range(1,170,10 ))
plt.legend()
plt.show()
```



```python
import matplotlib.pyplot as plt
import numpy as np
li_x=[]
for i in range(160):
    li_x.append(i+1)

li_y_ms1 = Z3[:,3].tolist()   #Z1[:,2] just mean all the distances
li_y_ms2 = Z3_new[:,3].tolist()

plt.plot(li_x, li_y_ms1,label='Group Average_original')  # Plot the chart
plt.plot(li_x, li_y_ms2,label='Group Average_subset6')
plt.xlabel('iteration')
plt.ylabel('mergeSize')
plt.xticks(range(1,170,10 ))
plt.legend()
plt.show()
```




```python
import matplotlib.pyplot as plt
import numpy as np
li_x=[]
for i in range(160):
    li_x.append(i+1)

li_y_ms1 = Z3[:,3].tolist()   #Z1[:,2] just mean all the distances
li_y_ms2 = Z3_new[:,3].tolist()
li_y_ms2 = Z3_new2[:,3].tolist()
plt.plot(li_x, li_y_ms1,label='Group Average_original')  # Plot the chart
#plt.plot(li_x, li_y_ms2,label='Group Average_subset6')
plt.plot(li_x, li_y_ms3,label='Group Average_subset3',alpha=0.7)
plt.xlabel('iteration')
plt.ylabel('mergeSize')
plt.xticks(range(1,170,10 ))
plt.legend()
plt.show()
```



From the graph obtained, we can see that by using the subset of attributes, the line for the merge size **is less fluctuated.** It means that by extracting subset of attributes, the distances among all the records can be reduced, and each point is more likely to find a suitable cluster in early iteration. This avoids merging between a very big cluster and a small one. We are less likely to be merging cluster that shouldn't be merged. The cluster solution can therefore be improved.

We can further investigate all the merge sizes for the 160 iterations:

We would consider Single Link (Full Attributes) and Single Link (6 Attributes) in following case study.

```
li_pt1_Z1=Z1[:,0].tolist()
li_pt2_Z1=Z1[:,1].tolist()
li_pt1_Z1_new=Z1_new[:,0].tolist()
li_pt2_Z1_new=Z1_new[:,1].tolist()

li_mergeSize=[]
for i in range(160):
    a1=1
    b1=1
    a2=1
    b2=1

    if li_pt1_Z1[i] >=161:
        idx=li_pt1_Z1[i]
        a1=Z1[int(idx-161),3]
    if li_pt2_Z1[i] >=161:
        idx=li_pt2_Z1[i]
        b1=Z1[int(idx-161),3]
    if li_pt1_Z1_new[i] >=161:
        idx=li_pt1_Z1_new[i]
        a2=Z1_new[int(idx-161),3]
    if li_pt2_Z1_new[i] >=161:
        idx=li_pt2_Z1_new[i]
        b2=Z1_new[int(idx-161),3]

    li_mergeSize.append([i+1,a1,b1,a2,b2])
import csv

header = ['Merge Step','ClusterA MergeSize(Single Link)',
          'ClusterB MergeSize(Single Link)',
          'ClusterA MergeSize(Single Link_new)',
          'ClusterB MergeSize(Single Link_new)',]


with open('SizeForMerges_subset.csv', 'w', encoding='UTF8',newline='') as f:
    writer = csv.writer(f)

    # write the header
    writer.writerow(header)
    writer.writerows(li_mergeSize)
```

The result are displayed as follows:

Since we have 161 points, Lets define cluster size X as follows:

If X<10, small cluster

If 50>X>=10, medium cluster

If X>=50, large cluster

We would highlight the size with Red, Green and Blue according to the size.

Merge Step	ClusterA MergeSize(Single Link)	ClusterB MergeSize(Single Link)	ClusterA MergeSize(Single Link_new)	ClusterB MergeSize(Single Link_new)
1	1	1	1	1
2	1	1	1	1
3	1	1	1	1
4	1	1	1	1
5	1	1	1	1
6	1	1	1	1
7	1	1	1	1
8	1	1	1	1
9	1	1	1	2
10	1	1	1	1
11	1	1	1	1

12	2	2	1	1
13	1	1	1	1
14	1	2	1	1
15	1	1	1	3
16	1	1	1	1
17	1	2	1	2
18	1	1	1	1
19	1	3	1	1
20	1	2	1	1
21	1	2	1	2
22	1	4	1	2
23	1	2	1	1
24	1	3	1	2
25	1	1	1	2
26	1	2	1	1
27	1	3	1	1
28	1	1	1	1
29	1	1	1	1
30	1	1	1	2
31	1	2	1	1
32	2	4	4	3
33	1	2	2	2
34	1	1	1	3
35	1	1	1	1
36	1	2	1	2
37	1	4	1	1
38	1	2	1	1
39	5	3	1	2
40	3	3	4	3
41	1	2	1	2
42	1	1	1	4
43	1	2	1	2
44	1	3	5	3
45	2	3	1	2
46	1	1	1	2
47	1	5	2	8
48	1	1	1	2
49	1	2	2	7
50	2	3	1	2
51	1	1	1	1
52	6	2	1	3
53	1	2	1	9
54	1	6	1	1
55	1	8	1	1
56	1	4	1	2
57	2	7	1	3

58	6	5	1	2
59	1	4	1	1
60	8	11	1	2
61	1	3	1	2
62	1	3	1	10
63	1	9	10	11
64	2	10	1	21
65	1	1	4	3
66	1	1	1	1
67	19	12	1	3
68	3	3	1	1
69	1	1	1	7
70	1	1	3	8
71	1	4	22	4
72	1	1	3	3
73	1	2	3	11
74	1	1	1	26
75	3	31	1	1
76	3	34	1	1
77	1	5	1	1
78	1	1	2	27
79	1	2	1	1
80	1	6	1	3
81	9	2	14	2
82	1	7	2	2
83	1	37	1	1
84	3	8	1	1
85	1	38	3	3
86	1	39	2	6
87	1	5	7	4
88	1	1	1	6
89	1	40	1	4
90	1	11	1	16
91	5	4	1	8
92	1	6	29	5
93	2	41	1	1
94	1	43	1	7
95	1	44	1	1
96	1	2	1	34
97	2	2	11	2
98	12	7	1	2
99	1	3	1	17
100	1	45	1	3
101	19	46	1	1
102	1	65	1	9
103	1	4	4	35

104	1	66	1	39
105	9	67	1	10
106	1	76	1	4
107	2	77	1	40
108	1	11	1	2
109	1	79	1	18
110	1	5	2	19
111	1	80	2	41
112	12	6	2	8
113	1	81	1	3
114	2	82	3	43
115	1	18	1	13
116	1	6	5	10
117	19	7	2	11
118	1	84	1	1
119	1	2	1	1
120	5	26	1	15
121	4	31	2	21
122	1	1	1	23
123	1	85	1	46
124	1	35	1	14
125	1	36	2	24
126	1	37	2	47
127	1	86	3	49
128	1	87	3	16
129	1	3	2	52
130	88	4	1	54
131	1	92	26	55
132	2	93	19	81
133	1	38	1	100
134	1	39	1	101
135	1	40	1	102
136	1	95	1	103
137	1	96	1	104
138	1	97	4	15
139	1	2	1	1
140	1	98	2	19
141	1	1	1	13
142	1	99	1	21
143	1	41	2	22
144	1	100	2	105
145	3	42	2	107
146	1	45	1	109
147	1	46	1	24
148	1	101	1	25
149	1	102	14	26

150	1	103	1	110
151	1	47	2	111
152	104	48	1	40
153	1	152	1	113
154	1	153	41	114
155	1	154	1	155
156	2	155	1	156
157	1	157	1	157
158	1	158	1	158
159	1	159	1	159
160	1	160	1	160
# large clusters	34		22	
# medium clusters	37		48	
# small clusters	249		250	

We can see that the cluster process provided by the subset of attributes are much smoother compared to the original one. In the original single link hierarchical clustering, there are more large cluster being merged with smaller clusters. This can be illustrated through the red highlighted cells starting from iteration 102. At this stage, the same algorithm using subset attributes is still merging on medium size cluster with smaller size cluster. This shows that when using subset of attributes, we can obtain smaller distance among all the points, therefore it would **slower the formation of very large-sized cluster in early iterations.**

To further validate this claim, we can also check the solution for k=80 under single link hierarchical clustering using subset of 3 attributes and subset of 6 attributes:

```python
kclusters1 = fcluster(Z1, 80, criterion='maxclust')
kclusters2 = fcluster(Z1_new,80, criterion='maxclust')
kclusters3 = fcluster(Z1_new2,80, criterion='maxclust')
print(kclusters1)
print(kclusters2)
print(kclusters3)
li_1=np.zeros(80)
li_2=np.zeros(80)
li_3=np.zeros(80)
for i in range(80):
    for j in range(161):
        if i+1==kclusters1[j]:
            li_1[i]+=1
        if i+1==kclusters2[j]:
            li_2[i]+=1
        if i+1==kclusters3[j]:
            li_3[i]+=1
import statistics

print('full attribute'+str(sorted(li_1)))
print('six attribute'+str(sorted(li_2)))
print('three attribute'+str(sorted(li_3)))
print(statistics.mean(li_1))
print(statistics.mean(li_2))
print(statistics.mean(li_3))
```

```
[17 20 25 17 26 17 17 18 21 17 17 17 17 45 17 17 17 17  6 17 36 17 17 17
 17 17 19 17 24 17 23 22 17 17 17 17 17  6 17 35 17 35 17 37 40 17 16 17
 75 17 17 17 17 16 17 17 17 48 50 33  3 27 32 46 10 49 47 77 39 44 41 31
 74  1  9 10  8 68  4  9  8  8  7  8  7  7  7 29 34 80  9  3 78 28 10 13
 10 10 42 10  7 11  5 13 76 30 13 10 12 73 38 14  2 79 13 13 15 43 67 62
 62 62 58 58 72 57 54 54 63 54 60 65 62 69 55 55 51 53 66 56 59 61 55 70
 55 54 54 62 55 57 55 55 55 71 62 55 55 55 51 52 64]
[51 59 59 52 59 57 50 63 26 60 60 66 59 27 59 58 58 28 59 51 59 56 59 56
 59 59 59 50 59 59 59 72 59 59 59 59 59 51 62 59 60 59 59 59 61 60 59 29
 69 59 53 59 59 59 64 59 57 74  7 54 25 30 65 73 49 79 78 70 54 41 32 55
 68 48 45 71 37 35 45 45 36 37 37 67 36 40 33 45 41 80 45 25 45 45 34 45
 38 46 42 43 33 45 44 45 31 39 45 37 45 33 37 44 45 47 45 45 45 37 24  4
  4  4 22 16 76  1  9  6  8  2  3 21  5 75 16 15 10 11 77 14 16 15 15 16
 20 19  5  5 15  1 13 12 17 18 23 12 15 15 13 13 15]
[57 54 55 70 54 66 51 51 74 63 57 51 59 80 64 57 57  3 58 57 53 54 53 54
 58 67 65 51 56 62 54 59 54 58 54 53 60 57 56 54 61 54 55 51 57 57 57  4
 75 59 52 59 59 59 57 55 52 46 43 50 71 32 72 55 73 76 46 54 50 36 45 68
 38 44 36 53 32 69 36 36 32 34 34 32 32 47 28 35 36 40 30 79 48 36 28 36
 33 36 36 36 37 36 25 35 39 41 36 33 35 28 32 27 35 26 31 71 35 32 29 42
 42 42 23  8 78 17 21 21 17 19 20  6 22 49  9 12  1  2 77 15  8 13 11  8
  7 17 21 21 11 18 15  5 12 12 24  5 10 14 15 16 10]
full attribute[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 3.0, 4.0, 5.0, 5.0, 5.0, 6.0,
7.0, 11.0, 37.0]
six attribute[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 2.0, 2.0,
2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 3.0, 3.0, 3.0, 3.0, 3.0, 4.0, 4.0, 6.0,
7.0, 16.0, 29.0]
three attribute[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.
0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0,
2.0, 2.0, 2.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 4.0, 4.0, 4.0, 5.0, 5.0, 6.0,
7.0, 10.0, 10.0, 12.0]
2.0125
2.0125
2.0125
```

We can see that by using subset of attributes, the number of points within every cluster would be more similar to each other. In order words, the **variance (dispersion) in the distribution of number of points is reduced**. From the last image, we can see that by setting k=80, the maximum number of points within one cluster is 37 when using full set of attributes. This number drops to 29 when we are using 6 attributes. It further drops to 12 when we are using 3 attributes. It means that **if we apply single link hierarchical clustering, we can avoid having very large clusters formed when choosing subset of attributes**. The rate of large cluster formation is reduced.

In the following, we would investigate about how taking subset of attributes would affect the merge size when using Complete Link and Group Average.

Complete Link

```
li_pt1_Z2=Z2[:,0].tolist()
li_pt2_Z2=Z2[:,1].tolist()
li_pt1_Z2_new=Z2_new[:,0].tolist()
li_pt2_Z2_new=Z2_new[:,1].tolist()
li_pt1_Z2_new2=Z2_new2[:,0].tolist()
li_pt2_Z2_new2=Z2_new2[:,1].tolist()

li_mergeSize=[]
for i in range(160):
    a1=1
    b1=1
    a2=1
    b2=1
    a3=1
    b3=1
    if li_pt1_Z2[i] >=161:
        idx=li_pt1_Z2[i]
        a1=Z2[int(idx-161),3]
    if li_pt2_Z2[i] >=161:
        idx=li_pt2_Z2[i]
        b1=Z2[int(idx-161),3]

    if li_pt1_Z2_new[i] >=161:
        idx=li_pt1_Z2_new[i]
        a2=Z2_new[int(idx-161),3]
    if li_pt2_Z2_new[i] >=161:
        idx=li_pt2_Z2_new[i]
        b2=Z2_new[int(idx-161),3]

    if li_pt1_Z2_new2[i] >=161:
        idx=li_pt1_Z2_new2[i]
        a3=Z2_new2[int(idx-161),3]
    if li_pt2_Z2_new2[i] >=161:
        idx=li_pt2_Z2_new2[i]
        b3=Z2_new2[int(idx-161),3]
    li_mergeSize.append([i+1,a1,b1,a2,b2,a3,b3])
import csv

header = ['Merge Step','ClusterA MergeSize(Complete Link)',
          'ClusterB MergeSize(Complete Link)',
          'ClusterA MergeSize(Complete Link_6subAttri)',
          'ClusterB MergeSize(Complete Link_6subAttri)',
          'ClusterA MergeSize(Complete Link_3subAttri)',
          'ClusterB MergeSize(Complete Link_3subAttri)',
         ]


with open('SizeForMerges_subset2.csv', 'w', encoding='UTF8',newline='') as f:
    writer = csv.writer(f)

    # write the header
    writer.writerow(header)
    writer.writerows(li_mergeSize)
```

The results are shown as below:

Similar as before, we would define cluster size X as follows:

If X<10, small cluster

If 50>X>=10, medium cluster

If X>=50, large cluster

Merge Step	ClusterA MergeSize(Complete Link)	ClusterB MergeSize(Complete Link)	ClusterA MergeSize(Complete Link_6subAttri)	ClusterB MergeSize(Complete Link_6subAttri)	ClusterA MergeSize(Complete Link_3subAttri)	ClusterB MergeSize(Complete Link_3subAttri)
1	1	1	1	1	1	1
2	1	1	1	1	1	1
3	1	1	1	1	1	1
4	1	1	1	1	1	1
5	1	1	1	1	1	1
6	1	1	1	1	1	1
7	1	1	1	1	1	1
8	1	1	1	1	1	2
9	1	1	1	1	1	1
10	1	1	1	1	1	1
11	1	1	1	1	1	1
12	1	1	1	1	1	1
13	1	1	1	1	1	3
14	1	1	1	1	1	1
15	1	1	1	1	1	1
16	1	1	1	1	1	1
17	1	1	1	1	1	1
18	1	1	1	1	1	1
19	1	1	1	2	1	1
20	1	1	1	1	1	1
21	1	1	1	1	1	1
22	1	1	1	1	1	1
23	1	1	1	1	1	1
24	1	1	1	1	1	1
25	1	2	1	1	1	1
26	1	1	1	1	1	1
27	1	1	1	2	1	1
28	1	1	1	1	1	1
29	1	1	1	1	1	1
30	1	1	1	2	1	1
31	1	1	1	3	1	1
32	2	2	1	2	1	2
33	1	1	1	1	1	1
34	1	2	1	2	1	1
35	1	1	1	1	1	1
36	1	1	1	1	1	1
37	1	2	1	2	1	2
38	1	1	1	2	1	1
39	1	1	1	1	1	1
40	1	2	1	1	1	1
41	1	1	1	2	1	1

42	1	1	1	1	1	1
43	1	2	1	1	1	1
44	1	1	2	2	1	2
45	1	2	1	1	1	1
46	1	1	1	1	1	2
47	1	1	1	1	1	1
48	2	2	1	2	1	1
49	1	1	1	2	1	1
50	1	1	1	1	1	2
51	1	2	1	2	2	2
52	2	2	1	1	1	1
53	1	1	1	1	1	1
54	1	2	1	1	1	1
55	1	1	1	1	1	2
56	1	1	1	1	1	2
57	1	2	1	1	1	1
58	1	1	1	1	1	1
59	2	2	1	2	1	1
60	1	4	1	1	1	2
61	1	3	1	1	1	2
62	1	2	1	1	1	2
63	1	2	2	2	1	2
64	2	3	1	1	1	2
65	1	2	1	1	1	1
66	1	2	2	2	2	2
67	1	1	1	1	1	1
68	1	3	1	1	1	2
69	1	1	1	1	1	1
70	1	1	1	2	2	2
71	1	1	1	1	1	1
72	1	2	1	2	1	2
73	1	2	1	1	1	2
74	1	4	1	2	1	2
75	1	2	1	1	1	2
76	1	2	2	3	3	2
77	1	2	1	2	1	3
78	1	2	1	2	2	2
79	2	2	3	4	2	2
80	1	1	1	2	2	2
81	1	1	1	2	1	3
82	1	3	1	2	1	1
83	1	2	3	3	1	4
84	2	3	2	3	1	2
85	1	1	1	2	2	4
86	1	1	1	1	2	3
87	1	3	1	3	1	2

88	1	2	1	4	1	3
89	1	2	1	2	2	2
90	1	3	1	3	1	1
91	3	4	1	2	2	4
92	1	2	1	1	1	2
93	2	3	2	3	1	1
94	1	2	1	2	1	1
95	2	3	4	3	4	3
96	1	3	2	2	3	2
97	1	1	1	6	3	4
98	1	2	3	2	1	3
99	4	2	2	2	1	3
100	2	4	1	4	1	4
101	3	3	1	3	1	2
102	1	2	2	2	1	2
103	1	5	2	5	1	2
104	1	3	1	1	2	2
105	5	5	2	3	1	3
106	2	3	3	3	1	2
107	2	4	1	2	1	2
108	1	3	1	3	1	1
109	2	4	1	2	3	2
110	4	6	2	2	1	5
111	1	3	1	1	3	4
112	1	4	3	3	2	3
113	1	3	1	2	3	5
114	1	2	1	2	3	7
115	2	5	2	7	1	2
116	2	2	2	4	1	2
117	3	6	2	3	1	4
118	5	4	5	4	4	2
119	1	2	3	5	2	5
120	4	7	2	3	2	5
121	3	4	5	3	1	1
122	1	2	1	5	1	2
123	1	2	2	4	3	4
124	1	5	4	4	4	3
125	1	1	5	7	4	7
126	2	11	2	6	3	6
127	7	5	1	3	3	3
128	2	6	1	2	4	3
129	6	4	3	9	3	6
130	3	6	3	9	2	2
131	10	3	2	2	6	10
132	3	10	5	6	4	3
133	3	9	1	6	1	7

134	4	9	4	3	1	7
135	1	2	2	12	7	16
136	3	3	5	12	5	8
137	5	8	3	7	1	3
138	1	3	7	3	5	7
139	4	10	4	2	7	9
140	6	12	6	8	2	7
141	1	4	5	4	5	7
142	1	13	4	7	8	16
143	7	13	8	8	3	9
144	3	6	2	6	3	6
145	3	9	8	17	11	13
146	13	12	14	10	6	9
147	3	20	11	14	4	15
148	4	14	12	24	23	12
149	2	25	3	7	5	3
150	13	9	6	11	8	9
151	6	23	16	10	2	4
152	13	18	8	25	24	24
153	14	27	9	17	4	12
154	18	41	4	25	12	8
155	12	29	10	26	35	17
156	5	22	1	26	6	48
157	3	31	33	29	16	52
158	41	27	36	27	19	20
159	59	34	36	62	68	39
160	68	93	63	98	54	107
#of large clusters	3		3		4	
#of medium clusters	27		23		19	
#of small clusters	290		294		297	

The merge size of the clusters constructed using subset of attributes are pretty similar to the original one. The only difference we can locate is that by taking a smaller subset of attributes, the number of small clusters formed would be greater.

This implies that we would have more smaller size clusters before we do the large-size-cluster merging. This make sense as taking useful subset of attributes just mean that the distance between records can be reduced. As a result, one point may have greater opportunity in locating a suitable cluster, which avoids it merging into the cluster it doesn't belong to. The size of each cluster can therefore be more evenly distributed.

For group average method, it is believed that the situation is similar. There would not be a very significant difference. But in general, the merge size would be smaller for the same reason.