Homework5

20200639 Chae Woojin

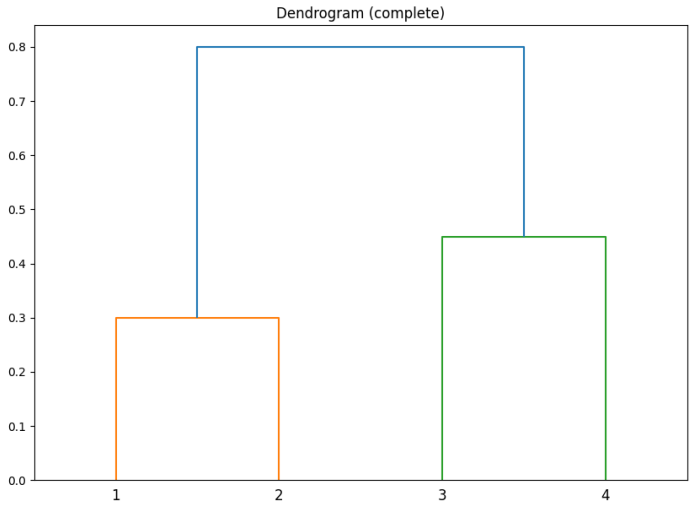
Chapter 12: 2, 3, 8, 9, 13

Chapter 13: 1, 2, 4, 6, 8

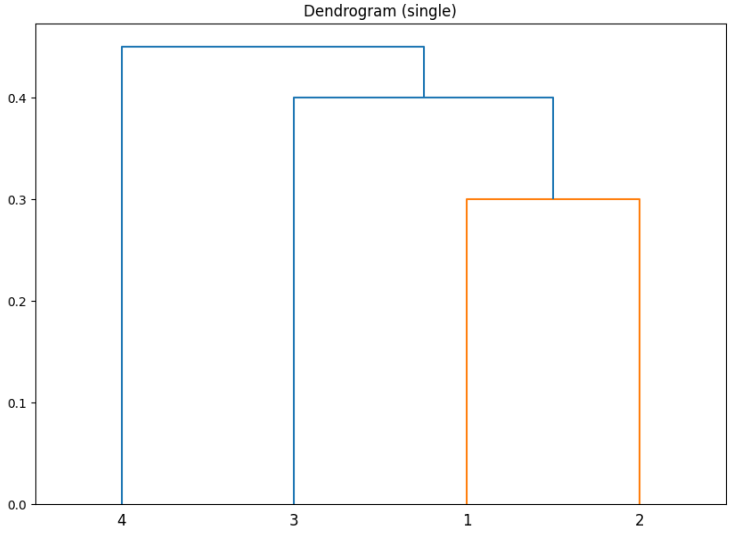
Chapter 12

2.

(a)



(b)



(c)

```
import numpy as np
from scipy.cluster.hierarchy import linkage, fcluster
import pandas as pd

# Cut the dendrogram to form two clusters
cluster_assignments = fcluster(Z_complete, 2, criterion='maxclust')

# Create a DataFrame for easy viewing
df = pd.DataFrame({
    'Observation': np.arange(1, 5),
    'Cluster': cluster_assignments
})

# Count the number of observations in each cluster
cluster_counts = df.groupby('Cluster')['Observation'].apply(list)
print(cluster_counts)
```

```
Cluster
1    [1, 2]
2    [3, 4]
Name: Observation, dtype: object
```

1st, 2nd observations are in the same cluster, and 3rd, 4th observations are in the same cluster.

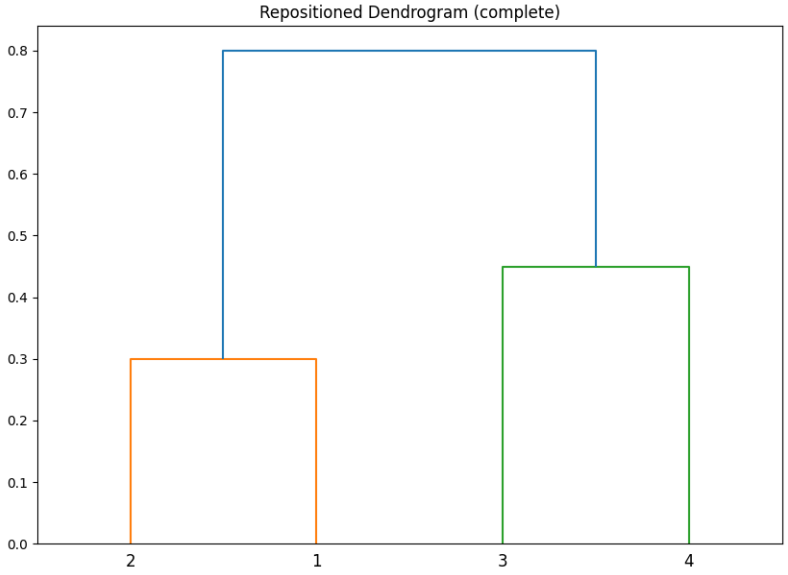
(d)

```
import numpy as np
from scipy.cluster.hierarchy import linkage, fcluster
import pandas as pd

# Cut the dendrogram to form two clusters
cluster_assignments = fcluster(Z_single, 2, criterion='maxclust')

# Create a DataFrame for easy viewing
df = pd.DataFrame({
    'Observation': np.arange(1, 5),
    'Cluster': cluster_assignments
})

# Count the number of observations in each cluster
cluster_counts = df.groupby('Cluster')['Observation'].apply(list)
print(cluster_counts)
```

```
Cluster
1    [1, 2, 3]
2          [4]
Name: Observation, dtype: object
```

1st, 2nd, and 3rd observations are grouped into a same cluster, and the 4th observation mapped to the other one.

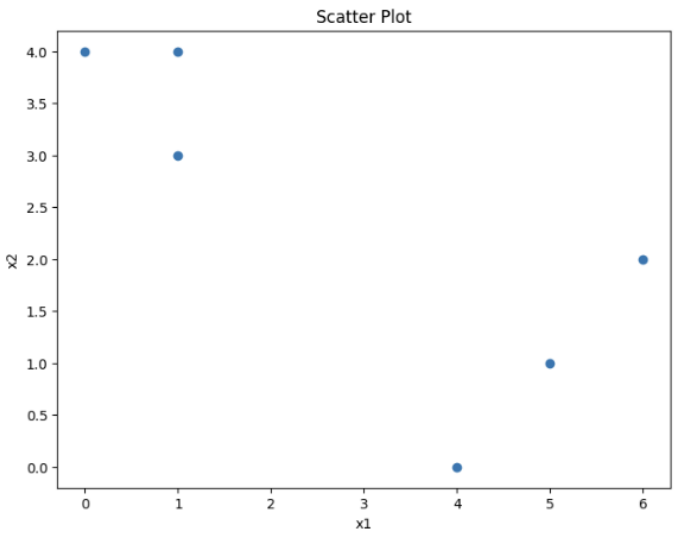
(e)



3.

(a)

```
import matplotlib.pyplot as plt
import pandas as pd

# Creating the data frame equivalent to the R code
d = pd.DataFrame({
    'x1': [1, 1, 0, 5, 6, 4],
    'x2': [4, 3, 4, 1, 2, 0]
})
d.index = d.index + 1

# Plotting the scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(d['x1'], d['x2'])
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Scatter Plot')
plt.show()
```



(b)

```
[67] # Setting the seed for reproducibility
     np.random.seed(42)

     # Assigning a random cluster (1 or 2) to each row
     d['cluster'] = np.random.choice([1, 2], size=len(d), replace=True)

     # Display the data frame
     print(d)
```

```
   x1  x2  cluster
1   1   4        1
2   1   3        2
3   0   4        1
4   5   1        1
5   6   2        1
6   4   0        2
```

1st, 3rd, 4th, and 5th observations are grouped into the same cluster, and the other observations are grouped into the other cluster.

(c)

```
[11] import pandas as pd
     import numpy as np

     # Calculating centroids for each cluster
     centroids = pd.DataFrame()
     for i in [1, 2]:
         centroids[i] = d[d['cluster'] == i][['x1', 'x2']].mean(axis=0)

     # Transposing the DataFrame for a format
     centroids = centroids.T

     # Display the centroids
     print(centroids)
```

```
    x1    x2
1  3.0  2.75
2  2.5  1.50
```

Centroids of two clusters are $(x1, x2) = (3.0, 2.75), (2.5, 1.50)$, respectively.

(d)

```
[69] # Function to calculate Euclidean distance
     def euclidean_distance(row, centroid):
         return np.sqrt((row['x1'] - centroid[0])**2 + (row['x2'] - centroid[1])**2)
```

```
[70] # Calculating distances from each centroid
     distances = pd.DataFrame()
     for i in centroids.index:
         distances[i] = d.apply(euclidean_distance, axis=1, centroid=centroids.loc[i])

     # Assigning the nearest cluster
     d['cluster'] = distances.idxmin(axis=1)

     # Display the updated DataFrame
     print(d)
```

```
   x1  x2  cluster
1   1   4        1
2   1   3        1
3   0   4        1
4   5   1        2
5   6   2        1
6   4   0        2
```

After new assignment, 1st, 2nd, 3rd, and 5th observations are grouped into same cluster, and 4th, and 6th observations are grouped into the same cluster.

(e)

```
[103] # Function to calculate Euclidean distance
      def euclidean_distance(row, centroid):
          return np.sqrt((row['x1'] - centroid['x1'])**2 + (row['x2'] - centroid['x2'])**2)
```

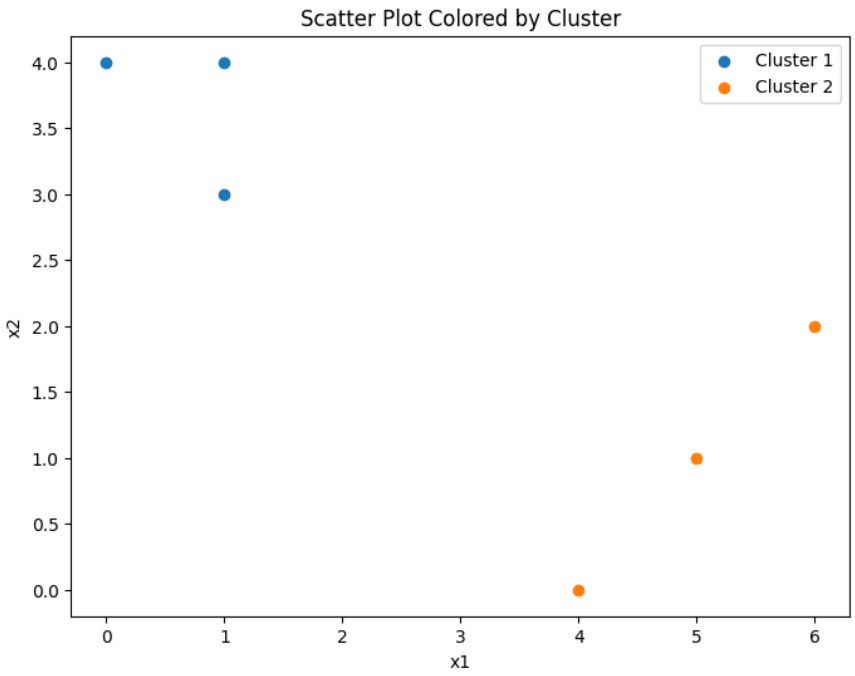
```
[113] def k_means_clustering(df, max_iterations=100):
          for _ in range(max_iterations):
              # Step 1: Compute the centroid for each cluster
              centroids = pd.DataFrame()
              for i in [1, 2]:
                  centroids[i] = df[df['cluster'] == i][['x1', 'x2']].mean(axis=0)
              centroids = centroids.T

              # Step 2: Assign each observation to the closest centroid
              distances = pd.DataFrame()
              for i in centroids.index:
                  distances[i] = df.apply(euclidean_distance, axis=1, centroid=centroids.loc[i])

              new_assignments = distances.idxmin(axis=1)

              # Check if cluster assignments have changed
              if new_assignments.equals(df['cluster']):
                  break

              df['cluster'] = new_assignments

          return df
```

```
# Running the k-means clustering process
d = k_means_clustering(d)

# Display the final cluster assignments
print(d)
```

```
   x1  x2  cluster
1   1   4        1
2   1   3        1
3   0   4        1
4   5   1        2
5   6   2        2
6   4   0        2
```

After equilibrium reached, one can see that 1st, 2nd, and 3rd observations are grouped into the same cluster, and the others are grouped into the same cluster.

(f)



8.

(a)

```
[19] import pandas as pd
     from sklearn.decomposition import PCA
     from sklearn.preprocessing import StandardScaler

     # Load your dataset
     df = pd.read_csv('./USArrests.csv', index_col='Unnamed: 0')

     # Standardizing the data
     scaler = StandardScaler()
     df_scaled = scaler.fit_transform(df)

     # Performing PCA
     pca = PCA()
     pca.fit(df_scaled)

     # Proportion of variance explained by each principal component
     explained_variance_ratio = pca.explained_variance_ratio_

     print(explained_variance_ratio)
```

```
[0.62006039 0.24744129 0.0891408  0.04335752]
```

PVE is [0.62006039, 0.24744129, 0.0891408, 0.04335752].

(b)

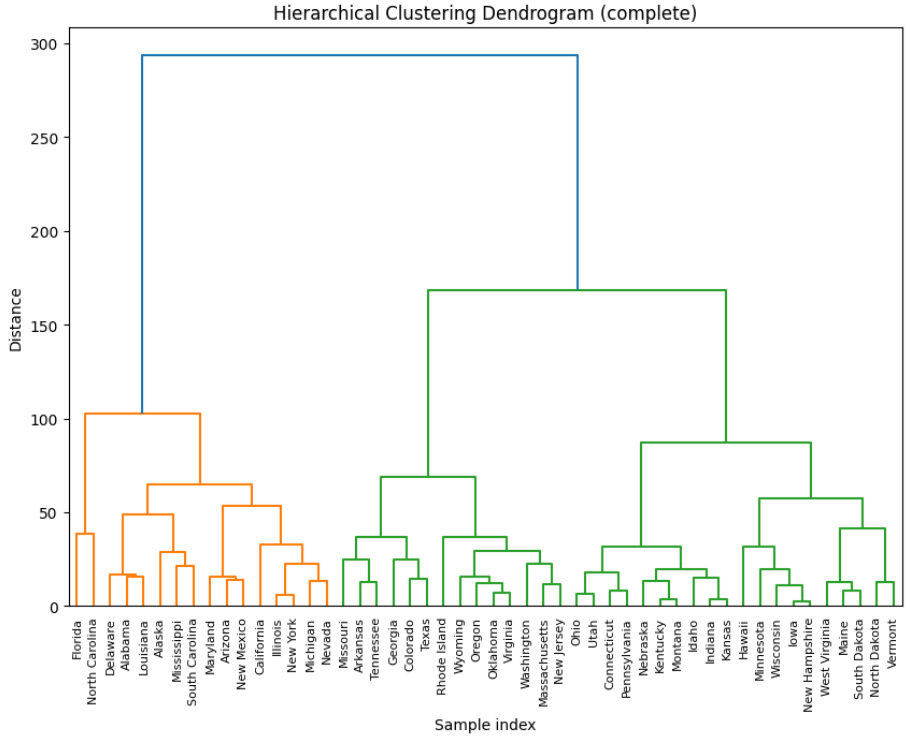
```
[20] # Performing PCA
     pca = PCA()
     pca.fit(df_scaled)
     transformed_data = pca.transform(df_scaled)

     # Calculating the proportion of total variance explained by each principal component
     variance_explained = np.sum(transformed_data**2, axis=0) / np.sum(df_scaled**2)

     print(variance_explained)
```

```
[0.62006039 0.24744129 0.0891408  0.04335752]
```

PVE is [0.62006039, 0.24744129, 0.0891408, 0.04335752] which is identical to the result of (a).

9.

(a)



(b)

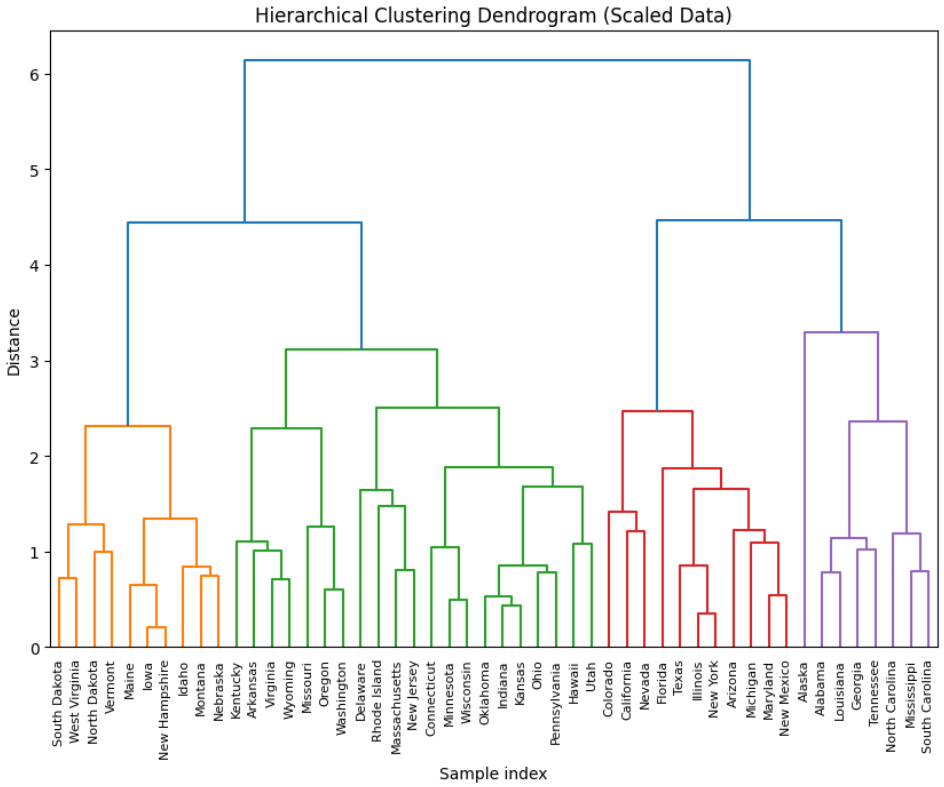
```
[22] import pandas as pd
     from scipy.cluster.hierarchy import fcluster

     # Cut the tree into 3 clusters
     cluster_assignments = fcluster(hc, 3, criterion='maxclust')

     # Create a DataFrame to map observations to their cluster assignments
     df_clusters = pd.DataFrame({'observation': df.index, 'cluster': cluster_assignments})

     # Group the observation names by cluster
     clusters = {i: df_clusters['observation'][df_clusters['cluster'] == i].tolist() for i in range(1, 4)}

     # Display the observations in each cluster
     for cluster, observations in clusters.items():
         print(f"Cluster {cluster}: {observations}")

Cluster 1: ['Alabama', 'Alaska', 'Arizona', 'California', 'Delaware', 'Florida', 'Illinois', 'Louisiana', 'Maryland', 'Michigan', 'Mississippi', 'Nevada', 'New Mexico', 'New York', 'North Carolina', 'South Carolina']
Cluster 2: ['Arkansas', 'Colorado', 'Georgia', 'Massachusetts', 'Missouri', 'New Jersey', 'Oklahoma', 'Oregon', 'Rhode Island', 'Tennessee', 'Texas', 'Virginia', 'Washington', 'Wyoming']
Cluster 3: ['Connecticut', 'Hawaii', 'Idaho', 'Indiana', 'Iowa', 'Kansas', 'Kentucky', 'Maine', 'Minnesota', 'Montana', 'Nebraska', 'New Hampshire', 'North Dakota', 'Ohio', 'Pennsylvania', 'South Dakota', 'Utah', 'Vermont', 'West Virginia', 'Wisconsin']
```

Cluster 1: ['Alabama', 'Alaska', 'Arizona', 'California', 'Delaware', 'Florida', 'Illinois', 'Louisiana', 'Maryland', 'Michigan', 'Mississippi', 'Nevada', 'New Mexico', 'New York', 'North Carolina', 'South Carolina']

Cluster 2: ['Arkansas', 'Colorado', 'Georgia', 'Massachusetts', 'Missouri', 'New Jersey', 'Oklahoma', 'Oregon', 'Rhode Island', 'Tennessee', 'Texas', 'Virginia', 'Washington', 'Wyoming']

Cluster 3: ['Connecticut', 'Hawaii', 'Idaho', 'Indiana', 'Iowa', 'Kansas', 'Kentucky', 'Maine', 'Minnesota', 'Montana', 'Nebraska', 'New Hampshire', 'North Dakota', 'Ohio', 'Pennsylvania', 'South Dakota', 'Utah', 'Vermont', 'West Virginia', 'Wisconsin']

(c)



(d)

```
[24] import pandas as pd
     from scipy.cluster.hierarchy import fcluster

     # Cut the tree into 3 clusters
     cluster_assignments = fcluster(hc, 3, criterion='maxclust')

     # Create a DataFrame to map observations to their cluster assignments
     df_clusters = pd.DataFrame({'observation': df.index, 'cluster': cluster_assignments})

     # Group the observation names by cluster
     clusters = {i: df_clusters['observation'][df_clusters['cluster'] == i].tolist() for i in range(1, 4)}

     # Display the observations in each cluster
     for cluster, observations in clusters.items():
         print(f"Cluster {cluster}: {observations}")

     Cluster 1: ['Alabama', 'Alaska', 'Arizona', 'California', 'Delaware', 'Florida', 'Illinois', 'Louisiana', 'Maryland', 'Michigan', 'Mississippi', 'Nevada', 'New Mexico', 'New York', 'North Carolina', 'South Carolina']
     Cluster 2: ['Arkansas', 'Colorado', 'Georgia', 'Massachusetts', 'Missouri', 'New Jersey', 'Oklahoma', 'Oregon', 'Rhode Island', 'Tennessee', 'Texas', 'Virginia', 'Washington', 'Wyoming']
     Cluster 3: ['Connecticut', 'Hawaii', 'Idaho', 'Indiana', 'Iowa', 'Kansas', 'Kentucky', 'Maine', 'Minnesota', 'Montana', 'Nebraska', 'New Hampshire', 'North Dakota', 'Ohio', 'Pennsylvania', 'South Dakota', 'Utah', 'Vermont', 'West Virginia', 'Wisconsin']
```

Cluster 1: ['Alabama', 'Alaska', 'Arizona', 'California', 'Delaware', 'Florida', 'Illinois', 'Louisiana', 'Maryland', 'Michigan', 'Mississippi', 'Nevada', 'New Mexico', 'New York', 'North Carolina', 'South Carolina']

Cluster 2: ['Arkansas', 'Colorado', 'Georgia', 'Massachusetts', 'Missouri', 'New Jersey', 'Oklahoma', 'Oregon', 'Rhode Island', 'Tennessee', 'Texas', 'Virginia', 'Washington', 'Wyoming']

Cluster 3: ['Connecticut', 'Hawaii', 'Idaho', 'Indiana', 'Iowa', 'Kansas', 'Kentucky', 'Maine', 'Minnesota', 'Montana', 'Nebraska', 'New Hampshire', 'North Dakota', 'Ohio', 'Pennsylvania', 'South Dakota', 'Utah', 'Vermont', 'West Virginia', 'Wisconsin']

One can see that scaling the variables have an effect on clustering results. Therefore, scaling ought to be informed by the unique attributes of the dataset under consideration. In the current scenario, the data fields are not uniform in terms of units (**Murder (Assault, Rape)**: **numeric number** of arrests per 100,000 people pertaining to murder (assault, rape) crime, **UrbanPop: numeric percent** urban population). Given this context, scaling the data prior to clustering seems to be a more appropriate approach.

13.

(a)

```
[25] df = pd.read_csv('./Ch12Ex13.csv', header=None)
     col_names = [f"H{i}" for i in range(1, 21)] + [f"D{i}" for i in range(1, 21)]
     df.columns = col_names
```

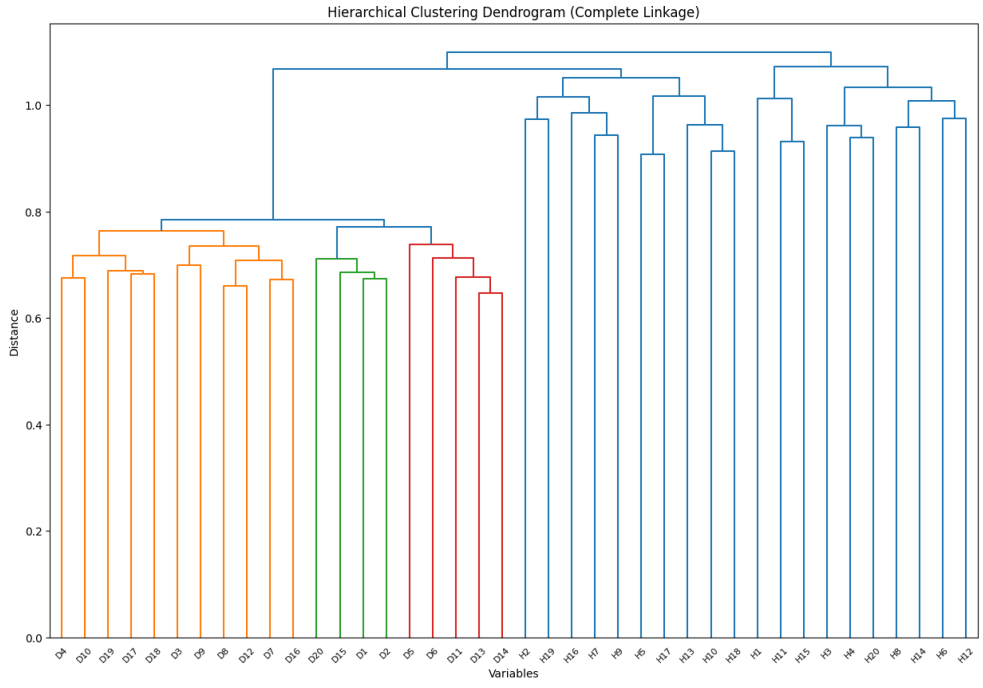
```
[26] df
```

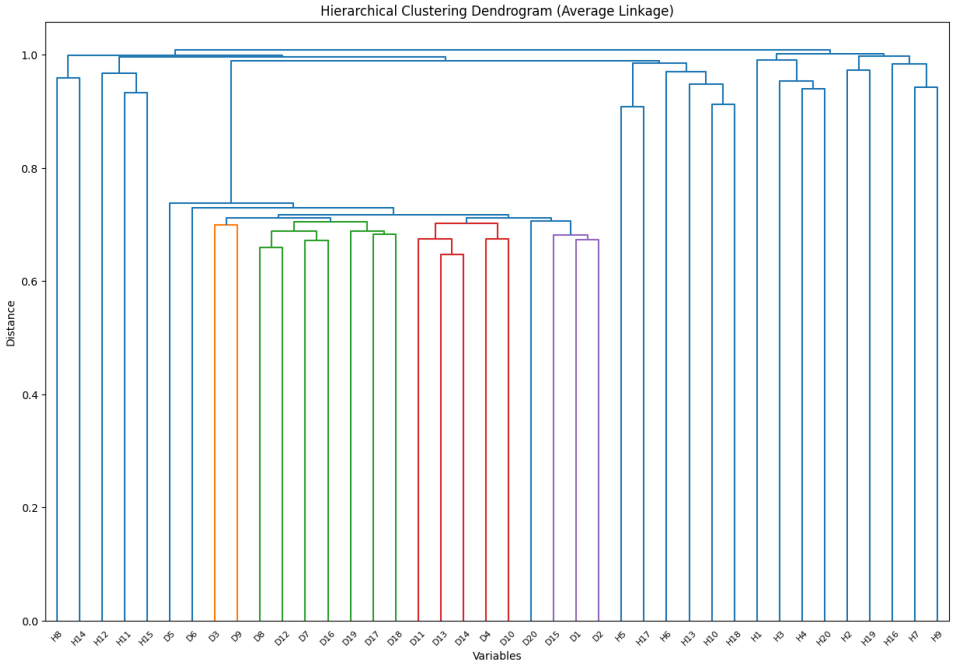
	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	...	D11	D12	D13	D14	D15	D16	D17	D18	D19	D20
0	-0.961933	0.441803	-0.975005	1.417504	0.818815	0.316294	-0.024967	-0.063966	0.031497	-0.350311	...	-0.509591	-0.216725	-0.055506	-0.484449	-0.521581	1.949135	1.324335	0.468147	1.061100	1.655970
1	-0.292526	-1.139267	0.195837	-1.281121	-0.251439	2.511997	-0.922206	0.059543	-1.409645	-0.656712	...	1.700708	0.007290	0.099062	0.563853	-0.257275	-0.581781	-0.169887	-0.542304	0.312939	-1.284377
2	0.258788	-0.972845	0.588486	-0.800258	-1.820398	-2.058924	-0.064764	1.592124	-0.173117	-0.121087	...	-0.615472	0.009999	0.945810	-0.318521	-0.117889	0.621366	-0.070764	0.401682	-0.016227	-0.526553
3	-1.152132	-2.213168	-0.861525	0.630925	0.951772	-1.165724	-0.391559	1.063619	-0.350009	-1.489058	...	-0.284277	0.198946	-0.091833	0.349628	-0.298910	1.513696	0.671185	0.010855	-1.043689	1.625275
4	0.195783	0.593306	0.282992	0.247147	1.978668	-0.871018	-0.989715	-1.032253	-1.109654	-0.385142	...	-0.692998	-0.845707	-0.177497	-0.166491	1.483155	-1.687946	-0.141430	0.200778	-0.675942	2.220611
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	1.075148	3.003267	-0.123441	-1.036740	-1.270604	-1.277029	-0.278504	1.249723	-0.706994	-0.704671	...	0.842161	-0.762154	0.546881	1.586981	-0.242043	0.507189	1.297424	0.314290	-1.513097	-0.074709
996	-1.226125	-0.501702	-0.717430	-0.169113	0.599530	-0.997987	0.028236	0.200508	-1.364865	0.564957	...	-1.012523	0.593252	-0.594506	-1.443559	-0.028870	0.052170	-0.867227	0.228531	-0.207759	-0.209665
997	-3.056328	0.449889	1.880362	-0.742841	2.238346	-0.291738	1.270233	0.696415	1.242857	0.429148	...	0.135085	-0.732077	-0.037468	-0.836689	0.020274	-0.803306	-0.907277	-0.781791	0.069908	1.336894
998	1.450658	1.310348	0.383837	-0.408860	-0.471111	-1.392396	-0.805808	0.210900	1.727079	0.862870	...	0.109018	-0.128522	0.860270	0.765015	-2.360090	0.252699	-1.461818	-0.812342	-1.095099	-1.460114
999	0.717977	0.763482	0.313576	-0.326473	-0.158700	0.468113	0.519161	-0.427099	2.759068	-2.571514	...	-1.105732	-0.876950	-0.298792	-0.576146	-1.196780	0.409031	-0.032282	-0.562064	-0.315811	0.192863

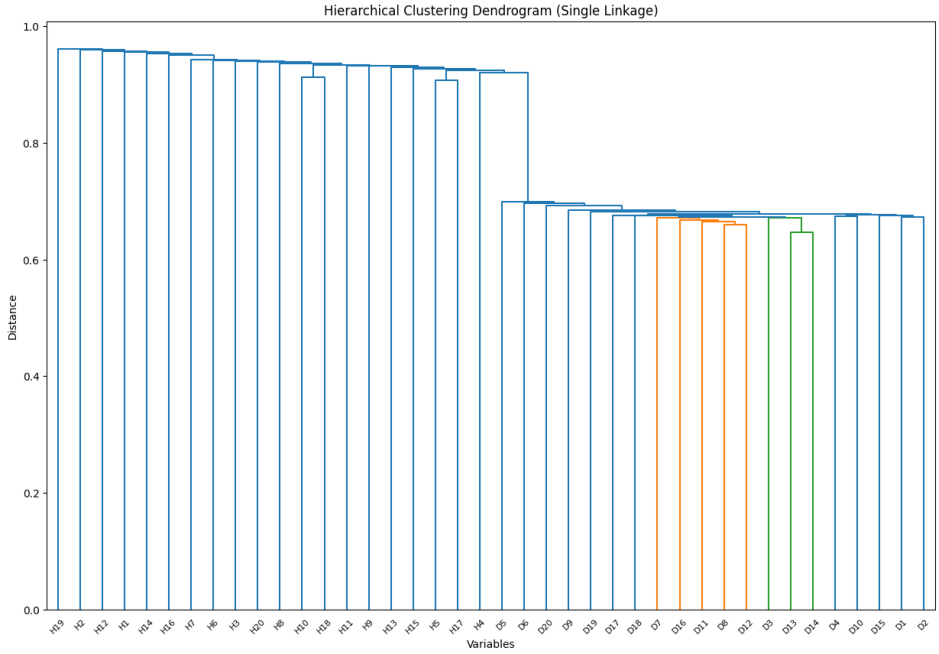
1000 rows × 40 columns

This code reads the whole raw data (1000 rows). Column H1~H20 represents samples from 20 healthy patients, and remaining columns (D1~D20) denotes the ones from 20 diseased group.

(b)







The samples are clearly divided into two distinct groups. However, it's crucial to understand that the degree of this division is affected by the selected linkage method. In particular, with average clustering, all disease samples cluster together, and they are encompassed within a portion of the healthy samples.

(c)

```
[30] from scipy.stats import ttest_ind
     from statsmodels.stats.multitest import multipletests

     # Create the class labels
     class_labels = np.repeat(["Healthy", "Diseased"], 20)

     # Perform t-tests for each row
     p_values = []
     for index, row in df.iterrows():
         group1 = row[class_labels == "Healthy"]
         group2 = row[class_labels == "Diseased"]
         t_stat, p_val = ttest_ind(group1, group2)
         p_values.append(p_val)

     # Adjust the p-values
     adjusted_pvals = multipletests(p_values, method='bonferroni')[1]

     # Find rows where adjusted p-value is less than 0.05
     significant_rows = np.where(adjusted_pvals < 0.05)[0]

     print("Significant rows:", significant_rows)
```

```
Significant rows: [ 10  11  12  13  14  15  16  17  18  19 500 501 502 503 504 505 506 507
 508 510 511 512 513 514 515 516 518 519 520 521 522 523 524 525 526 527
 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545
 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563
 564 565 566 567 568 569 570 571 573 574 575 576 577 578 579 580 581 582
 583 585 586 587 588 589 590 591 592 594 595 596 597 598 599]
```

The group comparison test provides significant genes that appear to give different results between the two groups. Test results which reveal p-values under 0.05 are deployed as above. One can conclude that corresponding genes to these row indices differ the most across the two groups.

Chapter 13

1.

(a) We expect α

(b) Let V represents the number of Type I errors when performing m independent null hypothesis tests, all of which are true. Then,

$$FWER(\alpha) = P(V \geq 1) = 1 - P(V = 0) = 1 - \Pr\left(\cap_{j=1}^{m} \{do\ not\ falsely\ reject\ H_{0j}\}\right)$$
$$= 1 - \prod_{j=1}^{m} \Pr\left(\{do\ not\ falsely\ reject\ H_{0j}\}\right) = 1 - (1-\alpha)^m$$

(c) If the p-values for the two tests are perfectly correlated, we can regard them as a single test. Therefore, we can expect FWER to be less than the case when two tests are independent. This can be explained mathematically, as $1 - (1-\alpha)^1 < 1 - (1-\alpha)^2$. In a similar manner, one can conclude that if two tests are positively correlated, FWER will be lower than $1 - (1-\alpha)^2$. This is because positively correlated tests have the effect of reducing the effective number of tests.

(d) If the p-values for the two tests are negatively correlated, the family-wise error rate will tend to be higher compared to the case where the tests are independent. This is because it leads to an increased likelihood of making Type I error in one or both tests, which raises the overall FWER. When the p-values are negatively correlated, it becomes more likely that one of the tests will produce a small p-value even if the null hypothesis is true for both tests. This is because the negative correlation increases the chance that one test will produce a significant result when the other does not. Consequently, the FWER is more likely to be elevated compared to the independent case ($1 - (1-\alpha)^2$).

2.

(a) A_j follows a Bernoulli distribution with probability α. (i.e, $A_j \sim Bernoulli(\alpha)$)

(b) $\sum_{j=1}^{m} A_j \sim B(m, \alpha)$

(c) Using the fact that binomial distribution $B(n,p)$ has standard deviation $\sqrt{np(1-p)}$,

the standard deviation of the number of Type 1 error($\sum_{j=1}^{m} A_j$'s) we will make is $\sqrt{m\alpha(1-\alpha)}$.

4.

(a) We should reject every null hypothesis with p-value under 0.05. Therefore, H_{01}, H_{02}, H_{03}, H_{08}, H_{09}, H_{10} should be rejected.

(b) If we want to control FWER at level 0.05, we can apply Bonferroni's method as a conservative approach. Using this approach, every null hypothesis with p-value under 0.005 will be rejected. Therefore, H_{01}, H_{09}, H_{10} should be rejected.

(c) Using Benjamini-Hochberg Procedure, we can control the FDR. As we want to control FDR at level 0.05, we have to find $L = \max\left\{j : p_{(j)} < \frac{0.05j}{10}\right\}$, where $p_{(j)}$'s are the sequence of p-values sorted in ascending order. In this case, it is {0.0009, 0.0011, 0.004, 0.006, 0.017, 0.031, 0.07, 0.11, 0.32, 0.90}. Then, simple evaluation gives $L = 5$. Finally, we have rejects null hypotheses having p-value $p_{(5)} = 0.0009$. Therefore, $H_{01}, H_{03}, H_{08}, H_{09}, H_{10}$ will be rejected.

(d) Similar to (c), using this procedure with different threshold 0.2 will reject H_{01}, $H_{02}, H_{03}, H_{05}, H_{07}, H_{08}, H_{09}, H_{10}$.

(e) 20% of these rejections are expected to be false positives.

6.

For each of the three panels in Figure 13.3, there are always 8 positives(red) and 2 negatives(black).

(a) Under Bonferroni procedure to control the FWER at level 0.05, classification result of each panel display can be summarized as follows.

Panel	FP	FN	TP	TN	Type 1 (FP)	Type 2 (FN)
1	0	1	7	2	0	1
2	0	1	7	2	0	1
3	0	5	3	2	0	5

(b) Under Holm procedure to control FWER at level 0.05, classification result of each panel display can be summarized as follows.

Panel	FP	FN	TP	TN	Type 1 (FP)	Type 2 (FN)
1	0	1	7	2	0	1
2	0	0	8	2	0	0
3	0	0	8	2	0	0

(c) False discovery rate is the expected ratio of false positives to total positives. As the number of false positives are 0 in every panel, false discovery rates are also 0.

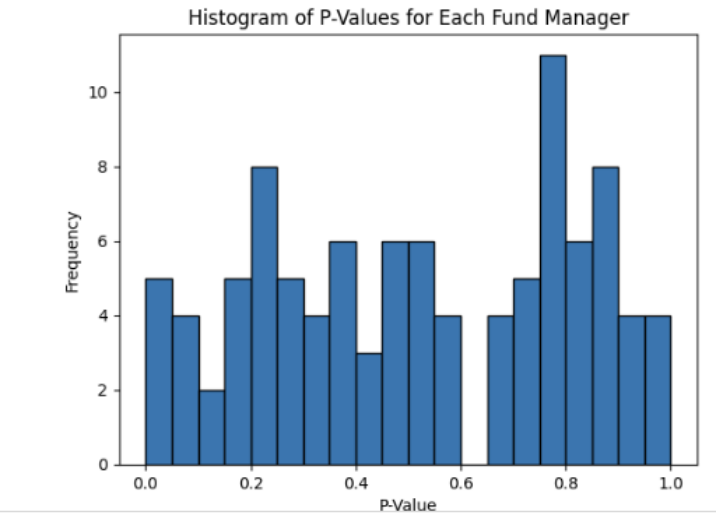
(d) False discovery rate for panel 1, 2, and 3 are 0 with the same reason in the case of (c).

(e) Although we lower the threshold of FWER at level 0.001, the number of false positives doesn't change. Therefore, the results would not change.

8.

(a)

```
[32] import numpy as np
     from scipy import stats
     import matplotlib.pyplot as plt

     # Setting the random seed and generating a sample data matrix
     rng = np.random.default_rng(5)
     n = 20  # Number of months that evaluates return
     m = 100  # Number of fund managers
     X = rng.normal(size=(n, m))  # Sample data matrix

     # Perform one-sample t-test for each fund manager
     p_values = np.array([stats.ttest_1samp(X[:, i], 0).pvalue for i in range(m)])

     # Plotting the histogram of the p-values
     plt.hist(p_values, bins=np.linspace(0, 1, 21), edgecolor='black')
     plt.xlabel('P-Value')
     plt.ylabel('Frequency')
     plt.title('Histogram of P-Values for Each Fund Manager')
     plt.show()
```



(b)

```
[33] (p_values < 0.05).sum()

     5
```

The number of rejections if we control Type 1 error at level 0.05 was 5.

(c)

```
[34] (p_values < 0.05/m).sum()

     0
```

The number of rejections if we control FWER at level 0.05 is 0.

(d)

```
[35] from statsmodels.stats.multitest import multipletests

     # Controlling the False Discovery Rate (FDR) at level 0.05
     fdr_adj_p_values = multipletests(p_values, alpha=0.05, method='fdr_bh')[1]
     fdr_rejections = np.sum(fdr_adj_p_values < 0.05)
     fdr_rejections

     0
```

The number of rejections if we control FDR at level 0.05 is 0.

(e)

```
[36] # Cherry-picking the 10 best-performing fund managers based on their mean returns
     top_10_indices = np.argsort(X.mean(axis=0))[-10:]

     # P-values for the top 10 fund managers
     top_10_p_values = p_values[top_10_indices]

     # Controlling the FWER for the top 10 fund managers using Bonferroni correction
     fwer_adj_p_values_top_10 = multipletests(top_10_p_values, alpha=0.05, method='bonferroni')[1]
     fwer_rejections_top_10 = np.sum(fwer_adj_p_values_top_10 < 0.05)

     # Controlling the FDR for the top 10 fund managers using Benjamini-Hochberg procedure
     fdr_adj_p_values_top_10 = multipletests(top_10_p_values, alpha=0.05, method='fdr_bh')[1]
     fdr_rejections_top_10 = np.sum(fdr_adj_p_values_top_10 < 0.05)

     fwer_rejections_top_10, fdr_rejections_top_10

     (1, 2)
```

The number of rejections under cherry-picked data if we control FWER, and FDR at level 0.05 are 1, and 2 respectively.

(f)

It could be misleading, because we selectively performed the test under cherry-picked data that are likely to be rejected. Therefore, results from this test cannot represent the whole data.