14. 군집분석

1. 군집분석의 이해

○ 군집분석의 특징

- 군집분석은 모집단 또는 범주에 대한 사전정보가 없는 경우에 사용되는 비지도학습법임.
- 군집분석의 목적은 각 군집내의 각 개체들은 다른 군집에 속하는 개체들보다 서로 더 유사하도록 주어진 개체들을 여러 군집들로 나누는 것임.
- 이렇게 나눈 군집들에 대하여 각 군집별 특성을 파악함으로써 전체 자료 구조에 대한 이해를 도움.
- 분류분석은 범주에 대한 사전정보가 있는 경우에 분류를 잘 할 수 있는 분류기를 구축하는데 그 목적이 있음.

- 군집분석은 사전정보가 없는 자료에 대한 탐색적 방법이므로 분석자의 주관에 따라 결과나 해석이 달라지는 특징을 가짐.
- 군집분석은 그 자체로서도 의미가 있지만 이상점 등의 특이값을 갖는 개체의 발견이나 결측값의 보정 등에도 활용될 수 있음.

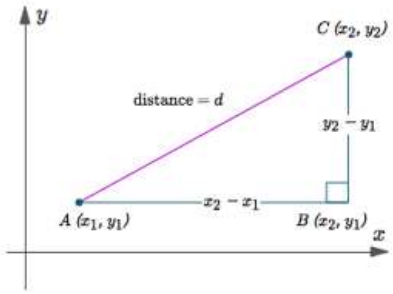
○ 거리

- 군집분석에서는 관측값들이 서로 얼마나 유사한지 또는 유사하지 않은지를 측정하는 측도가 필요하며, 보통 유사성보다는 비유사성을 이용하여 군집화를 진행함.
- 변수들이 연속형인 경우에는 거리들을 일반적으로 사용함.

1) 유클리드 거리: $d(x,y) = (\sum_{i=1}^{p}(x_i - y_i)^2)^{1/2}$

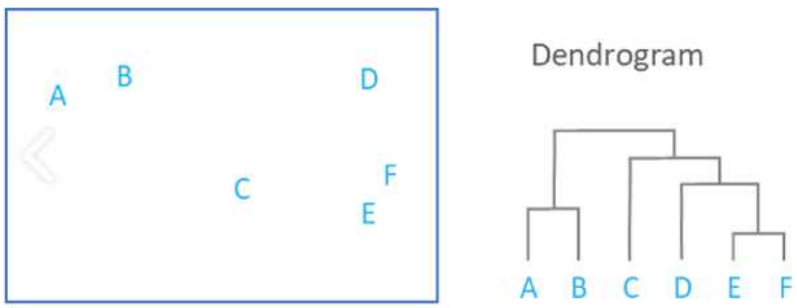
2) 민콥스키 거리: $d(x,y) = (\sum_{i=1}^{p}(x_i - y_i)^p)^{1/p}$

3) 맨해튼 거리: $d(x,y) = \sum_{i=1}^{p} | x_i - y_i |$



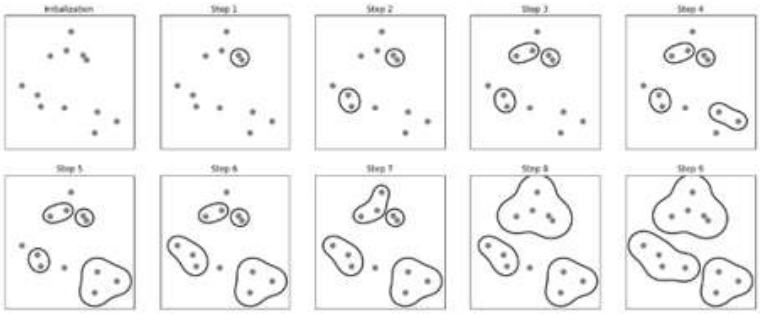
○ 계층적 군집분석

- 계층적 군집방법에는 순차적으로 가까운 관측값들끼리 묶어주는 병합 방법과 먼 관측값들을 나누어가는 분할 방법이 있으며, 주로 병합방법이 사용됨.
- 계층적 군집방법은 군집 결과를 나무구조의 덴드로그램을 통해 간단하게 나타낼 수 있고, 이를 이용하여 전체 군집들간의 구조적 관계를 쉽게 살펴볼 수 있다는 장점이 있음.



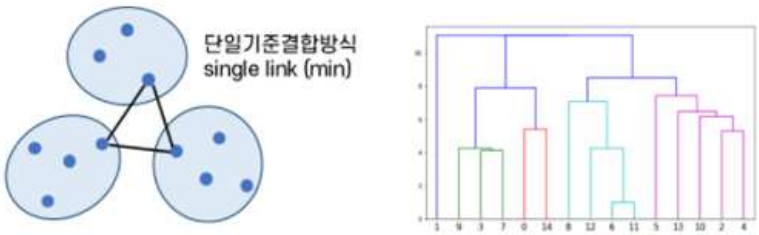
1) 병합방법

- 처음에는 n개의 자료 각각을 하나의 군집으로 취급하고, 이 n개의 군집 중 가장 거리가 가까운 두 군집을 병합하여 $n-1$개의 군집을 형성함.
- 주어진 군집들 중 거리가 가장 가까운 두 군지블 병합하고 군집의 개수를 하나씩 줄이는 과정을 반복하여 모든 자료가 하나의 군집에 속할 때까지 계속함.
- 이 과정은 시작부분에서는 군집의 크기는 작고 동질적이며 끝부분에서는 군집의 크기는 커지고 이질적으로 됨.

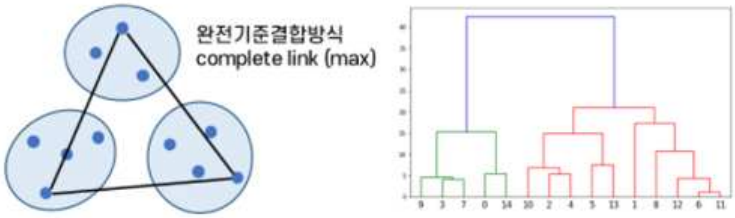


- 병합방법에는 군집들간의 거리를 측정하는 방법에 따라 다양한 종류가 있으며, 최단연결, 최장연결, 평균연결 방법 등이 있음.

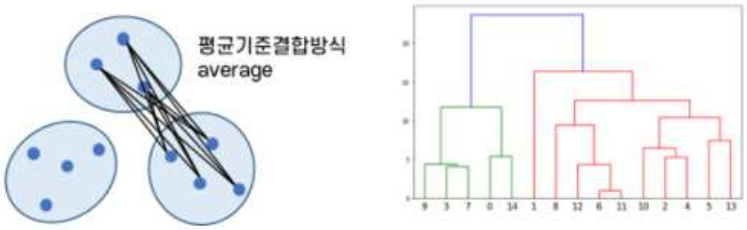
- 최단연결법은 두 군집 사이의 거리를 각 군집에서 하나씩 관측값을 뽑았을 때 나타날 수 있는 모든 조합의 거리의 최소값으로 측정하는 방법으로 같은 군집에 속하는 관측값은 다른 군집에 속하는 관측값에 비하여 거리가 가까운 변수를 적어도 하나는 갖고 있게 됨.
- 이는 고립된 군집을 찾는데 중점을 둔 방법으로서 군집이 고리형태로 연결되어 있는 경우에는 부적절한 결과를 줄 수도 있음.



- 최장연결법은 두 군집 사이의 거리를 각 군집에서 하나씩 관측값을 뽑았을 때 나타날 수 있는 거리의 최대값으로 측정함.
- 군집들의 내부 응집성에 중점을 둔 방법으로서 같은 군집에 속하는 관측치는 알려진 최대 거리보다 짧음.



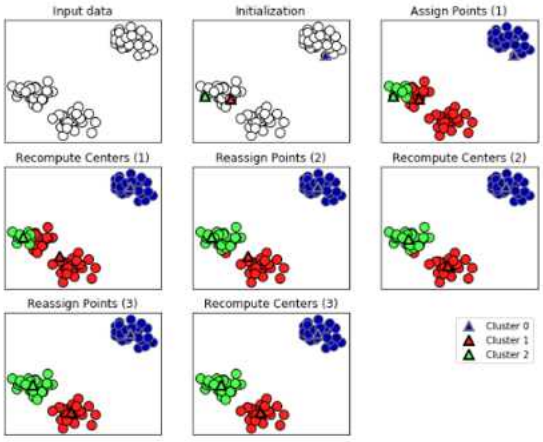
- 평균연결법은 한 군집 안에 속해 있는 모든 대상과 다른 군집에 속해있는 모든 대상이 쌍 집합에 대한 거리를 평균 계산 측정함.



○ k 평균 군집방법

- k 평균 군집방법은 주어진 군집수 k에 대해서 군집내 거리 제곱합의 합을 최소화 하는 것을 그 목적으로 함.
- k 평균 군집방법은 사전에 결정된 군집수 k가 주어지면 유클리드 거리를 이용하여 전체 자료를 사대적으로 유사한 k개의 군지으로 나눔.
- 계층적 군집방법에 비하여 계산량이 적고 대용량 데이터를 빠르게 처리할 수 있다는 장점이 있음.

- k 평균 군집방법을 실시하는 방법은 다음과 같음.
- 1) 특성변수의 자료타입에 따라 자료를 변환한 후 특성변수를 표준화 함.
- 2) 군집수 k가 주어지면 랜덤하게 초기 군집의 중심값을 $\mu_1,\cdots,\mu_k$로 놓음.
- 3) 유클리디안 거리를 기준으로 각 관측치를 가장 가까운 μ_i 군집에 편입함.
- 4) 각 군집의 평균을 구해 이들 새로운 군집의 중심값 μ_i로 놓은 후 3단계를 실행함.
- 5) 각 군집의 소속이 변하지 않을 때까지, 또는 미리 정해진 최대 반복횟수에 도달할 때까지 단계 4)를 반복함.



- 하지만 k 평균 군집방법은 단계 2)에서 뽑은 k개의 중심값이 나쁠 경우 군집의 결과가 나쁘게 나타날 수 있음.
- 이에 대한 해결책중 하나가 K-means++임.
- K-means++이 초기 중심점을 구하는 방법은 다음과 같음.

- 1) n개의 표본으로 구성된 학습데이터로부터 1개의임의표본을 뽑음. 이를 초기 중심값 μ_1으로 함.
- 2) 나머지 $n-1$개의 자료에 대해 μ_1으로부터의 유클리디안 거리를 구하고 이 거리에 비례한 확률을 $n-1$개의 자료 각각에 부여한 후, 이 확률에 비례하여 1개의 임의표본을 뽑음. 이를 두 번째 초기 중심값 μ_2로 함.
- 3) 이와 같은 방법으로 최종 μ_k를 구할 때까지 반복함.

2. scipy 및 sklearn을 활용한 군집분석

- 군집분석을 실시하기 위해 scipy 및 sklearn 모듈을 활용한 군집분석에 대해서 살펴보도록 하겠음.

- 군집분석은 거리를 측정하여 데이터간 인접성을 판단하므로, 데이터의 단위가 매우 중요함. 만약 단위의 크기가 매우 상이하다면 좋은 분석결과가 나타나지 않을 수 있음.
- 입력변수의 표준화를 위해 sklearn.preprocessing 모듈의 StandardScaler().fit() 함수를 이용할 수 있음.

```
from sklearn.preprocessing import StandardScaler
객체명 = StandardScaler().fit(데이터)
데이터 표준화: 객체명.transform(데이터)
```

- StandardScaler().fit() 함수에 분석에 사용되는 데이터를 입력함.
- 이를 통해 각 데이터의 입력변수별 표준화를 위한 평균과 표준편차를 구하게 됨. 이를 transform() 함수를 이용하여 자료를 변환함.

- 이렇게 표준화된 데이터를 이용하여 데이터간 거리행렬 및 계층적 군집분석을 실시함.

- 데이터간 거리를 측정한 거리행렬을 구하기 위해 scipy.spatial.distance 모듈의 pdist() 함수 및 squareform() 함수를 이용하여 실시할 수 있음.

```
from scipy.spartial.distance import pdist, squareform
데이터간 거리 측정: pdist(데이터, metric='euclidean'/'minkowski'/...)
데이터간 거리 2차원 배열: squareform(거리데이터)
```

- pdist() 함수에 분석에 사용되는 데이터를 먼저 입력함.
- 그리고 metric 옵션을 통해 거리를 측정하는 방법인 euclidean, minkowski 등을 지정할 수 있음. 이 때 euclidean 방법이 기본으로 사용되어짐.
- pdist() 함수를 통해 데이터간의 거리를 리스트 형태로 출력함. 하지만 이 자료를 통해서는 데이터간의 거리를 알아보기 힘들어 이를 행렬의 형태로 변경할 필요가 있음.
- 이 때 리스트 형태의 자료를 2차원 배열 자료의 형태로 나타내기 위해 squareform() 함수를 사용할 수 있음. pdist() 함수를 통해 계산된 데이터간 자료를 할당한 객체를 squareform() 함수에 대입함.
- squareform() 함수를 실행시키면 2차원 형태로 배열되어 있지만 데이터 번호가 나타나 있지 않기 때문에 알아보기 힘들어 지난 학습시간때 학습하였던 pandas 모듈의 DataFrame() 함수를 이용하여 데이터프레임 형태로 변환하여 군집분석을 실시하기 위한 데이터간 거리 행렬을 최종적으로 구할 수 있음.

- 그리고 데이터를 이용하여 계층적 군집분석을 실시하기 위해 scipy.cluster.hierarchy 모듈의 linkage() 함수를 이용하여 계층적 군집분석을 실시할 수 있음.

```
from scipy.cluster.hierarchy import linkage
linkage(데이터, method='complete'/'single'/'average',
metric='euclidean'/'minkowski'/...)
```

- likage() 함수에 계층적 군집분석에 사용되는 데이터를 먼저 입력함.
- 그리고 method 옵션을 통해 군집간 거리를 측정하는 방법인 최장연결법(complete), 최단연결법(single), 평균연결법(average) 등을 지정할 수 있음. 이 때 최단연결법(single) 방법이 기본으로 사용되어짐.
- 그리고 metric 옵션을 통해 거리를 측정하는 방법인 euclidean, minkowski 등을 지정할 수 있음. 이 때 euclidean 방법이 기본으로 사용되어짐.
- linkage() 함수를 통해 군집이 이루어지는 과정을 4개의 열을 가지는 2차원 배열의 자료가 표현됨.
- 0번째 열과 1번째 열에는 데이터의 행 인덱스가 출력되고 2번째 열에는 행 인덱스에 따른 데이터간의 거리가 출력됨. 마지막 열에는 군집이 이루어지는 순서를 나타냄.

- linkage() 함수를 통해 생성된 군집이 이루어지는 과정을 표로 확인하기에는 어려움이 많으므로 이를 덴드로그램을 통해 군집이 이루어지는 과정을 확인할 수 있음.
- 덴드로그램을 확인하기 위해 scipy.cluster.hierarchy 모듈의 dendrogram() 함수 및 matplotlib.pyplot 모듈을 이용하여 덴드로그램을 그릴 수 있음.

```
from scipy.cluster.hierarchy import dendrogram
dendrogram(군집분석 과정 데이터, leaf_rotation=행 인덱스 회전 출력,
leaf_font_size=행 인덱스 출력 크기)
```

- dendrogram() 함수에 linkage() 함수에 의해 생성된 계층적 군집분석 과정 데이터를 먼저 입력함.

- 그리고 leaf_rootation 옵션 및 leaf_font_size 옵션을 통해 덴드로그램에 표현되는 행 인덱스의 회전 출력 및 출력 크기를 수정할 수 있음.
- 하지만 dendrogram() 함수만으로는 그림을 효과적으로 보기 힘들기 때문에 가시성이 향상된 그림을 위해서는 matplotlib.pyplot 모듈의 함수들을 적절하게 이용하는 것이 좋음.

- 덴드로그램을 통해 분석자 임의로 몇 개의 군집으로 분류하면 적절할지를 결정하게 될 것임. 이 때 군집을 분류하여 군집 번호를 할당하기 위해 scipy.cluster.hierarchy 모듈의 fcluster() 함수를 사용할 수 있음.

```
from scipy.cluster.hierarchy import fcluster
fcluster(군집분석 과정 데이터, t=군집 구분 거리, criterion='distance')
```

- fcluster() 함수에 linkage() 함수에 의해 생성된 계층적 군집분석 과정 데이터를 먼저 입력함.
- 그리고 t 옵션을 통해 군집을 구분하기 위한 기준 거리를 입력함.

- 다음으로 데이터를 이용하여 비계층적 군집분석인 k 평균 군집분석을 실시하기 위해 sklearn.cluster 모듈의 KMeans() 함수를 이용하여 k 평균 군집분석을 실시할 수 있음.

```
from sklearn.cluster import KMeans
KMeans(n_cluster=군집수, init='k-means++'/'random').fit(데이터)
```

- KMeans().fit() 함수에 k 평균 군집분석에 사용되는 데이터를 입력함.
- 그리고 n_cluster 옵션을 통해 구하고자하는 군집의 수인 k 값을 입력함.
- 그리고 init 옵션을 통해 일반적인 k 평균 군집분석 방법을 수행하는 'random'과 군집의 초기 군집값 문제를 보완한 k-means++를 활용한 k 평균 군집분석 방법을 수행하는 'k-means++'를 입력할 수 있음.

- 또한, k 평균 군집분석의 결과를 저장한 객체에 다음과 같은 함수를 활용하여 군집의 중심값 및 군집 분류값을 확인할 수 있음.

군집 중심값: KMeans 객체.**cluster_centers_**
군집 분류값: KMeans 객체.**predict**(데이터)

- seaborn 모듈의 iris 데이터에서, species를 예측하기 위해 sepal_length, sepal_width, petal_length, petal_width를 사용한 계층적 군집분석을 실시하고자 함.
- info() 함수를 활용하여 자료를 확인한 결과 분석에 사용되는 모든 변수에서 결측자료가 없는 것을 확인할 수 있음.

```
In [1]: import seaborn as sns

In [2]: df = sns.load_dataset('iris')

In [3]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

- 다음으로 데이터의 표준화를 통하여 거리를 측정한 후 계층적 군집분석을 실시하기 위해 먼저 데이터를 표준화 하기 위해 StandardScaler() 함수를 사용하였음.

```
In [4]: X=df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]

In [5]: from sklearn.preprocessing import StandardScaler

In [6]: sc=StandardScaler().fit(X)

In [7]: X_std=sc.transform(X)

In [8]: print(X_std[:5])
[[-0.90068117  1.01900435 -1.34022653 -1.3154443 ]
 [-1.14301691 -0.13197948 -1.34022653 -1.3154443 ]
 [-1.38535265  0.32841405 -1.39706395 -1.3154443 ]
 [-1.50652052  0.09821729 -1.2833891  -1.3154443 ]
 [-1.02184904  1.24920112 -1.34022653 -1.3154443 ]]
```

- 주어진 데이터 X 객체를 StandardScaler().fit()에 대입하여 표준화를 위한 평균과 표준편차를 계산한 결과를 sc 객체에 할당하였음.
- 이 sc 객체에 transform() 함수를 적용하여 각각 X 객체를 표준화한 X_std 객체를 생성하였음.
- 다음으로 표준화된 주어진 자료를 이용하여 데이터간 거리행렬을 구하기 위해 pdist() 함수 및 squareform() 함수를 사용하였음.

```
In [9]: import pandas as pd

In [10]: from scipy.spatial.distance import pdist, squareform

In [11]: row_dist = pd.DataFrame(squareform(pdist(X_std, metric='euclidean')))

In [12]: print(row_dist)
          0         1         2    ...       147       148       149
0    0.000000  1.176219  0.845607  ...  3.805412  3.826157  3.335064
1    1.176219  0.000000  0.523373  ...  3.746638  4.017345  3.213476
2    0.845607  0.523373  0.000000  ...  3.936541  4.072716  3.380336
3    1.103685  0.434000  0.283891  ...  3.922954  4.097587  3.340154
4    0.260139  1.386485  0.991572  ...  3.935711  3.890518  3.457620
..        ...       ...       ...  ...       ...       ...       ...
145  4.170050  4.130727  4.317340  ...  0.463325  1.108067  1.173044
146  4.075223  3.660136  3.973046  ...  1.189008  2.153099  1.257071
147  3.805412  3.746638  3.936541  ...  0.000000  1.071844  0.775292
148  3.826157  4.017345  4.072716  ...  1.071844  0.000000  1.200930
149  3.335064  3.213476  3.380336  ...  0.775292  1.200930  0.000000

[150 rows x 150 columns]
```

- 이 때 pandas 모듈의 DataFrame() 함수를 적용하여 데이터 행 인덱스에 따른 데이터간 거리의 행렬을 확인할 수 있음.

- 표준화된 데이터인 X_std를 linkage() 함수에 대입하여 군집이 이루어지는 과정을 생성하였음. 이 때, method 옵션에 complete를 입력하여 군집간 거리측정 방법은 최장연결법을 사용하였고, metric 옵션에 euclidean을 입력하여 유클리디안 거리를 사용하였음.

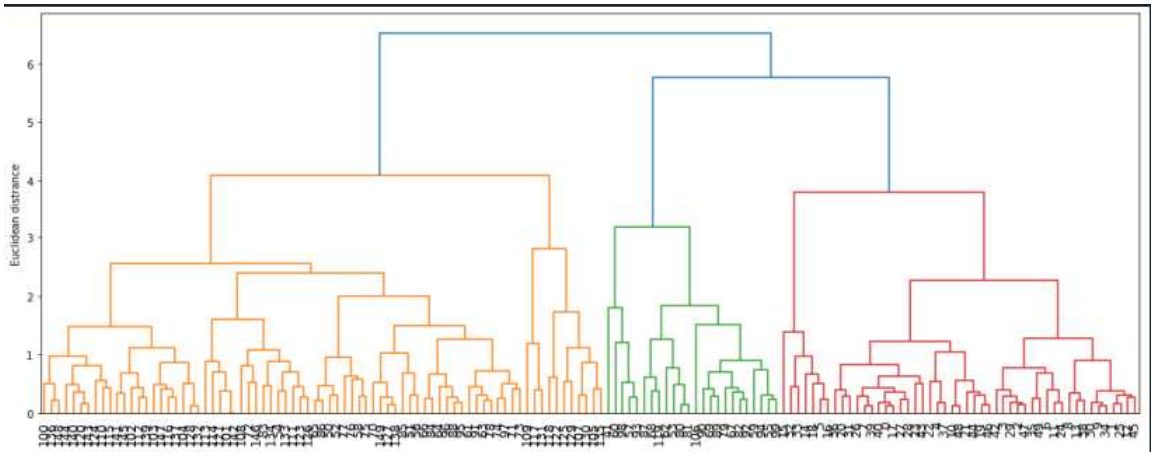
```
In [13]: from scipy.cluster.hierarchy import linkage

In [14]: clusters = linkage(y=X_std, method='complete', metric='euclidean')

In [15]: clusters[:5]
Out[15]:
array([[1.01000000e+02, 1.42000000e+02, 0.00000000e+00, 2.00000000e+00],
       [7.00000000e+00, 3.90000000e+01, 1.21167870e-01, 2.00000000e+00],
       [1.00000000e+01, 4.80000000e+01, 1.21167870e-01, 2.00000000e+00],
       [9.00000000e+00, 3.40000000e+01, 1.31632184e-01, 2.00000000e+00],
       [0.00000000e+00, 1.70000000e+01, 1.31632184e-01, 2.00000000e+00]])
```

- 그 결과 101번 데이터와 142번 데이터가 가장 가까운 데이터로 가장 먼저 군집으로 묶인 것을 확인할 수 있음. 그 다음으로 7번 데이터와 39번 데이터가 군집으로 묶이는 것을 확인할 수 있음.
- 이러한 결과를 그림으로 확인하기 위해 dendrogram() 함수를 이용하여 덴드로그램을 생성하였음.

```
In [17]: from scipy.cluster.hierarchy import dendrogram
    ...: import matplotlib.pyplot as plt
    ...: plt.figure(figsize = (15, 6))
    ...: dendrogram(clusters, leaf_rotation=90, leaf_font_size=12)
    ...: plt.tight_layout()
    ...: plt.ylabel('Euclidean distrance')
    ...: plt.show()
```



- 덴드로그램으로 확인한 결과 3개의 큰 군집으로 확인이 가능해짐. 물론 이는 분석자의 생각에 따라 달라질 수 있음.
- 3개의 군집은 거리의 기준이 5정도의 값에서 이루어진다는 것을 확인할 수 있음. 따라서 데이터를 이러한 기준으로 나누어 군집을 할당하기 위해 fcluster() 함수를 사용하였음.

```
In [17]: from scipy.cluster.hierarchy import fcluster

In [18]: cut_tree = fcluster(clusters, t=5, criterion='distance')

In [19]: print(cut_tree)
[3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 2 3 3 3 3 3 3 3 3 1 1 1 2 1 2 1 2 1 2 2 1 2 1 1 1 1 2 2 2 1 1 1 1
 1 1 1 1 1 2 2 2 2 1 1 1 1 2 1 2 2 1 2 2 2 1 1 1 2 2 1 1 1 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 1 1 1 1 1 1
 1 1]
```

- fcluster()를 통해 cut_tree 객체에 분류된 군집을 할당하였음.

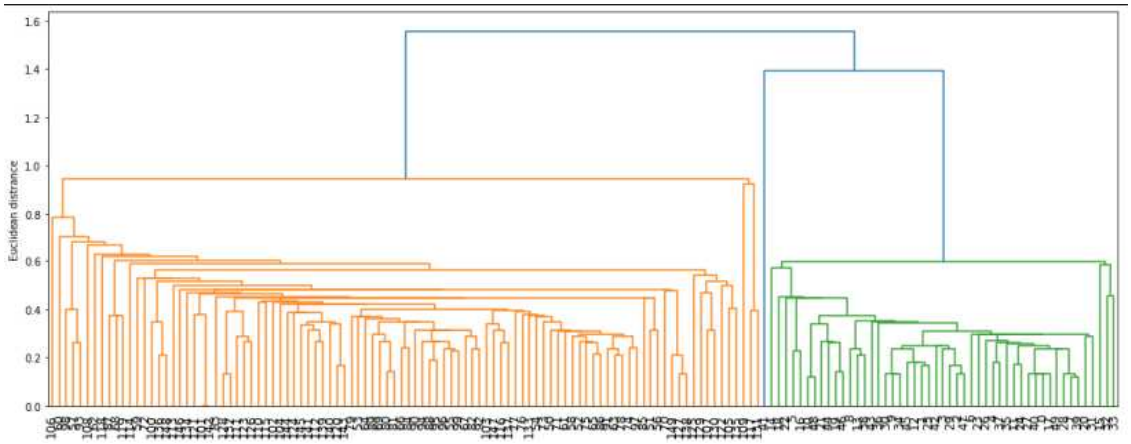
- 다음으로 method 옵션에 single을 입력하여 최단연결법으로 군집간 거리를 측정하였고, metric 옵션에 euclidean을 입력하여 유클리디안 거리를 사용하였음.

```
In [20]: from scipy.cluster.hierarchy import linkage

In [21]: clusters1 = linkage(y=X_std, method='single', metric='euclidean')

In [22]: clusters1[:5]
Out[22]:
array([[1.01000000e+02, 1.42000000e+02, 0.00000000e+00, 2.00000000e+00],
       [7.00000000e+00, 3.90000000e+01, 1.21167870e-01, 2.00000000e+00],
       [1.00000000e+01, 4.80000000e+01, 1.21167870e-01, 2.00000000e+00],
       [9.00000000e+00, 3.40000000e+01, 1.31632184e-01, 2.00000000e+00],
       [0.00000000e+00, 1.70000000e+01, 1.31632184e-01, 2.00000000e+00]])
```

- 그리고 이러한 결과를 그림으로 확인하기 위해 dendrogram() 함수를 이용하여 덴드로그램을 생성하였음.



- 덴드로그램으로 확인한 결과 3개의 큰 군집으로 확인이 가능해짐. 물론 이는 분석자의 생각에 따라 달라질 수 있음.
- 3개의 군집은 거리의 기준이 1.2정도의 값에서 이루어진다는 것을 확인할 수 있음. 따라서 데이터를 이러한 기준으로 나누어 군집을 할당하기 위해 fcluster() 함수를 사용하였음.

```
In [24]: from scipy.cluster.hierarchy import fcluster

In [25]: cut_tree1 = fcluster(clusters1, t=1.2, criterion='distance')

In [26]: print(cut_tree1)
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 3 2 2 2 2 2 2 2 2 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 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1]
```

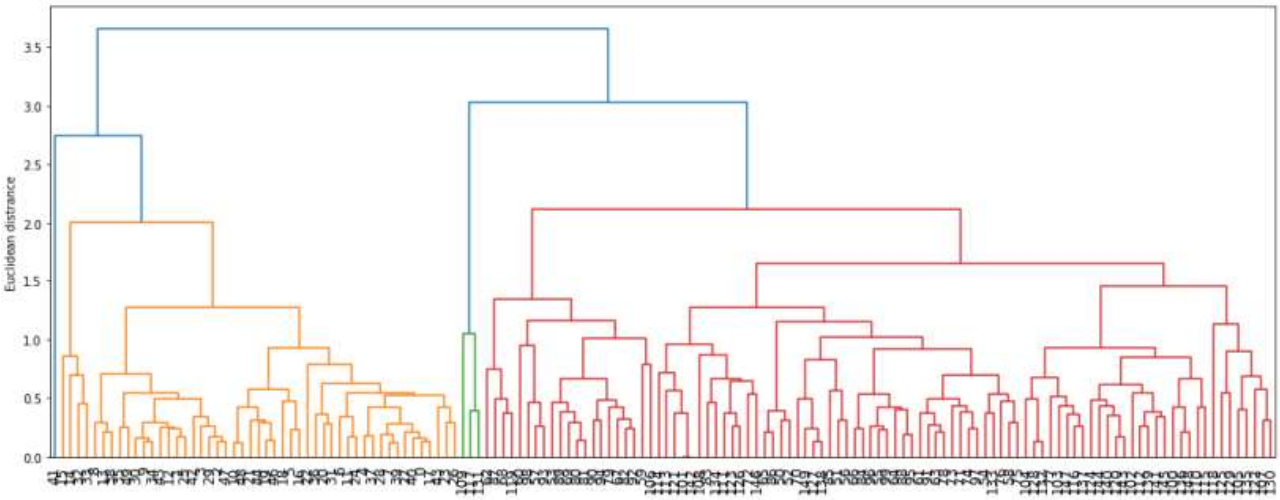
- fcluster()를 통해 cut_tree1 객체에 분류된 군집을 할당하였음.
- 덴드로그램과 fcluster() 결과를 살펴보면 41번째 자료에서 혼자 3번째 군집으로 군집이 할당된 것을 확인할 수 있음. 이를 통해 41번째 자료는 다른 자료에 비해 이상점으로 판단할 수 있음.
- 마지막으로 method 옵션에 average를 입력하여 평균연결법으로 군집간 거리를 측정하였고, metric 옵션에 euclidean을 입력하여 유클리디안 거리를 사용하였음.

```
In [27]: from scipy.cluster.hierarchy import linkage

In [28]: clusters2 = linkage(y=X_std, method='average', metric='euclidean')

In [29]: clusters2[:5]
Out[29]:
array([[1.01000000e+02, 1.42000000e+02, 0.00000000e+00, 2.00000000e+00],
       [7.00000000e+00, 3.90000000e+01, 1.21167870e-01, 2.00000000e+00],
       [1.00000000e+01, 4.80000000e+01, 1.21167870e-01, 2.00000000e+00],
       [9.00000000e+00, 3.40000000e+01, 1.31632184e-01, 2.00000000e+00],
       [0.00000000e+00, 1.70000000e+01, 1.31632184e-01, 2.00000000e+00]])
```

- 그리고 이러한 결과를 그림으로 확인하기 위해 dendrogram() 함수를 이용하여 덴드로그램을 생성하였음.



- 덴드로그램으로 확인한 결과 3개의 큰 군집으로 확인이 가능해짐. 물론 이는 분석자의 생각에 따라 달라질 수 있음.
- 3개의 군집은 거리의 기준이 2.8정도의 값에서 이루어진다는 것을 확인할 수 있음. 따라서 데이터를 이러한 기준으로 나누어 군집을 할당하기 위해 fcluster() 함수를 사용하였음.

```
In [31]: from scipy.cluster.hierarchy import fcluster

In [32]: cut_tree2 = fcluster(clusters2, t=2.8, criterion='distance')

In [33]: print(cut_tree2)
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 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 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 3 3
 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3]
```

- fcluster()를 통해 cut_tree2 객체에 분류된 군집을 할당하였음.

- 다음으로 표준화된 데이터인 X_std를 KMeans() 함수에 대입하여 k 평균 군집분석을 실시하고자 함. 이 때, n_clusters 옵션에 3을 입력하여 군집의 수가 3개가 생성되도록 하였음. 그리고 init 옵션에 'random'을 입력하여 일반적인 k 평균 군집분석을 실시하였음.

```
In [41]: from sklearn.cluster import KMeans

In [42]: model = KMeans(n_clusters=3, random_state=1, init='random').fit(X_std)
```

- k 평균 군집분석을 실시한 결과를 model 객체에 할당하여 predict()를 활용하여 군집을 확인할 수 있음. 또한 cluster_centers_ 함수를 활용하여 군집의 중심위치를 확인할 수 있음.

```
In [43]: centers = model.cluster_centers_

In [44]: print(centers)
[[-0.05021989 -0.88337647  0.34773781  0.2815273 ]
 [-1.01457897  0.85326268 -1.30498732 -1.25489349]
 [ 1.13597027  0.08842168  0.99615451  1.01752612]]

In [45]: pred = model.predict(X_std)

In [46]: print(pred)
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 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 0 0 0 2 0 0 0 0 0 0 0 0 2 0 0 0 0 2 0 0 0
 0 2 2 2 0 0 0 0 0 0 0 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 2 2 2 2 2 2 0 0 2 2 2 0 2 2 2 0 2 2 2 0 2
 2 0]
```

- 그리고 표준화된 데이터인 X_std를 KMeans() 함수에 대입하여 초기 군집 중심점 문제를 보완한 k-means++를 활용한 k 평균 군집분석을 실시하고자 함. 이 때, KMeans() 함수의 init 옵션에 'k-means++'을 입력하여 k-means++를 활용한 k 평균 군집분석을 실시하였음.

```
In [47]: from sklearn.cluster import KMeans

In [48]: model = KMeans(n_clusters=3, random_state=1, init='k-means++').fit(X_std)
```

- k 평균 군집분석을 실시한 결과를 model 객체에 할당하여 predict()를 활용하여 군집을 확인할 수 있음. 또한 cluster_centers_ 함수를 활용하여 군집의 중심위치를 확인할 수 있음.

```
In [49]: centers = model.cluster_centers_

In [50]: print(centers)
[[-0.05021989 -0.88337647  0.34773781  0.2815273 ]
 [-1.01457897  0.85326268 -1.30498732 -1.25489349]
 [ 1.13597027  0.08842168  0.99615451  1.01752612]]

In [51]: pred = model.predict(X_std)

In [52]: print(pred)
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 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 0 0 0 2 0 0 0 0 0 0 0 0 2 0 0 0 0 2 0 0 0
 0 2 2 2 0 0 0 0 0 0 0 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 2 2 2 2 2 2 0 0 2 2 2 0 2 2 2 0 2 2 2 0 2
 2 0]
```

- iris 자료의 경우 군집의 중심점이 k-means++와 일반적인 군집분석을 사용했을 때의 결과가 같게 나타나는 것을 확인할 수 있음.

3. 군집분석의 실습

○ seaborn 모듈의 penguins 데이터에서, species를 예측하기 위해 bill_length_mm, bill_depth_mm, flipper_length_mm, body_mass_g을 사용한 의사결정나무분석을 실시하세요.

- info() 함수를 활용하여 자료를 확인한 결과 bill_length_mm, bill_depth_mm, flipper_length_mm, body_mass_g에서 결측자료가 존재하는 것을 확인할 수 있음. 따라서 dropna() 함수를 이용하여 결측값을 제거할 필요가 있음.

```
In [1]: import seaborn as sns

In [2]: df = sns.load_dataset('penguins')

In [3]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   species            344 non-null    object
 1   island             344 non-null    object
 2   bill_length_mm     342 non-null    float64
 3   bill_depth_mm      342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
 6   sex                333 non-null    object
dtypes: float64(4), object(3)
memory usage: 18.9+ KB

In [4]: df = df.dropna(subset=['species', 'bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g'],
   ...:                how='any', axis=0)
```

- 다음으로 데이터의 표준화를 통하여 거리를 측정한 후 계층적 군집분석을 실시하기 위해 먼저 데이터를 표준화 하기 위해 StandardScaler() 함수를 사용하였음.

```
In [5]: X=df[['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g']]

In [6]: from sklearn.preprocessing import StandardScaler

In [7]: sc=StandardScaler().fit(X)

In [8]: X_std=sc.transform(X)

In [9]: print(X_std)
[[-0.88449874  0.78544923 -1.41834665 -0.56414208]
 [-0.81112573  0.1261879  -1.06225022 -0.50170305]
 [-0.66437972  0.43046236 -0.42127665 -1.18853234]
 ...
 [ 1.18828874 -0.73592307  1.50164406  1.93341896]
 [ 0.23443963 -1.19233476  0.7894512   1.24658968]
 [ 1.09657248 -0.53307343  0.86067049  1.49634578]]
```

- 주어진 데이터 X 객체를 StandardScaler().fit()에 대입하여 표준화를 위한 평균과 표준편차를 계산한 결과를 sc 객체에 할당하였음.
- 이 sc 객체에 transform() 함수를 적용하여 각각 X 객체를 표준화한 X_std 객체를 생성하였음.

- 다음으로 표준화된 주어진 자료를 이용하여 데이터간 거리행렬을 구하기 위해 pdist() 함수 및 squareform() 함수를 사용하였음.

```
In [10]: import pandas as pd

In [11]: from scipy.spatial.distance import pdist, squareform

In [12]: row_dist = pd.DataFrame(squareform(pdist(X_std, metric='euclidean')))

In [13]: print(row_dist)
          0         1         2    ...       339       340       341
0    0.000000  0.755455  1.248391  ...  4.623330  3.649215  3.886216
1    0.755455  0.000000  0.998348  ...  4.152622  3.052382  3.429834
2    1.248391  0.998348  0.000000  ...  4.270485  3.291964  3.589055
3    1.075704  1.277407  0.976404  ...  4.707812  3.774348  4.056464
4    1.164555  1.658632  1.465765  ...  4.731974  4.003018  4.055544
..        ...       ...       ...  ...       ...       ...       ...
337  4.039011  3.386142  3.544491  ...  1.661106  0.764195  1.443448
338  3.837219  3.201489  3.311103  ...  1.565831  0.622185  1.283987
339  4.623330  4.152622  4.270485  ...  0.000000  1.448134  0.807118
340  3.649215  3.052382  3.291964  ...  1.448134  0.000000  1.115952
341  3.886216  3.429834  3.589055  ...  0.807118  1.115952  0.000000

[342 rows x 342 columns]
```

- 이 때 pandas 모듈의 DataFrame() 함수를 적용하여 데이터 행 인덱스에 따른 데이터간 거리의 행렬을 확인할 수 있음.

- 표준화된 데이터인 X_std를 linkage() 함수에 대입하여 군집이 이루어지는 과정을 생성하였음. 이 때, method 옵션에 complete를 입력하여 군집간 거리측정 방법은 최장연결법을 사용하였고, metric 옵션에 euclidean을 입력하여 유클리디안 거리를 사용하였음.

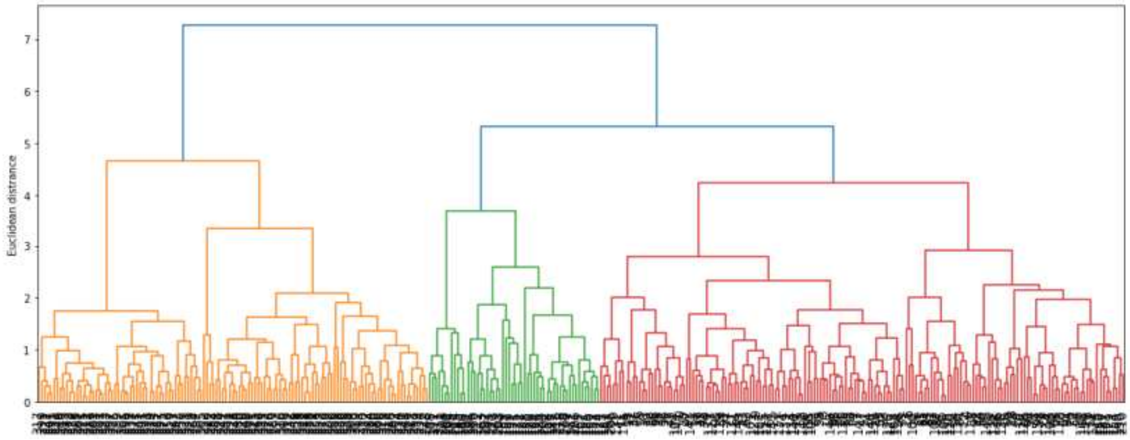
```
In [14]: from scipy.cluster.hierarchy import linkage

In [15]: clusters = linkage(y=X_std, method='complete', metric='euclidean')

In [16]: clusters[:5]
Out[16]:
array([[2.25000000e+02, 3.11000000e+02, 1.08991018e-01, 2.00000000e+00],
       [2.60000000e+02, 3.12000000e+02, 1.08991018e-01, 2.00000000e+00],
       [3.50000000e+01, 1.06000000e+02, 1.26537458e-01, 2.00000000e+00],
       [2.66000000e+02, 3.02000000e+02, 1.31843434e-01, 2.00000000e+00],
       [7.50000000e+01, 1.42000000e+02, 1.32965177e-01, 2.00000000e+00]])
```

- 그 결과 225번 데이터와 311번 데이터가 가장 가까운 데이터로 가장 먼저 군집으로 묶인 것을 확인할 수 있음. 그 다음으로 260번 데이터와 312번 데이터가 군집으로 묶이는 것을 확인할 수 있음.
- 이러한 결과를 그림으로 확인하기 위해 dendrogram() 함수를 이용하여 덴드로그램을 생성하였음.

```
In [17]: from scipy.cluster.hierarchy import dendrogram
    ...: import matplotlib.pyplot as plt
    ...: plt.figure(figsize = (15, 6))
    ...: dendrogram(clusters, leaf_rotation=90, leaf_font_size=12)
    ...: plt.tight_layout()
    ...: plt.ylabel('Euclidean distrance')
    ...: plt.show()
```



- 덴드로그램으로 확인한 결과 3개의 큰 군집으로 확인이 가능해짐. 물론 이는 분석자의 생각에 따라 달라질 수 있음.
- 3개의 군집은 거리의 기준이 5정도의 값에서 이루어진다는 것을 확인할 수 있음. 따라서 데이터를 이러한 기준으로 나누어 군집을 할당하기 위해 fcluster() 함수를 사용하였음.

```
In [18]: from scipy.cluster.hierarchy import fcluster

In [19]: cut_tree = fcluster(clusters, t=5, criterion='distance')

In [20]: print(cut_tree)
[3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 2 2 2 3 2 3 2 2 3 2 3 2 2 2 2 2 2 2 3 2 3 2 3 2 2 2 2 2 3 2 3 2 3 2
 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 3 2 2 2 2 3 2 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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1]
```

- fcluster()를 통해 cut_tree 객체에 분류된 군집을 할당하였음.

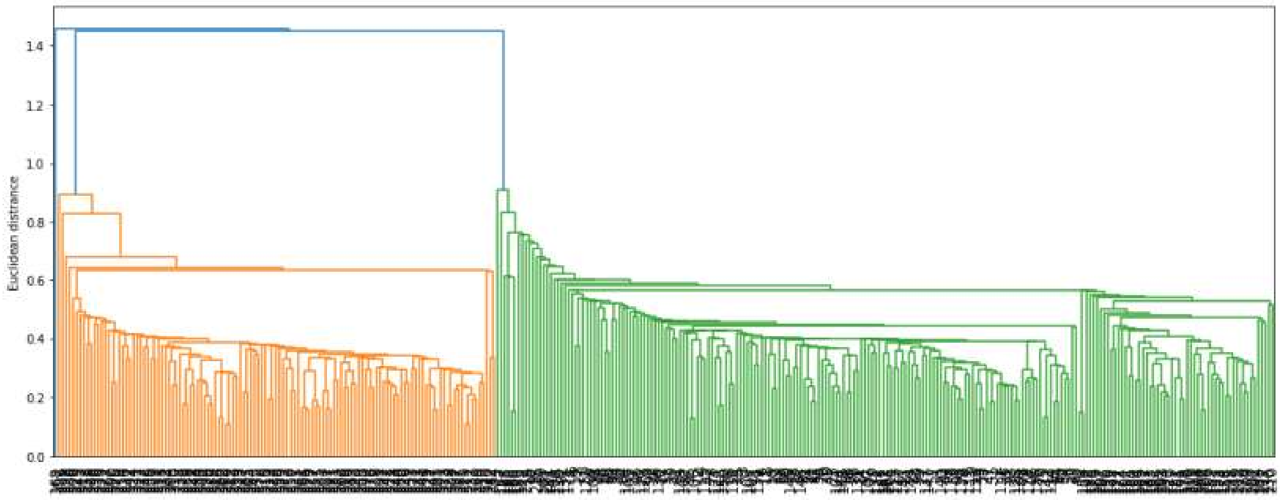
- 다음으로 method 옵션에 single을 입력하여 최단연결법으로 군집간 거리를 측정하였고, metric 옵션에 euclidean을 입력하여 유클리디안 거리를 사용하였음.

```
In [21]: from scipy.cluster.hierarchy import linkage

In [22]: clusters1 = linkage(y=X_std, method='single', metric='euclidean')

In [23]: clusters1[:5]
Out[23]:
array([[2.25000000e+02, 3.11000000e+02, 1.08991018e-01, 2.00000000e+00],
       [2.60000000e+02, 3.12000000e+02, 1.08991018e-01, 2.00000000e+00],
       [3.50000000e+01, 1.06000000e+02, 1.26537458e-01, 2.00000000e+00],
       [2.66000000e+02, 3.02000000e+02, 1.31843434e-01, 2.00000000e+00],
       [7.50000000e+01, 1.42000000e+02, 1.32965177e-01, 2.00000000e+00]])
```

- 그리고 이러한 결과를 그림으로 확인하기 위해 dendrogram() 함수를 이용하여 덴드로그램을 생성하였음.



- 덴드로그램으로 확인한 결과 3개의 큰 군집으로 확인이 가능해짐. 물론 이는 분석자의 생각에 따라 달라질 수 있음.
- 3개의 군집은 거리의 기준이 1.2정도의 값에서 이루어진다는 것을 확인 할 수 있음. 따라서 데이터를 이러한 기준으로 나누어 군집을 할당하기 위해 fcluster() 함수를 사용하였음.

```
In [25]: from scipy.cluster.hierarchy import fcluster

In [26]: cut_tree1 = fcluster(clusters1, t=1.2, criterion='distance')

In [27]: print(cut_tree1)
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 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 2 2 2 2 2 2 2 2 2 2 2 2 2 2 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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1]
```

- fcluster()를 통해 cut_tree1 객체에 분류된 군집을 할당하였음.

- 덴드로그램과 fcluster() 결과를 살펴보면 168번째 자료에서 혼자 3번째 군집으로 군집이 할당된 것을 확인할 수 있음. 이를 통해 168번째 자료는 다른자료에 비해 이상점으로 판단할 수 있음.

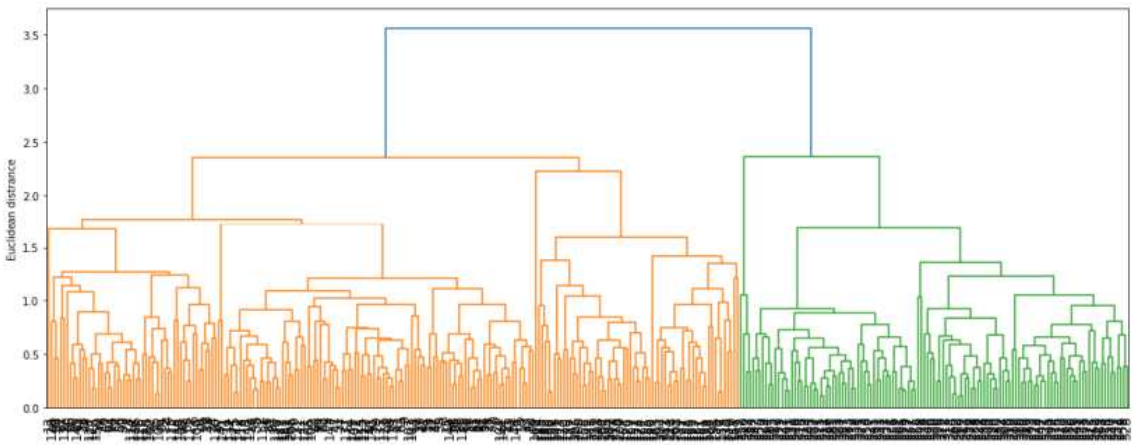
- 마지막으로 method 옵션에 average를 입력하여 평균연결법으로 군집간 거리를 측정하였고, metric 옵션에 euclidean을 입력하여 유클리디안 거리를 사용하였음.

```
In [28]: from scipy.cluster.hierarchy import linkage

In [29]: clusters2 = linkage(y=X_std, method='average', metric='euclidean')

In [30]: clusters2[:5]
Out[30]:
array([[2.25000000e+02, 3.11000000e+02, 1.08991018e-01, 2.00000000e+00],
       [2.60000000e+02, 3.12000000e+02, 1.08991018e-01, 2.00000000e+00],
       [3.50000000e+01, 1.06000000e+02, 1.26537458e-01, 2.00000000e+00],
       [2.66000000e+02, 3.02000000e+02, 1.31843434e-01, 2.00000000e+00],
       [7.50000000e+01, 1.42000000e+02, 1.32965177e-01, 2.00000000e+00]])
```

- 그리고 이러한 결과를 그림으로 확인하기 위해 dendrogram() 함수를 이용하여 덴드로그램을 생성하였음.



- 덴드로그램으로 확인한 결과 2개의 큰 군집으로 확인이 가능해짐. 물론 이는 분석자의 생각에 따라 달라질 수 있음.
- 2개의 군집은 거리의 기준이 2.5정도의 값에서 이루어진다는 것을 확인할 수 있음. 따라서 데이터를 이러한 기준으로 나누어 군집을 할당하기 위해 fcluster() 함수를 사용하였음.

```
In [32]: from scipy.cluster.hierarchy import fcluster

In [33]: cut_tree2 = fcluster(clusters2, t=2.5, criterion='distance')

In [34]: print(cut_tree2)
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 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 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2]
```

- fcluster()를 통해 cut_tree2 객체에 분류된 군집을 할당하였음.

- 다음으로 표준화된 데이터인 X_std를 KMeans() 함수에 대입하여 k 평균 군집분석을 실시하고자 함. 이 때, n_clusters 옵션에 3을 입력하여 군집의 수가 3개가 생성되도록 하였음. 그리고 init 옵션에 'random'을 입력하여 일반적인 k 평균 군집분석을 실시하였음.

```
In [43]: from sklearn.cluster import KMeans

In [44]: model = KMeans(n_clusters=3, random_state=1, init='random').fit(X_std)
```

- k 평균 군집분석을 실시한 결과를 model 객체에 할당하여 predict()를 활용하여 군집을 확인할 수 있음. 또한 cluster_centers_ 함수를 활용하여 군집의 중심위치를 확인할 수 있음.

```
In [45]: centers = model.cluster_centers_

In [46]: print(centers)
[[-1.04805941  0.48655336 -0.89121603 -0.77061658]
 [ 0.65722922 -1.09998045  1.15886506  1.0917612 ]
 [ 0.66097295  0.8169259  -0.28620559 -0.37431308]]

In [47]: pred = model.predict(X_std)

In [48]: print(pred)
[0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 2 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 2 0 0 0 0 0 2 0 0 0 2 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 2 0 0 0 2 0
 2 0 0 0 2 0 2 0 0 0 0 0 0 0 0 0 2 0 0 0 2 0 0 0 2 0 2 0 0 0 0 0 0 0 2 0 2
 0 2 0 2 0 0 0 0 0 0 0 2 0 0 0 0 0 2 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 0 2 2 2 2 2 2 2 0 2 0 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1]
```

- 그리고 표준화된 데이터인 X_std를 KMeans() 함수에 대입하여 초기 군집 중심점 문제를 보완한 k-means++를 활용한 k 평균 군집분석을 실

시하고자 함. 이 때, KMeans() 함수의 init 옵션에 'k-means++'을 입력하여 k-means++를 활용한 k 평균 군집분석을 실시하였음.

```
In [49]: from sklearn.cluster import KMeans

In [50]: model = KMeans(n_clusters=3, random_state=1, init='k-means++').fit(X_std)
```

- k 평균 군집분석을 실시한 결과를 model 객체에 할당하여 predict()를 활용하여 군집을 확인할 수 있음. 또한 cluster_centers_ 함수를 활용하여 군집의 중심위치를 확인할 수 있음.

```
In [51]: centers = model.cluster_centers_

In [52]: print(centers)
[[ 0.66097295  0.8169259  -0.28620559 -0.37431308]
 [ 0.65722922 -1.09998045  1.15886506  1.0917612 ]
 [-1.04805941  0.48655336 -0.89121603 -0.77061658]]

In [53]: pred = model.predict(X_std)

In [54]: print(pred)
[2 2 2 2 2 2 2 2 0 2 2 2 2 2 2 2 0 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 0 2 2 2 2 2 0 2 2 2 0 2 2 2 2 2 2 2 0 2 2 2 0 2 2 2 2 2 2
 0 2 2 2 0 2 0 2 2 2 2 2 2 2 2 2 2 0 2 2 2 0 2 2 2 0 2 0 2 2 2 2 2 2 0
 2 0 2 0 2 2 2 2 2 2 2 0 2 2 2 2 2 2 0 2 0 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 2 0 2 0 0 0 0 0 0 0 2 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1]
```

- penguins 자료의 경우 군집의 중심점이 k-means++와 일반적인 군집분석을 사용했을 때의 결과가 같게 나타나는 것을 확인할 수 있음.