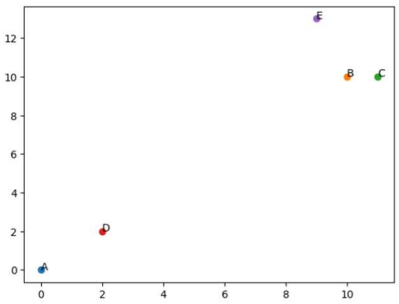
군집분석(Cluster Analysis)

군집분석이란 데이터들의 거리를 기반으로 가까운 점들을 모아서 그룹화하는 방법이라고 할 수 있다. 주성분분석과 같은 대표적인 비지도학습(unsupervised learning)[1]으로 분류되는 군집분석은 접근하는 방법론에 따라 계층적 군집분석, K-Means 군집분석, DBSCAN[2] 분석, GMM[3] 분석 등 다양한 방법이 사용되고 있다. 여기서는 계층적 군집분석과 K-Means 군집분석에 대해서만 간단히 리뷰하도록 한다.

문제1. A(0,0), B(10,10), C(11,10), D(2,2), E(9,13)의 점을 바탕으로 비슷한 점끼리 묶어 보아라.

먼저 좌표평면에 찍어보도록 하자. 크게 두 그룹으로 나눈다면 (A,D), (B,C,E)가 한 그룹이 될 것이다. 하지만, 컴퓨터로 하여금 이 작업을 수행하게 한다면 어떻게 하면 될까? 이 질문에 대한 답이 바로 군집분석의 핵심작업이 된다.

```python
import matplotlib.pyplot as plt

# 점의 좌표를 사전형으로 정의
points = {'A': (0, 0), 'B': (10, 10), 'C': (11, 10), 'D': (2, 2), 'E': (9, 13)}

# *coords: coords의 모든 값을 의미함
# points.items()라고 하면 사전의 이름과 내용을 가져옴
for name, coords in points.items():
    plt.scatter(*coords, label=name)  # 좌표에 점을 표시
    plt.text(coords[0], coords[1], name)  # 좌표에 이름을 표시

# 그래프 출력
plt.show()
```



유클리디안 거리(Euclidean distance)는 우리가 중,고등학교 시절 배웠던 두 점간의 거리 공식을 떠올리면 되는데, 두 점간의 거리를 구하는 방법은 유클리디안 거리 말고도 여러가지 방법이 존재[4]한다. 아래는 'scipy'의 'distance_matrix'를 이용하여, 유클리디안 거리에 의해 각 좌표의 거리 행렬을 만들어

---

[1] 주어진 데이터에 대한 레이블 또는 목표값이 주어지지 않는 상황에서 데이터의 구조, 패턴, 그룹을 발견하고 모델을 학습하는 방법론이다.

[2] Density-based spatial clustering of application with noise의 약자로 고차원의 데이터베이스들을 처리하며, 노이즈를 잘 걸러낸다고 알려져 있다.

[3] Gaussian Mixture Model은 여러 개의 혼합된 가우시안 분포에서 생성되었다고 가정하는 확률모델에 기반한 방법으로 K-Means 평균 알고리즘과 비슷하게 동작한다고 알려져 있다.

[4] '다차원데이터의 분석' 편 참조

본 것이다.

```
from scipy.spatial import distance_matrix
import pandas as pd

# 좌표를 리스트로 변환
coords_list = list(points.values())

# p는 Minkowski p-norm
# p=2: 유클리디안 거리 행렬 계산 (디폴트값)
dist_matrix=distance_matrix(coords_list, coords_list,p=2)

# 결과를 DataFrame으로 변환하여 출력
df_dist_matrix = pd.DataFrame(dist_matrix,
                              index=points.keys(),
                              columns=points.keys())
df_dist_matrix
```

	A	B	C	D	E
A	0.000000	14.142136	14.866069	2.828427	15.811388
B	14.142136	0.000000	1.000000	11.313708	3.162278
C	14.866069	1.000000	0.000000	12.041595	3.605551
D	2.828427	11.313708	12.041595	0.000000	13.038405
E	15.811388	3.162278	3.605551	13.038405	0.000000

위 거리 행렬에서 점들간의 거리를 살펴보면, 가장 거리가 가까운 두 점은 B,C라는 것을 확인할 수 있는데, 이는 (B,C)가 하나의 그룹으로 묶일 수 있다는 의미이다.

위 거리 행렬에 의해 가장 거리가 가까운 두 점 B,C는 하나의 그룹으로 하여. B,C를 새로운 점 B1라고 하고 A, D, E, B1 점의 거리행렬을 추가로 계산해 보자. 이 때, B1과 A의 거리를 구하기 위해서는 A와 B1의 내부의 각 점들과의 거리를 정의하는 로직이 필요하다. 한 점과 군집간의 거리 혹은 군집과 군집간의 거리를 정하는 방법에도 여러가지 종류가 있다.

1) 단일연결법(Single Linkage): 두 군집 내에서 가장 가까운 데이터 포인트 사이의 거리를 사용하여 군집 간의 거리를 계산한다.

2) 완전연결법(Complete Linkage): 두 군집 내에서 가장 먼 데이터 포인트 사이의 거리를 사용하여 군집 간의 거리를 계산한다.

3) 평균연결법(Average Linkage): 두 군집 내의 모든 데이터 포인트 사이의 거리를 평균하여 군집 간의 거리를 계산한다. 이 방법은 완전 연결법과 단일 연결법 사이의 중간 지점을 찾는 방법으로, 다양한 군집 형태를 잘 처리할 수 있다.

4) 중심연결법(Centroid Linkage): 두 군집의 중심(평균) 사이의 거리를 사용하여 군집 간의 거리를 계산한다. 이 방법은 군집의 중심을 고려하므로 비교적 균형 잡힌 군집을 형성하는 경향이 있다.

5) 워드연결법(Ward's Linkage): 두 군집을 병합한 후 새로운 군집의 분산을 최소화하는 방식으로 군집 간의 거리를 계산한다. 이 방법은 비교적 균형 잡힌 군집을 형성하고, 계층적 군집 분석에서 많이 사용된다.

먼저 단일연결법, 즉 거리들의 최소값을 중심으로 연결해 보도록 하자. 먼저, A,D,E 점들과 B1 내부 점들과의 최소값을 구하면 아래와 같다. 아래 왼쪽은 하나씩 코드를 작성한 것이고, 아래 오른쪽은 cdist라는 군집간 모든 점들의 거리를 계산하는 함수를 이용한 것이다.

```python
# B1과 각 점들과의 최소값 계산
from scipy.spatial.distance import minkowski

B1={'B': (10, 10), 'C': (11, 10)}
pts = {'A': (0, 0), 'D': (2, 2), 'E': (9, 13)}

res={} # 빈 사전형 자료
for name,coords in pts.items():
    tag=name+'&B1' # 사전에 들어갈 이름
    d=[] # 빈 리스트
    for j in B1.values():
        d.append(minkowski(coords,j,p=2))
    res[tag]=np.min(d) # 최소값을 value로 추가
res # 결과 출력
```

```
{'A&B1': 14.142135623730951,
 'D&B1': 11.313708498984761,
 'E&B1': 3.1622776601683795}
```

```python
# 군집간 개별 거리를 계산하는 cdist
from scipy.spatial.distance import cdist
B1={'B': (10, 10), 'C': (11, 10)}
pts = {'A': (0, 0), 'D': (2, 2), 'E': (9, 13)}

# B1 군집과 pts 군집간의 유클리디안 거리 계산
# metric 매개변수 추가 가능(디폴트 값 = 'euclidean')
linkage=cdist(list(B1.values()), list(pts.values()))

np.min(linkage,axis=0) # 세로 방향의 최소값 계산
```

```
array([14.14213562, 11.3137085 ,  3.16227766])
```

B1을 추가하여 거리 행렬을 다시 만들면 아래와 같다.

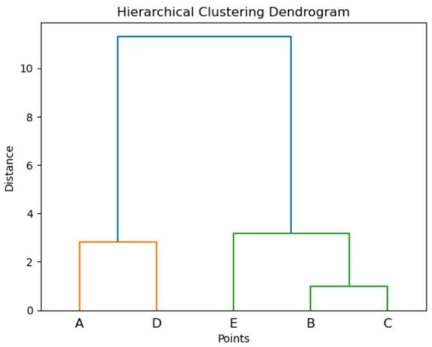
```python
# B1을 추가하여 행렬로 표시
df=df_dist_matrix
B1_dist=[14.14213562, 11.3137085 ,  3.16227766]
df['B1'] = B1_dist # 'B1'이라는 칼럼 추가
df=df.reindex(sorted(df.columns),axis=1) #칼럼 이름으로 소팅
#'B1'이라는 이름의 인덱스 행 추가
df.loc['B1']=[14.14213562, 0, 11.3137085 ,  3.16227766]
df.sort_index() # 인덱스이름으로 소팅
```

	A	B1	D	E
A	0.000000	14.142136	2.828427	15.811388
B1	14.142136	0.000000	11.313709	3.162278
D	2.828427	11.313709	0.000000	13.038405
E	15.811388	3.162278	13.038405	0.000000

위 거리행렬에서 가장 가까운 두 점은 'A', 'D'가 되며 이들은 그 다음 군집으로 묶일 수 있음을 의미한다. 이제, (B,C), (A,D), E의 3그룹으로 나누어지게 될 것이다.

군집분석은 위에서 설명한 로직에 의해 다양한 군집을 만들어 주게 된다. 'scipy'에는 위에서 설명한 작업을 한꺼번에 해주면서 그림[5]으로 계층적으로 보여주는 함수 'linkage'가 존재한다. 이 함수는 'method'라는 매개변수를 통해 군집간 거리 구하는 방식을 지정하게 되며, 'metric' 이라는 매개변수를 통해 점들의 거리 구하는 방식을 지정할 수 있다.

```python
from scipy.cluster.hierarchy import linkage, dendrogram
# 점의 좌표
points = {'A': (0, 0), 'B': (10, 10), 'C': (11, 10), 'D': (2, 2), 'E': (9, 13)}
df = pd.DataFrame(data=points)
df = df.T

# linkage 함수를 사용하여 계층적 군집화
linkage_result = linkage(df.values, method='single', metric='euclidean')

# 덴드로그램 그리기
dendrogram(linkage_result, labels=list(points.keys()))
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Points')
plt.ylabel('Distance')
plt.show()
```



위 오른쪽 결과를 보면, 앞에서 가장 가까운 두 점은 (B,C)이고 다음 가까운 두 점은 (A,D), 그 다음은

[5] 덴드로그램(dendrogram)이라고 부른다.

E와 (B,C) 등이 된다는 것을 살펴볼 수 있다.

이렇게 모든 점들간의 상호 거리를 계산하고, 군집간 거리결정방식(linkage)에 따라 하나씩 묶어 가는 방식을 계층적 군집분석(hierarchical cluster analysis)이라고 부른다[6].

문제2. 앞 문제에서 임의로 (A,B), (C,D,E) 2개의 군집으로 나누고, 초기값 C1=(0,1), C2=(2,0)와의 유클리디안 거리를 계산하여 그 결과를 출력해 보아라.

이 문제에 대한 해는 'cdist'를 이용하여 손쉽게 계산할 수 있다.

```
# 점의 좌표
points = {'A': (0, 0), 'B': (10, 10), 'C': (11, 10), 'D': (2, 2), 'E': (9, 13)}
cs={'C1':(0,1),'C2':(2,0)}

# points와 cs 점들과의 유클리디안 거리 계산
linkage=cdist(list(points.values()), list(cs.values()))

# 데이터프레임으로 전환
df = pd.DataFrame(data=linkage,columns=cs.keys(),index=points.keys())
df
```

	C1	C2
A	1.000000	2.000000
B	13.453624	12.806248
C	14.212670	13.453624
D	2.236068	2.000000
E	15.000000	14.764823

위 결과 값을 기준으로 어느 쪽 값이 더 작은 지 확인해 보자. 아래 결과를 살펴보면 A는 C1에 더 가깝고, B,C,D,E는 C2에 더 가깝다는 것을 알 수 있다[7].

```
# C1<C2보다 작은지 판단
df['판단']=df.C1<df.C2
df
```

	C1	C2	판단
A	1.000000	2.000000	True
B	13.453624	12.806248	False
C	14.212670	13.453624	False
D	2.236068	2.000000	False
E	15.000000	14.764823	False

---

[6] 계층적 군집분석 방법은 데이터양이 큰 경우에는 많은 계산시간을 필요로 하게 되어, 빅데이터 분석시에는 잘 사용되지 않는다.

[7] C1에 가까우면 True, 아니면 False로 판단함

이제, C1의 좌표를 A, C2의 좌표를 (B,C,D,E)의 중심값으로 갱신하여 위의 작업을 다시한번 해 보도록 하자.

```
# 중심값(centroid) 계산
df_points=pd.DataFrame(data=points)
df_points=df_points.T # 행과 열을 뒤바꿈
df_points.columns=['x','y'] # 칼럼이름 갱신
# df['판단'] 칼럼의 False인 index 행들만의 평균계산
df_points.loc[df[~df.판단].index].mean()
```

```
x    8.00
y    8.75
dtype: float64
```

```
# numpy array로 변환
C1=tuple(df_points.loc[df[df.판단].index].mean())
C2=tuple(df_points.loc[df[~df.판단].index].mean())
cs={'C1':C1,'C2':C2}
cs
```

```
{'C1': (0.0, 0.0), 'C2': (8.0, 8.75)}
```

```
# points와 cs 점들과의 유클리디안 거리 계산
linkage=cdist(list(points.values()), list(cs.values()))

# 데이터프레임으로 전환
df = pd.DataFrame(data=linkage,columns=cs.keys(),index=points.keys())
df['판단']=df.C1<df.C2
df
```

	C1	C2	판단
A	0.000000	11.855906	True
B	14.142136	2.358495	False
C	14.866069	3.250000	False
D	2.828427	9.031196	True
E	15.811388	4.366062	False

중심값을 갱신하였더니, (A,D), (B,C,E) 값으로 나눌 수 있다는 것을 알 수 있는데, 이 방식에 의하면 자료가 많더라도 훨씬 빨리 군집을 나눌 수 있다는 것을 알 수 있다. 이 방식을 K-Means 군집분석이라고 부르며, 비계층적 군집분석 방법의 하나에 속한다.

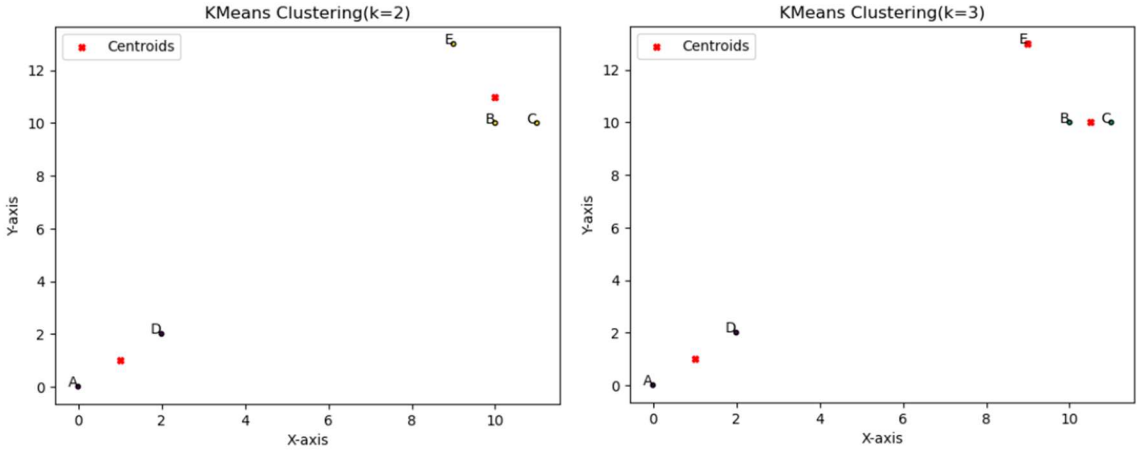
K-Means 군집분석도 'scipy'의 'kmeans'를 이용하면 원하는 결과를 쉽게 얻을 수 있다.

```
import numpy as np
from scipy.cluster.vq import kmeans, vq
import matplotlib.pyplot as plt

# 주어진 좌표 데이터
points = {'A': (0, 0), 'B': (10, 10), 'C': (11, 10), 'D': (2, 2), 'E': (9, 13)}

# 좌표 데이터를 NumPy 배열로 변환 (dtype=float)
X = np.array(list(points.values()), dtype=float)

# 적절한 K 값 선택 (예: K=2)
k = 2

# KMeans 군집분석 수행
# vq함수는 군집의 인덱스를 반환해줌
centroids, _ = kmeans(X, k) # centroids에는 군집의 중심점들이 저장됨
labels, _ = vq(X, centroids) # label에는 각 포인터가 속한 군집번호가 있음
```

위 군집분석의 결과를 시각화하여 보자. 군집을 2개로 나눌 때(k=2)는 (A,D), (B,C,E)로 구분된다는 것을 알 수 있다.

```python
# 군집화 결과를 시각화
# c=labels : labels 숫자에 따라 다른 색깔로 표시
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=10, edgecolors='k')
plt.scatter(centroids[:, 0], centroids[:, 1], marker='X', s=20, c='red', label='Centroids')

# 각 점에 텍스트 추가
# zip()은 각 객체로 부터 하나씩 가져와 묶어주는 역할을 함
for label, (x, y) in zip(points.keys(), X):
    # ha: horizaontal alignment로 텍스트의 수평정렬(left,right,center)
    plt.text(x, y, label, fontsize=10, ha='right')
------
plt.title('KMeans Clustering')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend()
plt.show()
```

아래 그림의 오른쪽 그림은 군집을 3개로 나눌 때의 결과를 추가로 보여주고 있는데, 이 때는 (A,D), (B,C), E 3개로 구분된다는 것을 보여주고 있다.



군집이 3개인 경우로 나눈 결과를 데이터프레임으로 만들고, 군집별 특성치들을 feature별로 살펴볼 수도 있다.

```python
# 3개의 군집으로 나눈 후, 데이터프레임으로 다시 보기
import pandas as pd
df=pd.DataFrame(X,columns=['feature1','feature2'],index=points.keys())
df['Cluster']=labels
df
```

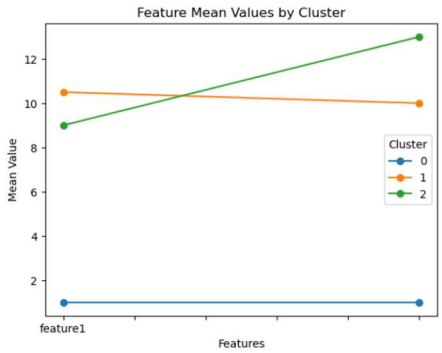
	feature1	feature2	Cluster
A	0.0	0.0	0
B	10.0	10.0	1
C	11.0	10.0	1
D	2.0	2.0	0
E	9.0	13.0	2

```
# 클러스터별로 각 feature에 대한 기본 통계량 살펴보기
df.groupby('Cluster').describe()
```

	feature1								feature2							
	count	mean	std	min	25%	50%	75%	max	count	mean	std	min	25%	50%	75%	max
Cluster																
0	2.0	1.0	1.414214	0.0	0.50	1.0	1.50	2.0	2.0	1.0	1.414214	0.0	0.5	1.0	1.5	2.0
1	2.0	10.5	0.707107	10.0	10.25	10.5	10.75	11.0	2.0	10.0	0.000000	10.0	10.0	10.0	10.0	10.0
2	1.0	9.0	NaN	9.0	9.00	9.0	9.00	9.0	1.0	13.0	NaN	13.0	13.0	13.0	13.0	13.0

각 feature별로 군집별 평균값들을 비교해 그려보면 아래와 같이 할 수 있다. 아래 결과를 보면 모든 feature에서 cluster 0의 평균값들이 가장 작고, feature1 기준으로 cluster 1이 가장 크며, feature 2 기준으로는 cluster 2의 평균값이 가장 크다는 것을 한 눈에 파악할 수 있다.

```
# 각 feature에 대한 군집별 평균값 그래프 그리기
df.groupby('Cluster').mean().T.plot(kind='line', marker='o')

# 그래프에 제목과 라벨 추가
plt.title('Feature Mean Values by Cluster')
plt.xlabel('Features')
plt.ylabel('Mean Value')

# 그래프 표시
plt.show()
```

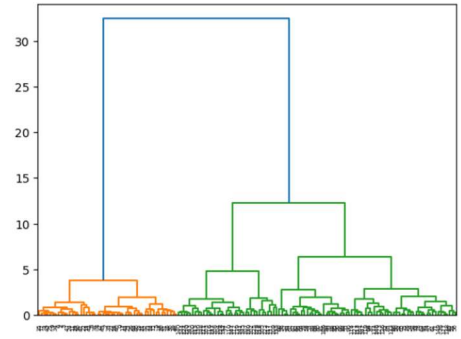


문제3. 앞에서 사용했던 scikit-learn의 붓꽃 데이터 자료를 다시 불러오자. 붓꽃의 feature 데이터 150개를 기반으로 각 자료의 거리를 계산하고, 이를 바탕으로 비슷한 거리끼리 묶어서 그려보는 '덴드로그램(Dendrogram)'을 그려보아라.

'scipy' 라이브러리에 있는 'hierarchy' 도구를 사용해서 분석해 보면 아래와 같은 덴드로그램을 볼 수 있는데 크게는 2개, 작게는 3개의 그룹으로 나누어지는 것 같아 보인다.

```
# 붓꽃 데이터 읽어들이기
from sklearn.datasets import load_iris
import numpy as np
import pandas as pd

IR=load_iris()
df=pd.DataFrame(data=IR.data,columns=IR.feature_names)
```

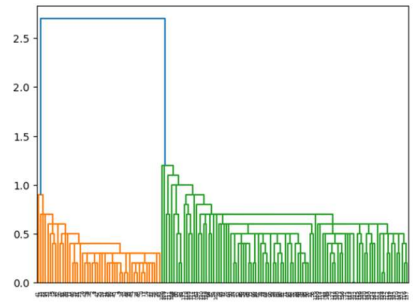
```python
# 덴드로그램 그리기
from scipy.cluster import hierarchy #.계층적 군집분석 도구
import matplotlib.pyplot as plt #.그래프 그리는 도구

# IR.data를 바탕으로 ward 연결방법에 의해 계층적 군집분석 수행
Z=hierarchy.linkage(IR['data'],method='ward')


# 덴드로그램으로 보여주기
plt.figure()
out=hierarchy.dendrogram(Z)
```



위 분석에서 데이터간 거리를 계산하는 방식을 따로 지정하지 않았는데, 이 경우에는 유클리디언 거리가 디폴트 값으로 선정된다. 만일 다른 거리 측정방식을 원할 경우 매개변수 metric을 사용하여 별도로 지정할 수 있다.

데이터간의 거리는 'cityblock'으로 하고, 연결방법은 단일연결법에 의해서 덴드로그램을 그려보면 아래와 같다.

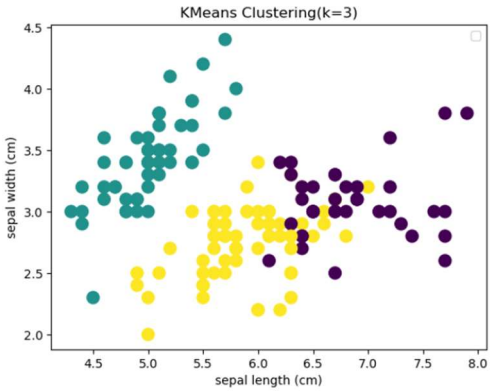
```python
# 데이터 거리 계산: 맨하탄 거리
# 클러스터 거리계산 방법: single linkage
Z=hierarchy.linkage(IR['data'],
                    method='single',
                    metric='cityblock')

# 덴드로그램으로 보여주기
plt.figure()
out=hierarchy.dendrogram(Z)
```



문제4. 붓꽃 데이터의 feature 데이터들을 3개의 군집으로 나누고 각 feature별로 평균값들을 비교해 보아라.

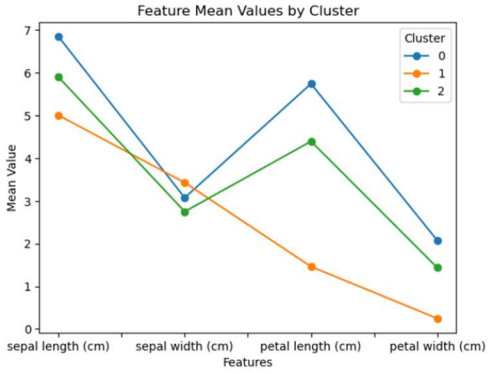
'scipy' 라이브러리의 'kmeans'에 의해 군집분석을 수행해 보도록 하자.

```python
from scipy.cluster.vq import kmeans,vq

k=3 # 군집개수 설정
X=IR['data'] # feature 데이터 선택

# KMeans 군집분석 수행
centroids, _ = kmeans(X, k)
labels, _ = vq(X, centroids)

# 군집화 결과를 시각화, 처음 2개의 feature에 대해서만 그려보기
plt.scatter(X[:, 0], X[:, 1], c=labels,
            cmap='viridis', s=100)
plt.title('KMeans Clustering(k=3)')
plt.xlabel(IR['feature_names'][0])
plt.ylabel(IR['feature_names'][1])
plt.legend()
plt.show()
```



각 feature들별 군집들의 평균값을 비교해 보도록 하자.

```python
# 각 feature에 대한 군집별 평균값 그래프 그리기
import pandas as pd
df=pd.DataFrame(X,columns=IR.feature_names)
df['Cluster']=labels # 새 칼럼 생성

# 클러스터 별로 평균값 계산하기
df.groupby('Cluster').mean().T.plot(kind='line', marker='o')

# 그래프에 제목과 라벨 추가
plt.title('Feature Mean Values by Cluster')
plt.xlabel('Features')
plt.ylabel('Mean Value')

# 그래프 표시
plt.show()
```

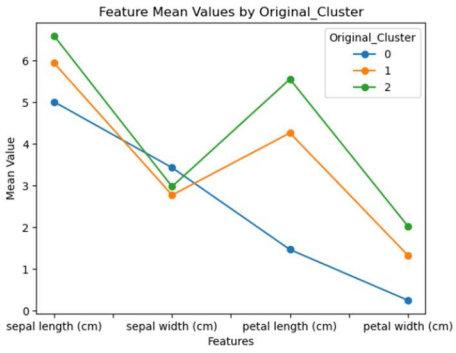


IR['target']에는 각 자료에 원래 품종 관련 분류값이 들어 있다. 원래 값을 바탕으로 위의 그래프를 다시 그려보면 아래와 같다. 아래에서 'Orginial_Cluster' 값이 0,1,2는 각각 'setosa', 'versicolor','virginica'에 해당하는 품종으로서 위 군집분석 결과와 비교해 보면, Cluster 0이 'virginica', 1이 'setosa', 2가 'versicolor' 임을 추정할 수 있다.

```python
IR.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

```python
# 원래의 target 결과와 비교해서 보기
import pandas as pd
df=pd.DataFrame(X,columns=IR.feature_names)
df['Original_Cluster']=IR.target

# 클러스터 별로 평균값 계산하기
df.groupby('Original_Cluster').mean().T.plot(kind='line', marker='o')

# 그래프에 제목과 라벨 추가
plt.title('Feature Mean Values by Original_Cluster')
plt.xlabel('Features')
plt.ylabel('Mean Value')

# 그래프 표시
plt.show()
```



군집분석에 의한 결과값과 원래의 결과값을 비교하기 위해 군집분석에서의 숫자를 바꾸고 원래의 값들을 얼마나 복원했는지 계산해 보도록 하자.

```python
df=pd.DataFrame(X,columns=IR.feature_names)
df['Cluster']=labels # 새 칼럼 추가
df['Original_Cluster']=IR.target # 새 칼럼추가
# df.Cluster 열의 값 변환
cluster_mapping = {0: 2, 1: 0, 2: 1} # 매핑할 딕셔너리 정의
df['Cluster'].replace(cluster_mapping, inplace=True)

# 두 칼럼의 서로 같은 원소가 있는지 참, 거짓 판별
different_elements = df['Cluster'] == df['Original_Cluster']

# 정답율 계산
count_different_elements = different_elements.sum()
accuracy_percentage = count_different_elements / len(df.Cluster) * 100
# 소수 둘째 자리 까지만 출력하도록 포맷지정
formatted_accuracy = "{:.2f}".format(accuracy_percentage)
print(f"정답율: {formatted_accuracy}%")
```

```
정답율: 89.33 %
```

참고문헌

1. https://github.com/HST0077/Shinhan_AI_2025/blob/main/clustering.ipynb