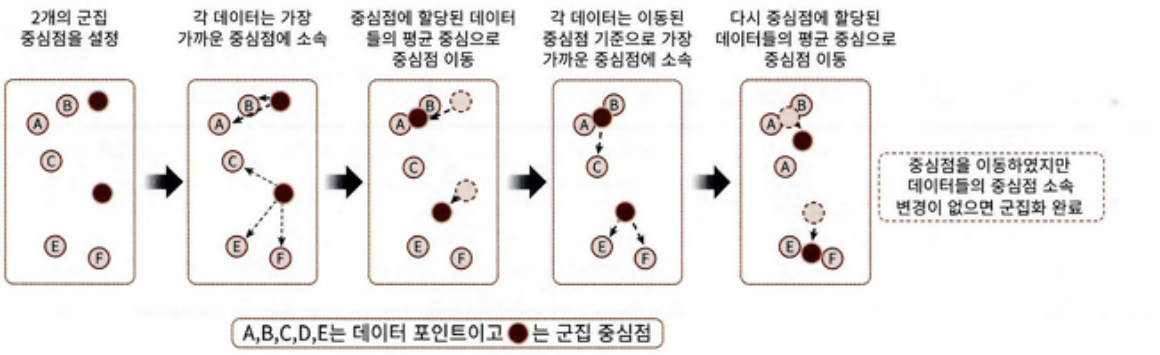
❤️

07 군집화 431p~481p

Week11_예습과제_권혜수.ipynb

01 K - 평균 알고리즘 431p

- 군집 중심점이라는 특정한 임의의 지점을 선택해 해당 중심에 가장 가까운 포인트들을 선택
- 일반적인 군집화에서 가장 많이 활용되는 알고리즘 & 간결



하지만

- 속성의 개수가 매우 많을 경우 군집화 정확도가 떨어짐(PCA 필요할 수도)
- 반복 횟수가 많을 경우 수행 시간 느려짐
- 몇 개의 군집(cluster)을 선택?

사이킷런 KMeans 클래스

```
class sklearn.cluster.KMeans(n_clsuter=8, init = 'k-means++', n_init=10, ma
x_iter=300, tol=0.0001,
                    precompute_distances='auto', verbose = 0, random_state
```

```
=None,
                                    copy_x=True, n_jobs=None, algorithm='auto')
```

기본 파라미터

- **n_clsuter=8 가장 중요 군집 중심점의 개수**
- init = 초기 군집 중심점의 좌표. 보통 기본 값
- max_iter= 최대 반복 횟수이며, 이 횟수 이전에 모든 데이터의 중심점 이동이 없으면 종료
- fit() fit_transform() → 군집화 완료 관련 주요 속성 → labels_(각 데이터 포인트가 속한 군집 중심점 레이블), cluster_centers_(각 군집 중심점 좌표 Shape는 [군집 개수, 피처 개수])

붓꽃 데이터 세트 군집화 433p

```
from sklearn.preprocessing import scale
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
%matplotlib inline

iris = load_iris()
df = pd.DataFrame(iris.data, columns=['sepal_length', 'sepal_width', 'petal_l
ength', 'petal_width'])

kmeans = KMeans(n_clusters = 3, init='k-means++', max_iter=300, random
_state=0)
kmeans.fit(df)
print(kmeans.labels_)
```

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

labels—의 값이 0, 1, 2로 돼 있으며, 이는 각 레코드가 첫 번째 군집, 두 번째 군집, 세 번째 군집에 속함

```
df['target'] = iris.target
df['cluster'] = kmeans.labels_
df_result = df.groupby(['target', 'cluster'])['sepal_length'].count()
print(df_result)
```

```
target  cluster
0       1          50
1       0          47
        2           3
2       0          14
        2          36
Name: sepal_length, dtype: int64
```

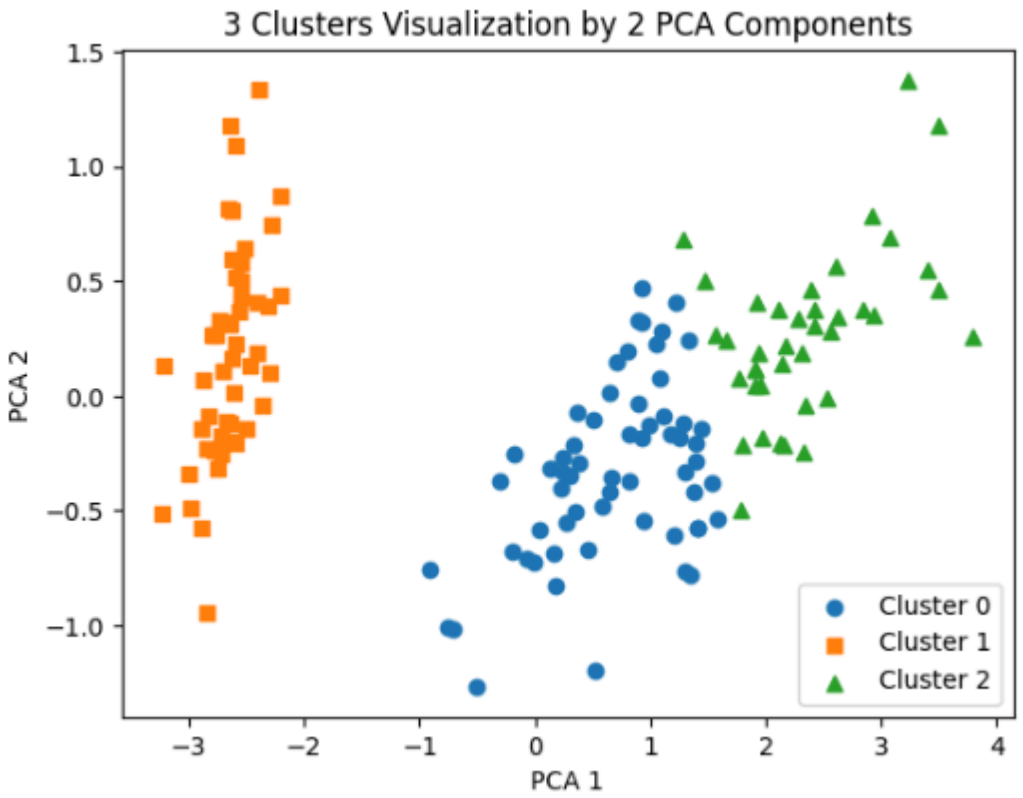
분류 타깃이 o값인 데이터는 1번 군집으로 모두 잘 그루핑

Target 1 값 데이터는 2개만 2번군집으로 그루핑됐고, 나머지 48개는 모두 0번 군집으로 그루핑

Target 2값 데이터는0번 군집에 14개, 2번 군집에 36개로 분산

```
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)
pca_transformed = pca.fit_transform(iris.data)

df['pca_x'] = pca_transformed[:,0]
df['pca_y']= pca_transformed[:, 1]

# 군집 값이 0, 1, 2인 경우마다 별도의 인덱스로 추출
marker0_ind = df[df['cluster'] == 0].index
marker1_ind = df[df['cluster'] == 1].index
```

```
marker2_ind = df[df['cluster'] == 2].index

plt.scatter(x=df.loc[marker0_ind, 'pca_x'], y=df.loc[marker0_ind, 'pca_y'], m
arker='o',label='Cluster 0')

plt.scatter(
    x=df.loc[marker1_ind, 'pca_x'],
    y=df.loc[marker1_ind, 'pca_y'],
    marker='s',
    label='Cluster 1'
)

plt.scatter(
    x=df.loc[marker2_ind, 'pca_x'],
    y=df.loc[marker2_ind, 'pca_y'],
    marker='^',
    label='Cluster 2'
)

plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.title('3 Clusters Visualization by 2 PCA Components')
plt.legend()
plt.show()
```



Cluster 0을 나타내는 동그라미('o')와Cluster 2를 나타내는 세모(, A,)는상당수준 분리돼 있지만, 네모만큼 명확하게는 분리돼 있지 않음

군집화 알고리즘 테스트를 위한 데이터 생성 437p

군집화용 데이터 생성기로는 `make_blobs( )` 와 `make classification( )` API

- 하나의 클래스에 여러 개의 군집이 분포될 수 있게 데이터를 생성할 수 있음
- make_blobs( )는 개별 군집의 중심점과 표준 편차 제어 기능이 추가
- make_classifcation()은 노이즈를 포함한 데이터 생성
- `make_circle()` , `make_moon()` 중심 기반의 군집화로 해결하기 어려운 데이터 세트를 만드는 데 사용

make_blobs()

- 피처 데이터 세트와 타깃 데이터 세트가 **튜플**(Tuple)로 반환

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
%matplotlib inline

X, y = make_blobs(n_samples=200, n_features=2, centers=3, cluster_std=
0.8, random_state=0)
print(X.shape, y.shape)
unique, counts = np.unique(y, return_counts=True)
print(unique, counts)
```

```
(200, 2) (200,)
[0 1 2] [67 67 66]
```

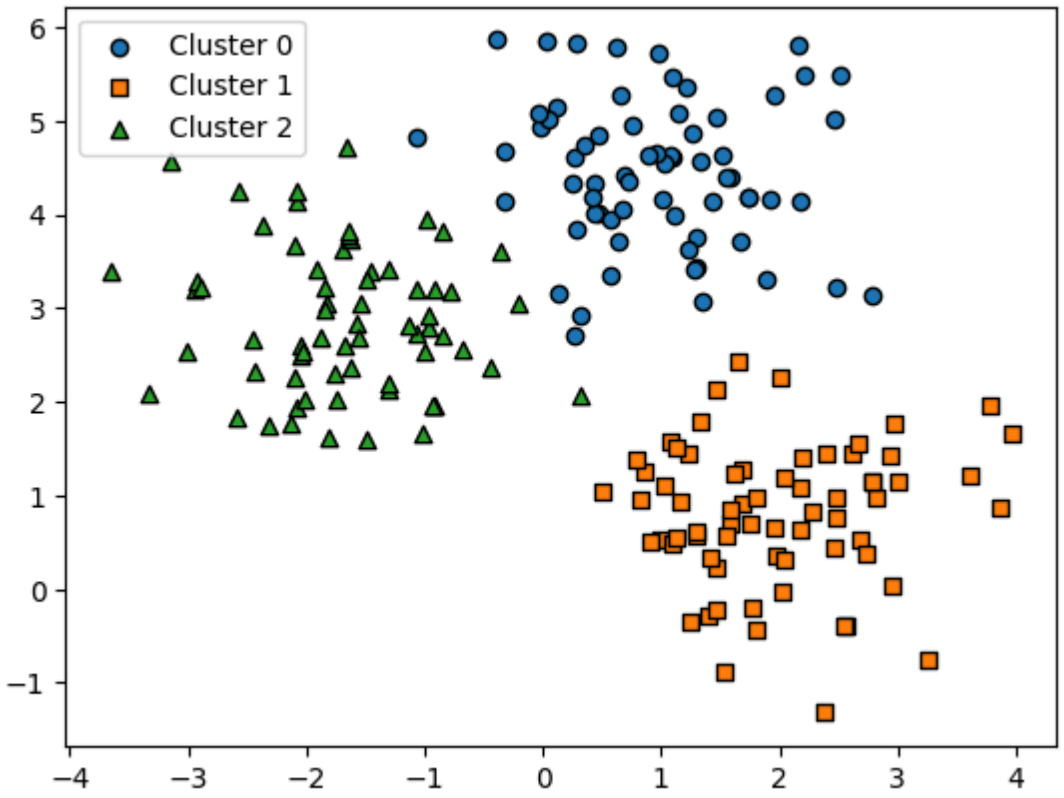
클러스터 값은 [0, 1, 2] 각각 균일하게 구성

```
import pandas as pd

clusterDF = pd.DataFrame(data=X, columns=['ftr1', 'ftr2'])
clusterDF['target'] = y
```

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

clusterDF = pd.DataFrame(data=X, columns=['ftr1', 'ftr2'])
clusterDF['target'] = y

target_list = np.unique(y)

# 각 타깃별 산점도 마커
markers = ['o', 's', '^', 'P', 'D', 'H', 'x']   # 필요 이상 많이 넣어도 됨
```

```
# target_list = [0,1,2]라면 군집 3개
for target in target_list:
    target_cluster = clusterDF[clusterDF['target'] == target]

    plt.scatter(
        x=target_cluster['ftr1'],
        y=target_cluster['ftr2'],
        edgecolor='k',
        marker=markers[target],
        label=f"Cluster {target}"
    )

plt.legend()
plt.show()
```

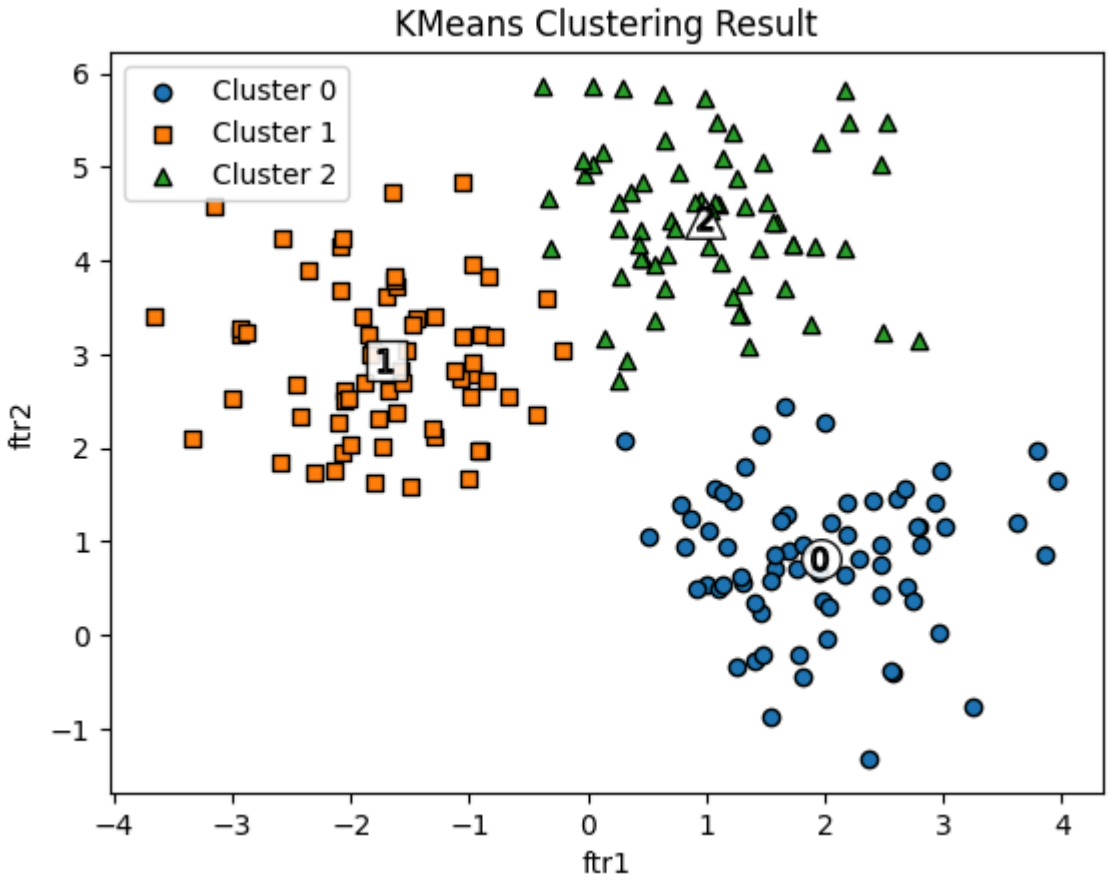
make_blob()으로 만든 피처 데이터 세트가 어떠한 군집화 분포?



```
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt

# KMeans 모델 생성 및 학습
kmeans = KMeans(
    n_clusters=3,
    init='k-means++',   # ← 올바른 옵션
    max_iter=200,
    random_state=0
)

cluster_labels = kmeans.fit_predict(X)

# 결과를 DataFrame에 저장
clusterDF['kmeans_label'] = cluster_labels

# 클러스터 중심
centers = kmeans.cluster_centers_
unique_labels = np.unique(cluster_labels)

# 사용할 마커 리스트
markers = ['o', 's', '^', 'P', 'D', 'H', 'x']

# 군집별 시각화
for label in unique_labels:
    label_cluster = clusterDF[clusterDF['kmeans_label'] == label]
    center_x_y = centers[label]

    # 각 cluster 산점도
    plt.scatter(
        x=label_cluster['ftr1'],
        y=label_cluster['ftr2'],
        edgecolor='k',
        marker=markers[label],
        label=f"Cluster {label}"
    )
```

```
    # 군집 중심(큰 흰 점)
    plt.scatter(
        x=center_x_y[0],
        y=center_x_y[1],
        s=200,
        color='white',
        alpha=0.9,
        edgecolor='k',
        marker=markers[label]
    )

    # 중심에 cluster 번호 표시
    plt.scatter(
        x=center_x_y[0],
        y=center_x_y[1],
        s=70,
        color='none',
        edgecolor='k',
        marker='$%d$' % label   # 텍스트 마커
    )

  plt.legend()
  plt.title("KMeans Clustering Result")
  plt.xlabel("ftr1")
  plt.ylabel("ftr2")
  plt.show()
```

만들어진 데이터 세트에 KMeans 군집화를 수행한 뒤에 군집별로 시각화



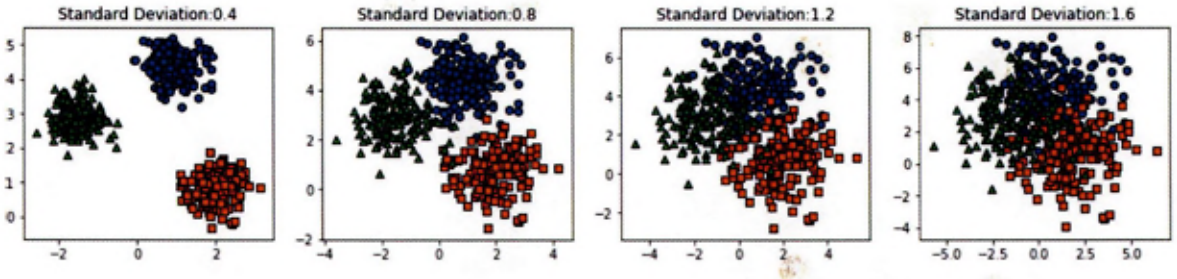
make_blobs() 의 타깃과 kmeansjabel은 군집 번호를 의미하므로 서로 다른 값으로 매핑 가능

```
print(clusterDF.groupby('target')['kmeans_label'].value_counts())
```

```
target  kmeans_label
0       2              66
        1               1
1       0              67
2       1              65
        0               1
Name: count, dtype: int64
```

거의 대부분 잘 매핑

- make_blobs()은 cluster_std 파라미터로 데이터의 분포도를 조절

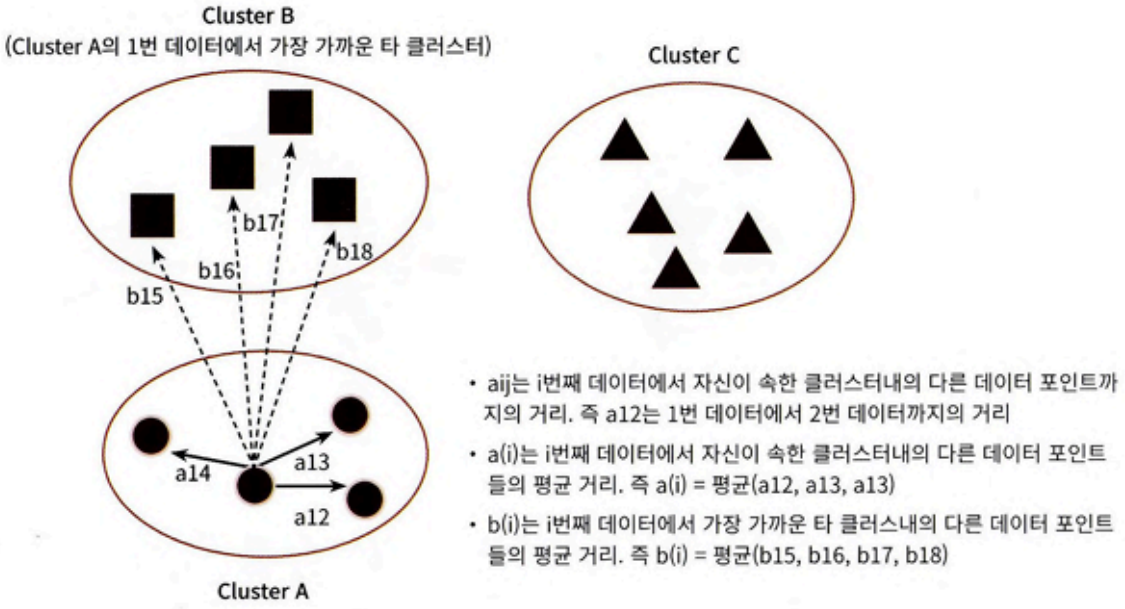


군집 평가 441

실루엣 분석의 개요

각 군집 간의 거리가 얼마나 효율적으로 분리돼 있는지

- 개별 군집은 비슷한 정도의 여유공간을 가지고 떨어져 있을 것
- 실루엣 계수(silhouette coefficient): 해당 데이터가 같은 군집 내의 데이터와 얼마나 가깝게 군집화돼 있고, 다른 군집에 있는 데이터와는 얼마나 멀리 분리돼 있는지



두 군집 간의 거리가 얼마나 떨어져 있는가의 값은 b(i) - a(i), 이 값을 정규화하기 위해 MAX(a(i), b(i)) 값으로 나눔

$$s(i) = \frac{(b(i) - a(i))}{(\max(a(i), b(i))}$$

- 1로 가까워질수록 근처의 군집과 더 멀리 떨어져 있음
- 0에 가까울수록 근처의 군집과 가까워짐
- - 값은 아예 다른 군집에 데이터 포인트가 할당됐음

사이킷런

`sklearn.metrics.silhouette_samples(X, labels, metric='euclidean')`

- **X**: 원본 feature 데이터
- **labels**: 각 샘플이 속한 군집 번호
- 모든 데이터 포인트에 대해 **개별 실루엣 계수(silhouette coefficient)** 를 배열 형태로 반환

sklearn.metrics.silhouette_score(X, labels, metric='euclidean', sample_size=None)

- **X**: feature 데이터
- **labels**: 군집 번호
- 전체 데이터의 **평균 실루엣 계수 → 이 값이 높을수록 군집화 잘됨(항상 X)**

붓꽃 데이터

`silhouette_samples()` 와 `silhouette_score()`

```
from sklearn.preprocessing import scale
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

```
%matplotlib inline

# 데이터 로드
iris = load_iris()

feature_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
irisDF = pd.DataFrame(data=iris.data, columns=feature_names)

# KMeans 군집화
kmeans = KMeans(
    n_clusters=3,
    init='k-means++',
    max_iter=300,
    random_state=0
).fit(irisDF)

irisDF['cluster'] = kmeans.labels_

# 개별 실루엣 계수
score_samples = silhouette_samples(iris.data, irisDF['cluster'])
print('silhouette_samples() return 값의 shape:', score_samples.shape)

# DF에 실루엣 계수 추가
irisDF['silhouette_coef'] = score_samples

# 평균 실루엣 계수
average_score = silhouette_score(iris.data, irisDF['cluster'])
print('붓꽃 데이터 세트 Silhouette Analysis Score: {:.3f}'.format(average_scor
e))

irisDF.head(3)
```

```
silhouette_samples() return 값의 shape: (150,)
붓꽃 데이터 세트 Silhouette Analysis Score: 0.551
```

	sepal_length	sepal_width	petal_length	petal_width	cluster	silhouette_coef
0	5.1	3.5	1.4	0.2	1	0.852582
1	4.9	3.0	1.4	0.2	1	0.814916
2	4.7	3.2	1.3	0.2	1	0.828797

```
1 irisDF.groupby('cluster')['silhouette_coef'].mean()
```

cluster	silhouette_coef
0	0.422323
1	0.797604
2	0.436842

붓꽃 데이터 세트의 평균 실루엣 계수 값은 약 0.553

클러스터마다 달라서 평균이 영향 받음

군집별 평균 실루엣 계수의 시각화를 통한 군집 개수 최적화 방법

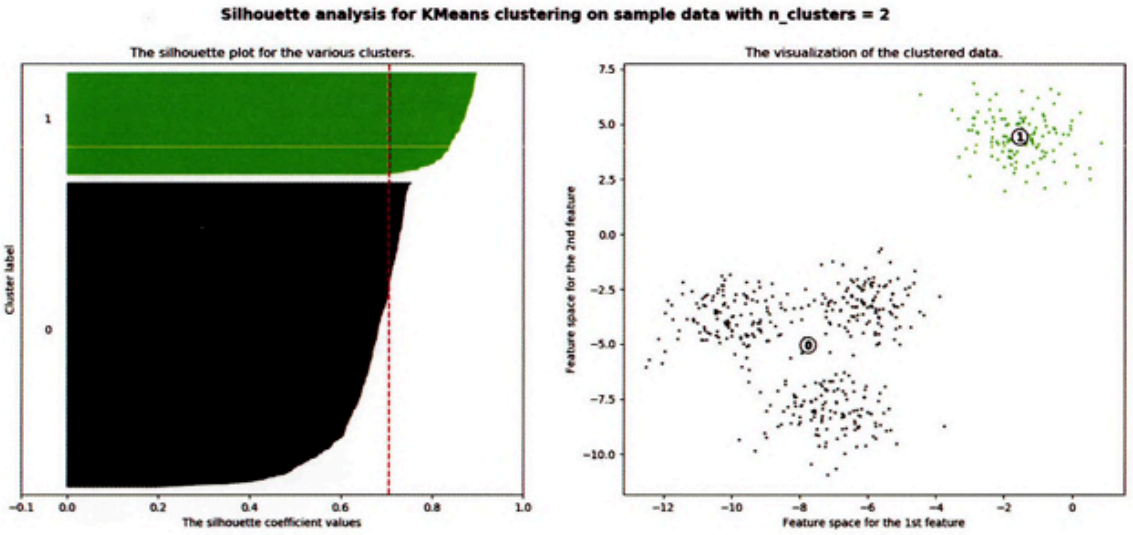
전체 데이터의 평균 실루엣 계수가 높아도 **반드시 최적의 군집 개수라고 볼 수는 없다.**

왜냐하면 특정 군집만 실루엣 값이 매우 높고, **다른 군집은 분리도가 낮아 값이 떨어지더라도**

전체 평균은 높게 유지될 수 있기 때문이다.

따라서

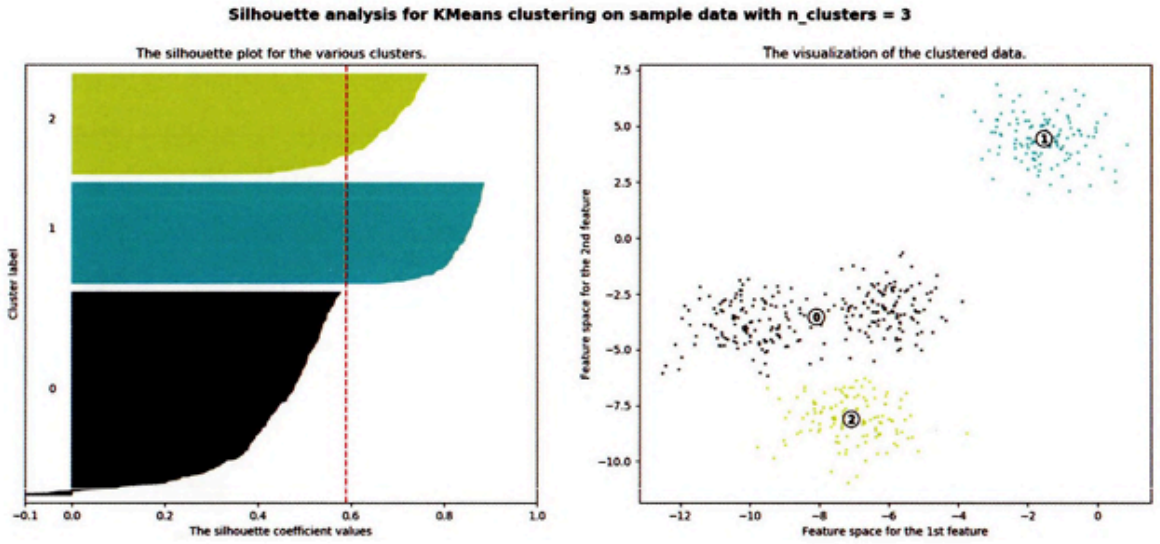
- 각 **개별 군집별 실루엣 값 분포**를 함께 확인하고,
- 군집들이 **서로 적당히 분리**되어 있으면서
- **군집 내부는 잘 뭉쳐 있는지** 함께 평가해야 한다.



군집이 2개일 경우 평균 실루엣 계수 값: 0.704

군집의 개수 2개

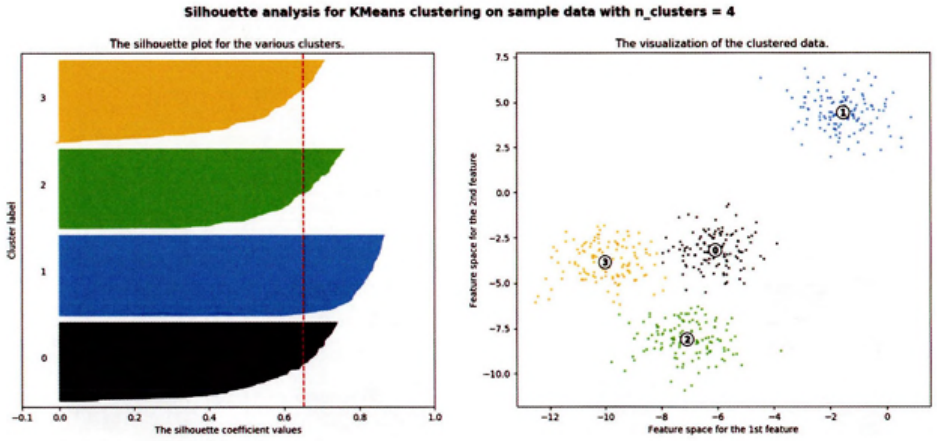
- 평균 실루엣 계수, 즉 silhouette_score는 약 0.704로 매우 높게 나타남
- 2번 군집의 경우는 평균보다 적은 데이터 값이 매우 많음



군집이 3개일 경우 평균 실루엣 계수 값: 0.588

군집 개수가 3개일 경우

- 전체 데이터의 평균 실루엣 계수 값은 약 0.588

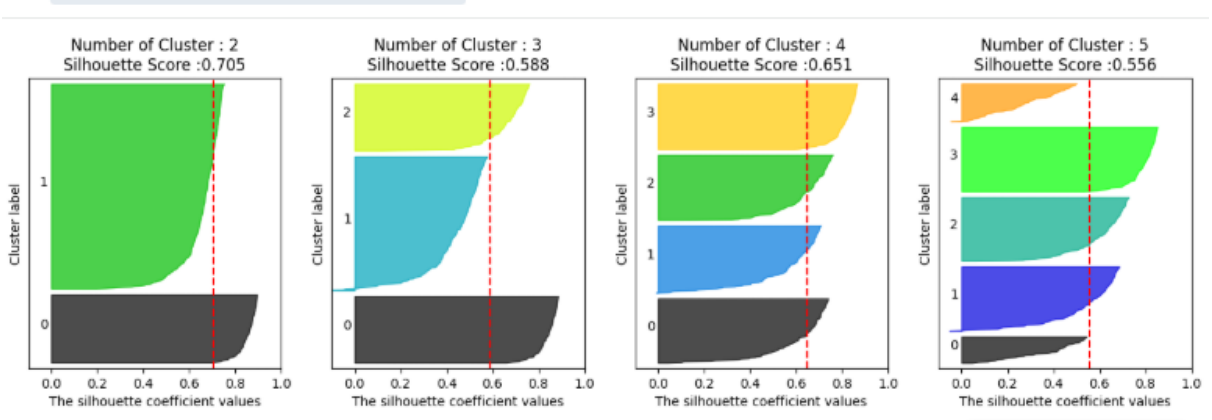


군집이 4개일 경우 평균 실루엣 계수 값: 0.65

군집이 4개

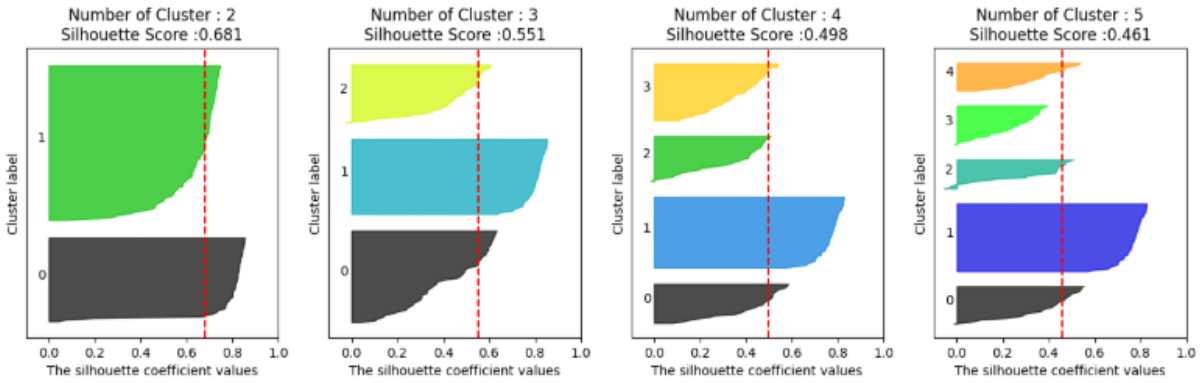
- 평균 실루엣 계수 값은 약 0.65
- 평균 실루엣 계수 값이 작지만 4개인 경우가 가장 이상적

```
from sklearn.datasets import make_blobs

X, y = make_blobs(n_samples=500, n_features=2, centers=4, cluster_std=
1,
                  center_box=(-10.0, 10.0), shuffle=True, random_state=1)
 # 군집 개수가 2개, 3개, 4개, 5개일 때의 군집별 실루엣 계수 평균값을 시각화
visualize_silhouette([ 2, 3, 4, 5], X)
```



make_blobs() 함수를 통해 4개 군집 중심의 500개 2차원 데이터 세트 → K-평균으로 군집화

```
from sklearn.datasets import load_iris
iris=load_iris()
visualize_silhouette([ 2, 3, 4, 5 ], iris.data)
```



붓꽃 데이터

- 군집 개수를 2개가 제일 나음

몇 만 건 이상의 데이터 → 군집별로 임의의 데이터를 샘플링해 실루엣 계수를 평가하는 방안

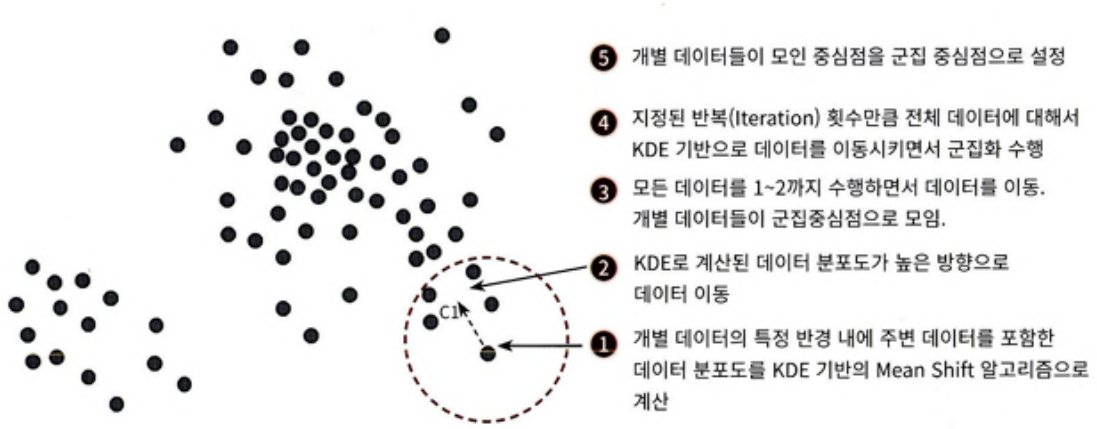
평균 이동

Mean Shift는 데이터가 **가장 밀집된 방향(밀도 peak)** 으로 중심을 계속 이동시켜 군집을 찾는 방법이다.

- K-means는 **평균 위치**로 중심을 옮기지만,
- Mean Shift는 **밀도가 높은 곳**으로 중심을 옮긴다.

데이터 밀집도를 추정하기 위해 **KDE(커널 밀도 추정)** 를 사용하며,

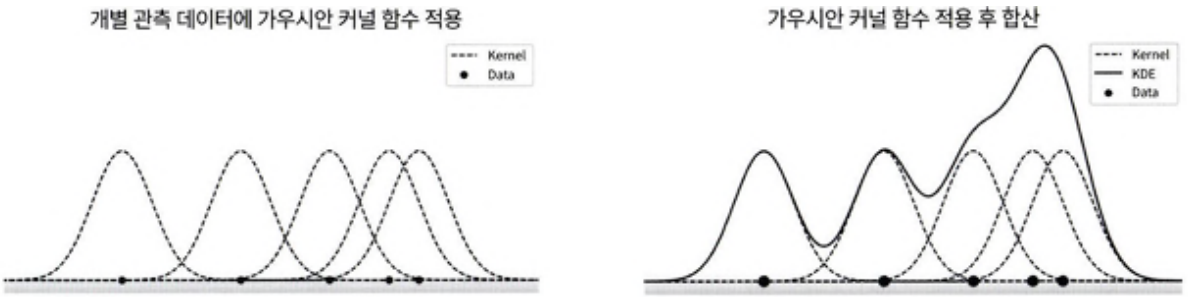
각 점은 주변 데이터의 밀도값을 따라 반복적으로 이동하여 **밀도 peak를 군집 중심**으로 만든다.



KDE는 **커널 함수(보통 가우시안)** 를 사용해 관측 데이터의 **확률 밀도 함수(PDF)** 를 추정하는 방법이다.

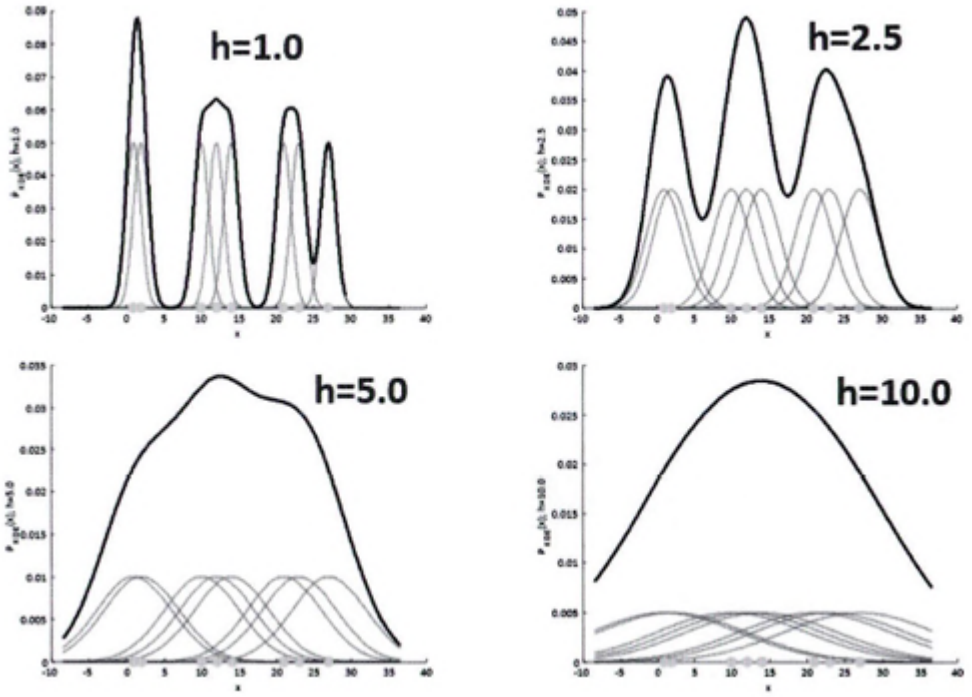
각 데이터 점에 커널을 씌운 값을 **모두 더한 뒤 데이터 개수로 나누어** 매끄러운 밀도 곡선을 만든다.

- PDF(Probability Density Function)는 확률변수의 분포(정규분포 등)를 나타내는 함수
- KDE는 각 데이터 주변에 작은 '봉우리'를 만들고 이를 합쳐 전체 분포 형태를 추정하는 방식
    - 개별 관측 데이터에 커널 함수를 적용한 뒤, 이 적용 값을 모두 더한 후 개별 관측 데이터의 건수로 나눠 확률 밀도 함수를 추정
    - 커널 → 가우시안 분포 함수 대표적



$$\text{KDE} = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

KDE의 대역폭 h는 밀도 곡선을 얼마나 **부드럽게 만들지** 결정하는 값이다.



- **h가 작으면** KDE가 뾰족해져 **과적합(overfitting)**
- **h가 크면** KDE가 너무 평평해져 **과소적합(underfitting)**

따라서 적절한 h 선택은 KDE뿐 아니라, KDE를 기반으로 군집 중심을 찾는 **Mean Shift 군집화에서 매우 중요하다.**

- Mean Shift의 **군집 개수는 대역폭 h에 의해 결정**된다.
  - **h가 크면** KDE가 평활화되어 → **적은 군집**
  - **h가 작으면** KDE가 세분화되어 → **많은 군집**
- Mean Shift는 K-means처럼 군집 수 k를 지정하지 않고, **오직 bandwidth 값만으로** 군집 수가 자동으로 정해진다.
- Scikit-learn에서 MeanShift는 `bandwidth` (커널(가우시안)의 폭(넓이))로 h를 설정하며, 최적의 h 추정을 위해 **estimate_bandwidth()** 함수를 제공한다.

```
from sklearn.cluster import estimate_bandwidth

bandwidth = estimate_bandwidth(X)

print("bandwidth 값:", round(bandwidth, 3))
```

bandwidth 값: 1.816

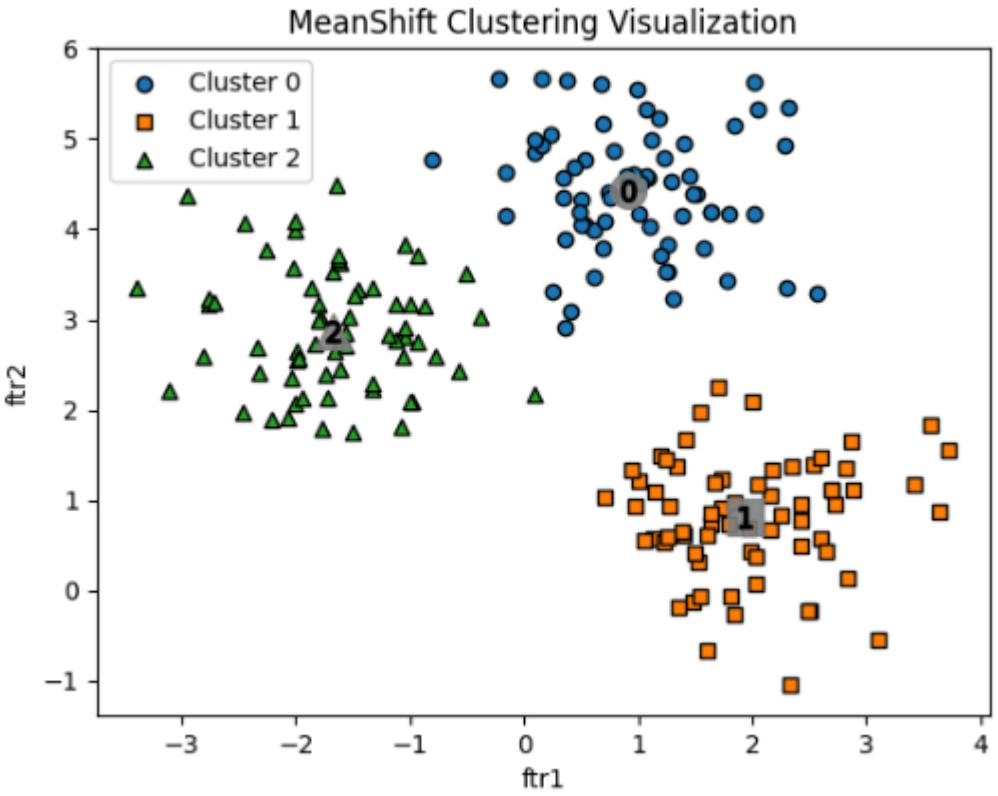
```
import pandas as pd
from sklearn.cluster import MeanShift, estimate_bandwidth

# DataFrame 생성
clusterDF = pd.DataFrame(data=X, columns=['ftr1', 'ftr2'])
clusterDF['target'] = y

# 최적 bandwidth 계산
best_bandwidth = estimate_bandwidth(X)

# MeanShift 모델 생성
meanshift = MeanShift(bandwidth=best_bandwidth)

# 군집 예측
cluster_labels = meanshift.fit_predict(X)

print("cluster labels 유형:", np.unique(cluster_labels))
```

→ 3개의 군집으로 구성

→ 시각화: K-평균과 유사하게 중심을 가지고 있으므로 cluster_centers_ 속성으로 군집 중심 좌표를 표시

```
import matplotlib.pyplot as plt
%matplotlib inline

# cluster label 저장
clusterDF['meanshift_label'] = cluster_labels

# 클러스터 중심
centers = meanshift.cluster_centers_
unique_labels = np.unique(cluster_labels)

# 마커 리스트
markers = ['o', 's', '^', 'x', '*', 'D']

# 군집별 시각화
for label in unique_labels:
```

```
    label_cluster = clusterDF[ clusterDF['meanshift_label'] == label ]
    center_x_y = centers[label]

    # 군집별 점 산점도
    plt.scatter(
        x=label_cluster['ftr1'],
        y=label_cluster['ftr2'],
        edgecolor='k',
        marker=markers[label],
        label=f"Cluster {label}"
    )

    # 군집 중심점 (큰 점)
    plt.scatter(
        x=center_x_y[0],
        y=center_x_y[1],
        s=200,
        color='gray',
        alpha=0.9,
        marker=markers[label]
    )

    # 중심에 라벨 숫자 표시
    plt.scatter(
        x=center_x_y[0],
        y=center_x_y[1],
        s=70,
        color='k',
        edgecolor='k',
        marker='$%d$' % label
    )

plt.title("MeanShift Clustering Visualization")
plt.xlabel("ftr1")
plt.ylabel("ftr2")
plt.legend()
plt.show()
```




```
1 print(clusterDF.groupby('target')['meanshift_label'].value_counts())
2
```

```
target  meanshift_label
0       0                 67
1       1                 67
2       2                 66
Name: count, dtype: int64
```

- Target 값과 군집 label 값이 1 ：1로 잘 매칭

장점

- **분포 가정이 필요 없어** 매우 유연함
- **이상치 영향이 적음**
- **군집 수(k)를 미리 지정할 필요 없음**

단점

- **계산 비용이 크고 느림**
- **bandwidth 값에 군집 결과가 크게 좌우됨**

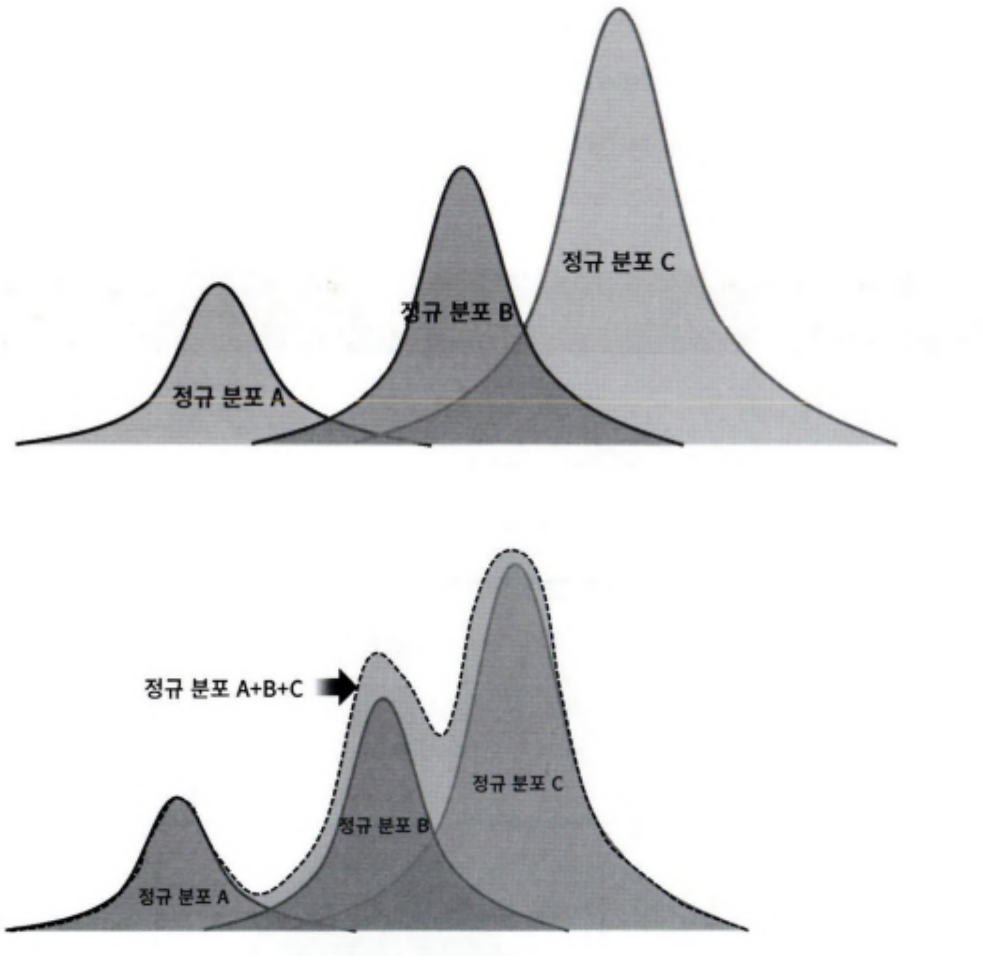
활용 분야

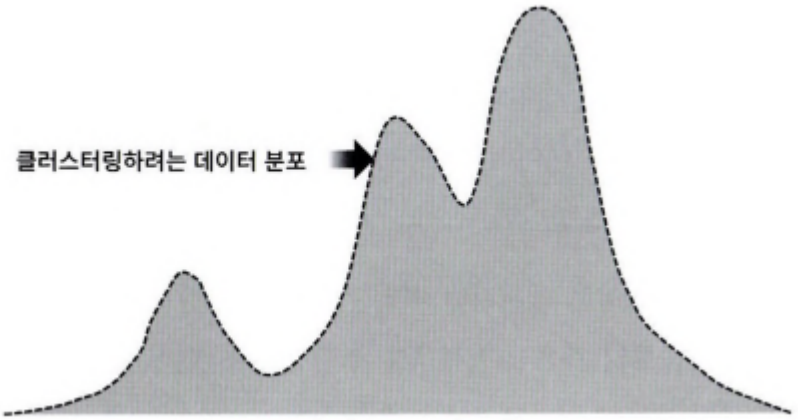
- 보통 일반 데이터 분석보다 **컴퓨터 비전(영상/이미지) 영역**에서 더 자주 사용

4. GMM(Gaussian Mixture Model) 455p

GMM 군집화는 군집화를 적용하고자 하는 데이터가 여러 개의 가우시안 분포 가진 데이터 집합들이 섞여서 생성된 것이라 가정

- 데이터를 여러 개의 가우시안 분포가 섞인 것
- 개별 유형의 가우시안 분포를 추출





- 전체 데이터 세트는 서로 다른 정규 분포 형태를 가진 여러 가지 확률 분포 곡선으로 구성
- 서로 다른 정규 분포에 기반해 군집화을 수행
- 여러 개의 정규 분포 곡선을 추출하고, 개별 데이터가 이 중 어떤 정규 분포에 속하는지

이는 모수 추정이라고 함 → EM(Expectation and Maximization) 방법을 적용

- 개별 정규 분포의 평균과 분산
- 각 데이터가 어떤 정규 분포에 해당하는지의 확률

사이킷런 `GaussianMixture 클래스`

붓꽃 데이터 세트 군집화

GMM은 확률 기반 군집화이고 K-평균은 거리 기반 군집화 → 비교

- 가장중요한초기화파라미터는 n_component: 모델의 총 개수

```
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
%matplotlib inline

iris = load_iris()
feature_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
```

```
irisDF = pd.DataFrame(data=iris.data, columns=feature_names)
irisDF['target'] = iris.target
```

```
from sklearn.mixture import GaussianMixture
import pandas as pd

gmm = GaussianMixture(n_components=3, random_state=0).fit(iris.data)

# 군집 라벨 예측
gmm_cluster_labels = gmm.predict(iris.data)

# DF 컬럼에 결과 저장
irisDF['gmm_cluster'] = gmm_cluster_labels
irisDF['target'] = iris.target

# target별 gmm_cluster 분포 확인
iris_result = irisDF.groupby('target')['gmm_cluster'].value_counts()

print(iris_result)
```

```
target  gmm_cluster
0       1              50
1       0              45
2               5
2       2              50
Name: count, dtype: int64
```

→ K-평균 군집화 결과보다 더 효과적인 분류 결과

```
from sklearn.cluster import KMeans

kmeans = KMeans(
    n_clusters=3,
    init='k-means++',
    max_iter=300,
    random_state=0
```

```
).fit(iris.data)

kmeans_cluster_labels = kmeans.predict(iris.data)

irisDF['kmeans_cluster'] = kmeans_cluster_labels

iris_result = irisDF.groupby('target')['kmeans_cluster'].value_counts()

print(iris_result)
```

```
target  kmeans_cluster
0       1                50
1       0                47
2                3
2       2                36
0               14
Name: count, dtype: int64
```

** 어떤 알고리즘에 더 뛰어나다는 의미가 아니라 붓꽃 데이터 세트가 GMM 군집화에 더 효과적이라는 의미

- K-평균은 데이터가 원형으로 흩어져 있는 경우에 매우 효과적으로 군집화가 수행

GMM과 K—평균의 비교

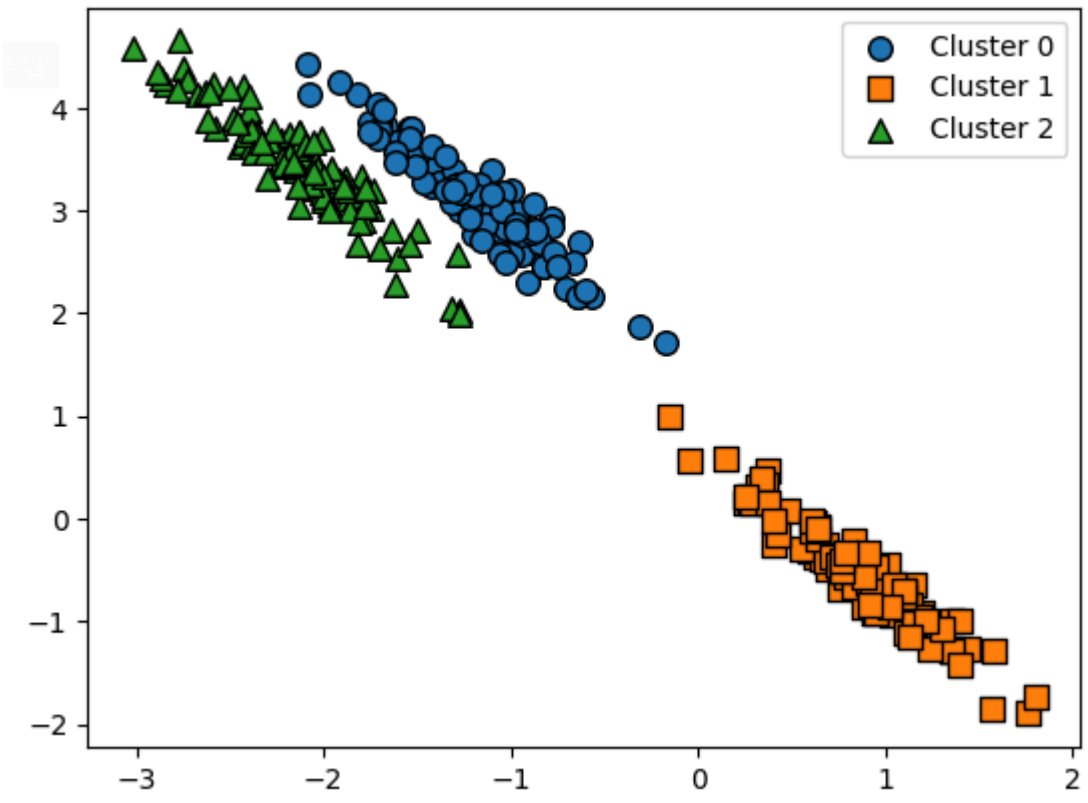
- KMeans는 원형의 범위에서 군집화를 수행
- cluster_std를 작게 설정하면 데이터가 원형 형태로 분산
- 데이터가 길쭉한 타원에서는 성능 저하

```
def visualize_cluster_plot(clusterobj, dataframe, label_name, iscenter=Tru
e):

    # 군집별 중심 위치: K-Means, Mean Shift 등
    if iscenter:
        centers = clusterobj.cluster_centers_

    # Cluster 값 종류
    unique_labels = np.unique(dataframe[label_name].values)
```

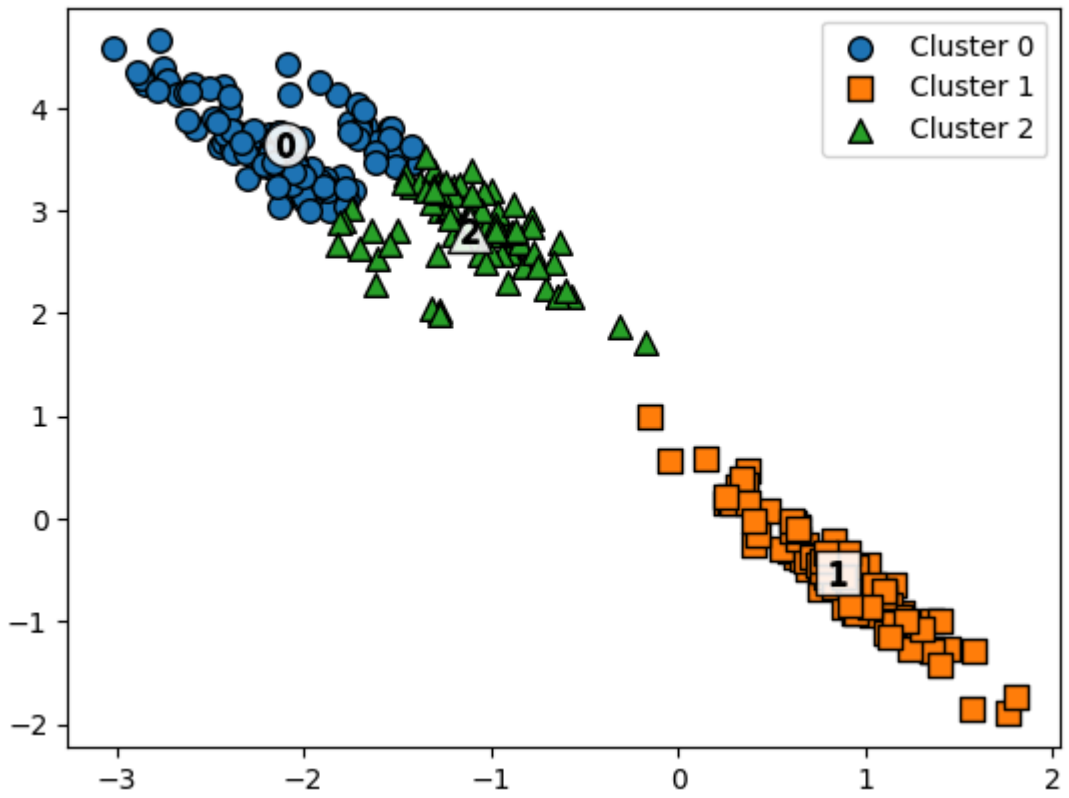
```
    markers=['o', 's', '^', 'x', '*']
    isNoise=False

    for label in unique_labels:
        # 군집별 데이터 프레임
        label_cluster = dataframe[dataframe[label_name]==label]

        if label == -1:
            cluster_legend = 'Noise'
            isNoise=True
        else:
            cluster_legend = 'Cluster '+str(label)

        # 각 군집 시각화
        plt.scatter(x=label_cluster['ftr1'], y=label_cluster['ftr2'], s=70,
                    edgecolor='k', marker=markers[label], label=cluster_legend)

        # 군집별 중심 위치 시각화
        if iscenter:
            center_x_y = centers[label]
            plt.scatter(x=center_x_y[0], y=center_x_y[1], s=250, color='white',
                        alpha=0.9, edgecolor='k', marker=markers[label])
            plt.scatter(x=center_x_y[0], y=center_x_y[1], s=70, color='k',\
                        edgecolor='k', marker='$%d$' % label)

    if isNoise:
        legend_loc='upper center'
    else:
        legend_loc='upper right'

    plt.legend(loc=legend_loc)
    plt.show()
```

```
from sklearn.datasets import make_blobs

# make_blobs로 300개 데이터, 3개 군집 생성
X, y = make_blobs(
    n_samples=300,
```

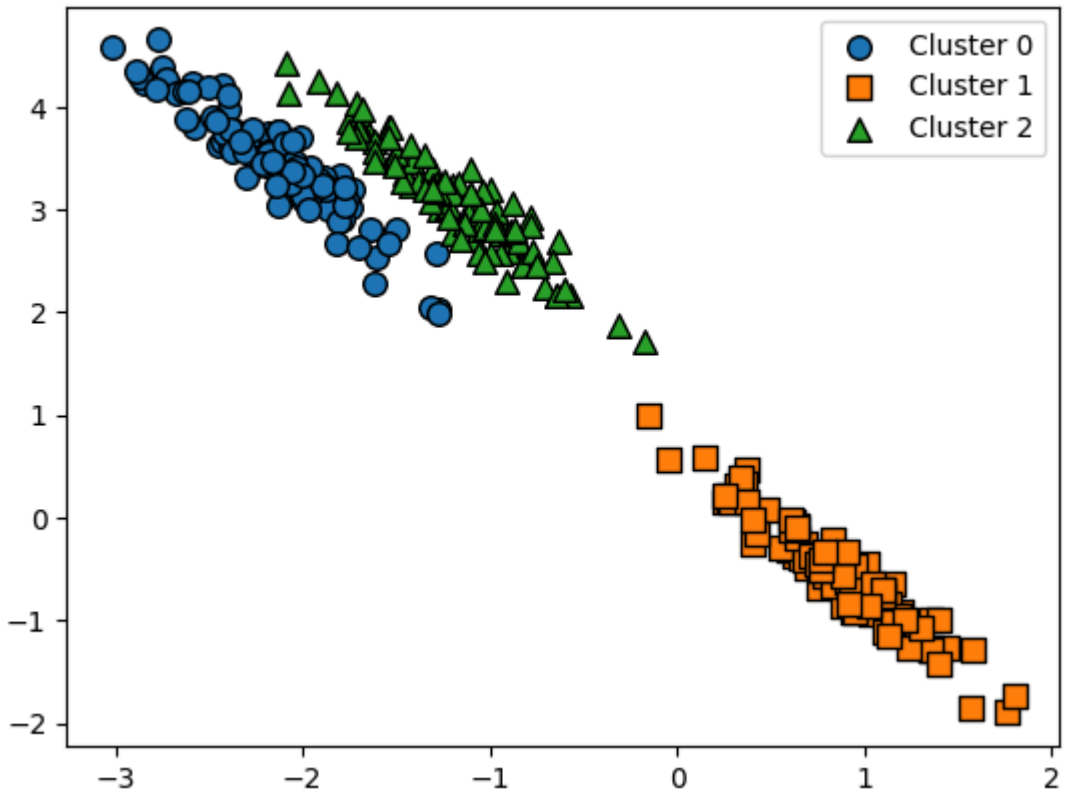
```
    n_features=2,
    centers=3,
    cluster_std=0.5,
    random_state=0
)

transformation = np.array([
    [0.60834549, -0.63667341],
    [-0.40887718,  0.85253229]
])

X_aniso = np.dot(X, transformation)

clusterDF = pd.DataFrame(data=X_aniso, columns=['ftr1', 'ftr2'])
clusterDF['target'] = y

# target별로 다른 마커로 시각화
visualize_cluster_plot(None, clusterDF, 'target', iscenter=False)
```



```
# 3개의 군집 기반 KMeans를 X_aniso 데이터 세트에 적용
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans_label = kmeans.fit_predict(X_aniso)
clusterDF['kmeans_label'] = kmeans_label

visualize_cluster_plot(kmeans, clusterDF, 'kmeans_label', iscenter=True)
```




```
from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n_components=3, random_state=0)
gmm_label = gmm.fit(X_aniso).predict(X_aniso)

clusterDF['gmm_label'] = gmm_label

# GaussianMixture는 cluster_centers_ 속성이 없으므로 iscenter=False
visualize_cluster_plot(gmm, clusterDF, 'gmm_label', iscenter=False)
```



데이터가 분포된 방향에 따라 정확하게 군집화

```
1 print("### KMeans Clustering ###")
2 print(clusterDF.groupby('target')['kmeans_label'].value_counts())
3
4 print("₩n### Gaussian Mixture Clustering ###")
5 print(clusterDF.groupby('target')['gmm_label'].value_counts())
6
```

```
### KMeans Clustering ###
target  kmeans_label
0       2               73
        0               27
1       1              100
2       0               86
        2               14
Name: count, dtype: int64

### Gaussian Mixture Clustering ###
target  gmm_label
0       2             100
1       1             100
2       0             100
Name: count, dtype: int64
```

KMeans의 경우 군집 1 번만 정확히 매핑

GMM의 경우는 군집이 target 값과 잘 매핑

→ GMM의 경우는 KMeans보다 유연하게 다양한 데이터 세트에 잘 적용됨

DBSCAN 463p

DBSCAN은 **밀도 기반 군집화 알고리즘**으로,

데이터가 복잡한 형태(예: 내부 원 + 외부 원 같은 기하학적 구조)일 때도 군집화가 잘 된다.

- K-means, Mean Shift, GMM처럼 **모양이 단순한 군집**만 잘 잡는 알고리즘들과 달리
- DBSCAN은 **데이터 밀도 차이**를 이용해 군집을 찾기 때문에
  비선형·복잡한 형태의 분포에서도 효과적으로 군집화할 수 있다.

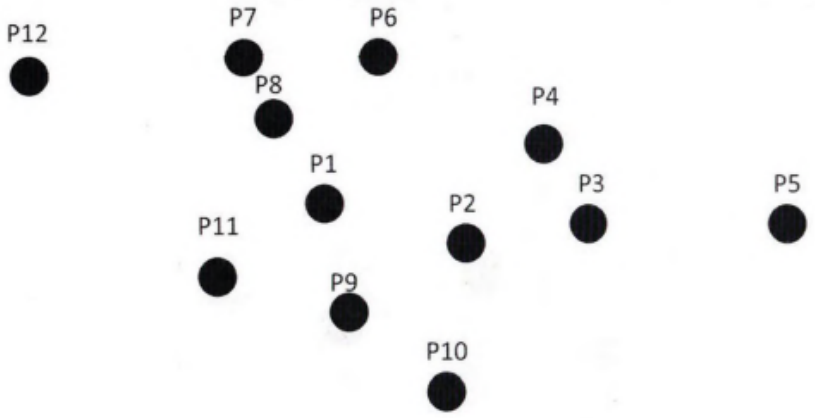
또한 노이즈(이상치)를 자연스럽게 분리해내는 장점도 있다.

DBSCAN은 두 파라미터로 군집을 만든다:

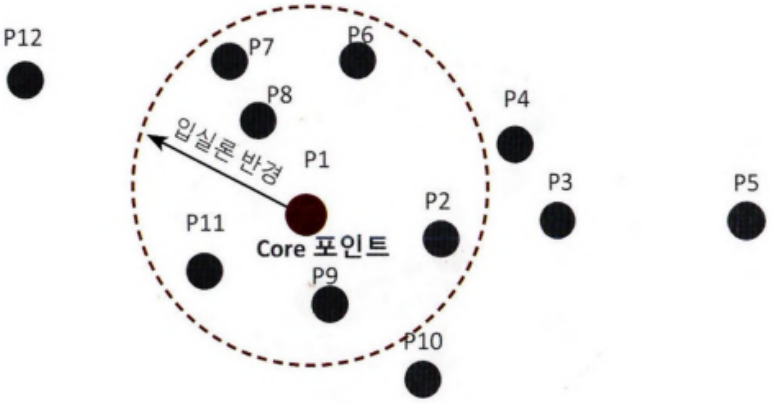
- **epsilon(ε)**: 각 점을 중심으로 한 반경(이웃 범위)
- **min_points**: 그 반경 안에 있어야 하는 최소 이웃 수

이 두 조건으로 데이터 포인트를 다음처럼 분류한다:

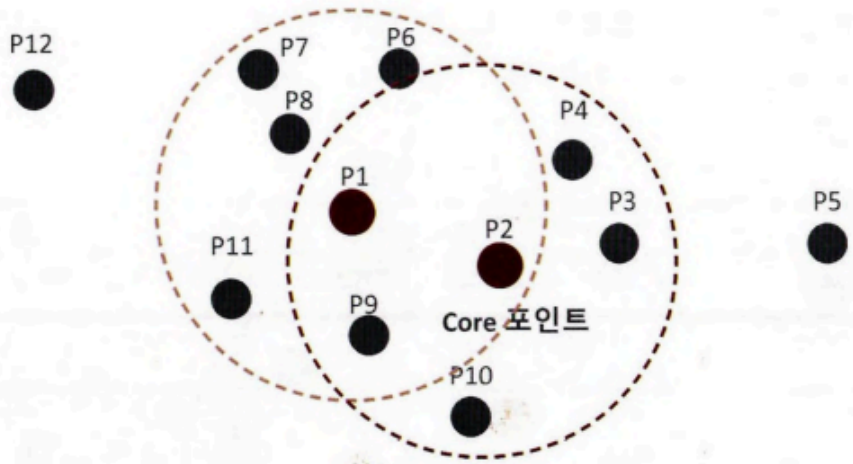
- **핵심 포인트(Core Point)**:
  ε 안에 이웃이 min_points 이상 있는 점
- **경계 포인트(Border Point)**:
  자체는 min_points를 못 채우지만, 핵심 포인트의 이웃인 점
- **잡음 포인트(Noise)**:
  주변에 이웃도 적고, 핵심 포인트의 이웃도 아닌 점



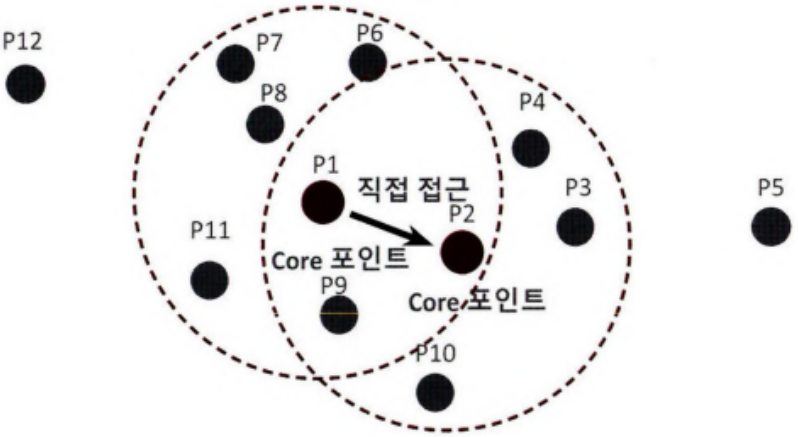
1. 특정 입실론 반경 내에 포함될 최소 데이터 세트를 6개로(자기 자신의 데이터를 포함) 가정



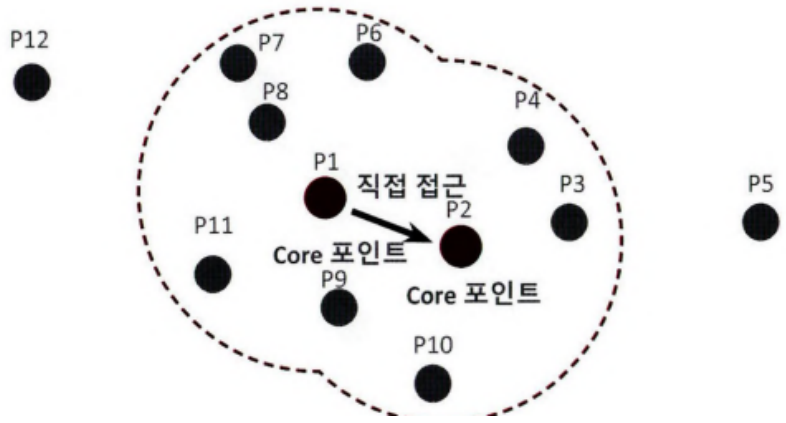
2. P1 데이터를 기준으로 입실론 반경 내에 포함된 데이터가 7개（자신은 P1, 이웃 데이터 P2, P6, P7, P8, P9, P11）로 최소 데이터 5개 이상을 만족하므로 P1 데이터는 핵심 포인트（Core Point）



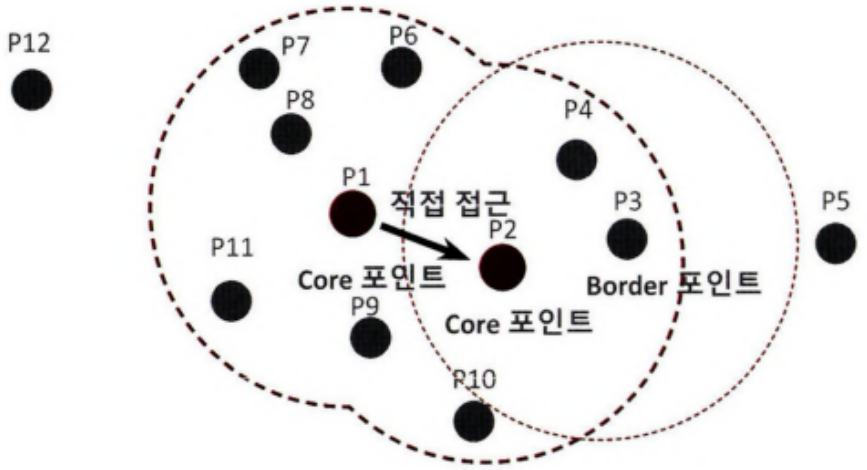
3. P2 역시 반경 내에 6개의 데이터 (자신은 P2, 이웃 데이터 P1, P3, P4, P9, P10) 를 가지고 있0므루 핵심 포인트



4. P2 역시 핵심 포인트일 경우 P1 에서 P2로 연결해 직접 접근이 가능



5. 특정 핵심 포인트에서 직접 접근이 가능한 다른 핵심 포인트를 서로 연결하면서 군집화



6. P3 데이터의 경우 반경 내에 포함되는 이웃 데이터는 P2, P4로 2개이므로 군집으로 구분할 수 있는 핵심 포인트가 될 수 없음 → 경계 포인트 & P5는 반경 내에 최소 데이터도 핵심 포인트도 없음 → 잡음 포인트

DBSCAN 클래스

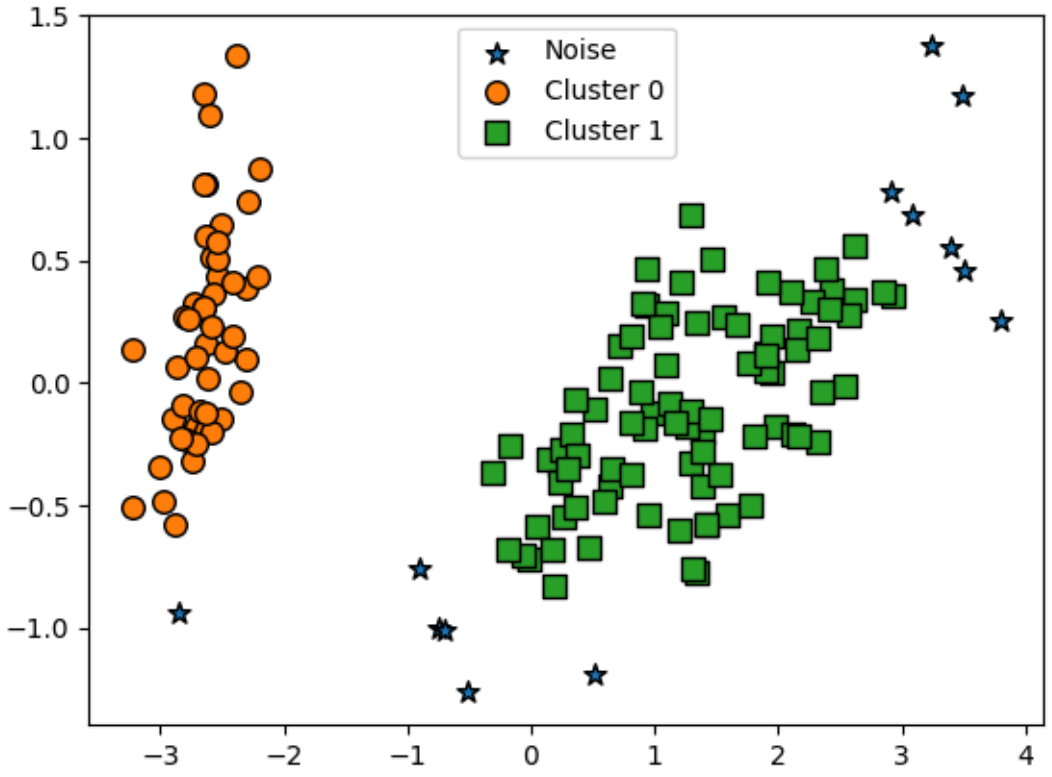
- eps: 입실론 주변 영역의 반경
- min_sampels: 핵심 포인트가 되기 위한 최소 데이터 수(자신을 포함)

붓꽃 데이터 세트

```
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=0.6, min_samples=8, metric='euclidean')
dbscan_labels = dbscan.fit_predict(iris.data)
irisDF['dbscan_cluster'] = dbscan_labels
irisDF['target'] = iris.target

iris_result = irisDF.groupby('target')['dbscan_cluster'].value_counts()

print(iris_result)
```

```
target  dbscan_cluster
0        0               49
        -1                1
1        1               46
        -1                4
2        1               42
        -1                8
Name: count, dtype: int64
```

- 군집 레이블이 -L인 것은 노이즈에 속하는 군집
- DBSCAN에서 0과 1 두 개의 군집으로 군집화
- DBSCAN에서 군집의 개수를 지정하는 것은 무의미 → 자동 지정하기에
- PCA를 이용해 2개의 피처로 압 축 변환한 뒤, 앞 예제에서 사용한 visualize_cluster_plot( ) 함수를 이용해 시각화

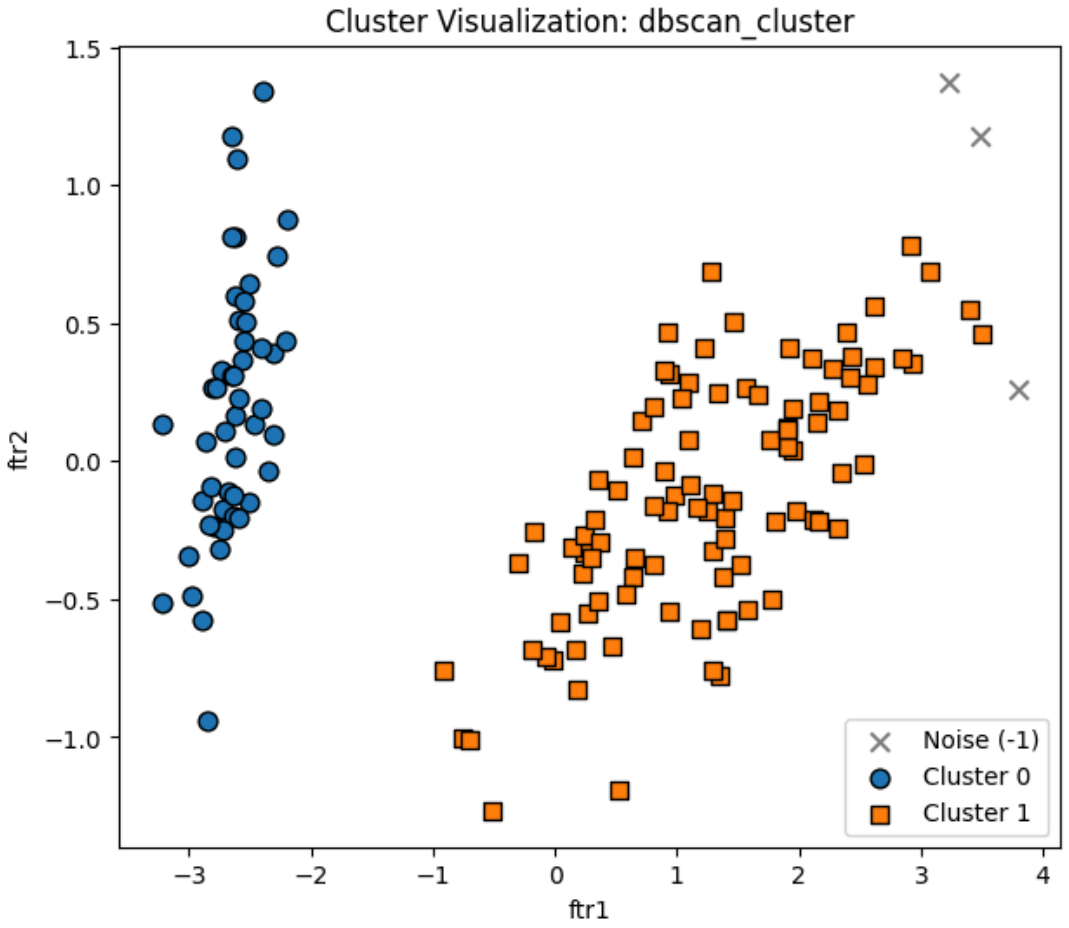


PCA로 2차원으로 표현하면 이상치인 노이즈 데이터가 명 확히 드러남

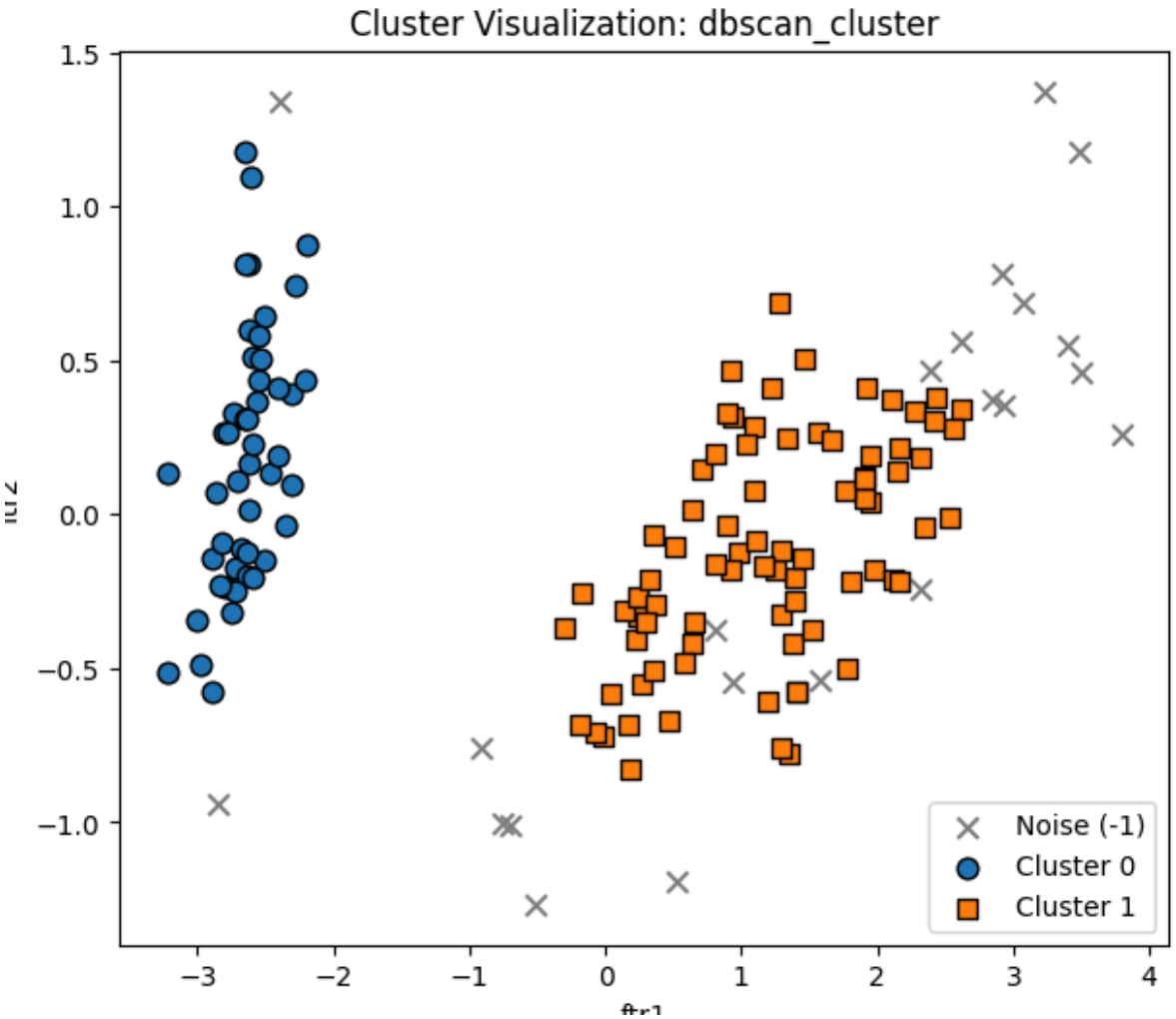
- DBSCAN 알고리즘에 적절한 eps와 min_samples 파라미터를 통해 최적의 군집 찾는 것이 중요
- 일반적으로 eps의 값을 크게 → 반경이 커져 포함하는 데이터가 많 → 노이즈 데이터 개수 작아짐

- min_samples를 크게 → 반경 내에서 더 많은 데이터를 포함 → 노이즈 데이터 개수가 커짐

esp 0.6 → 0.8



esp 0.6 유지, min_samples 16으로



DBSCAN 적용하기 - make_circles() 데이터 세트

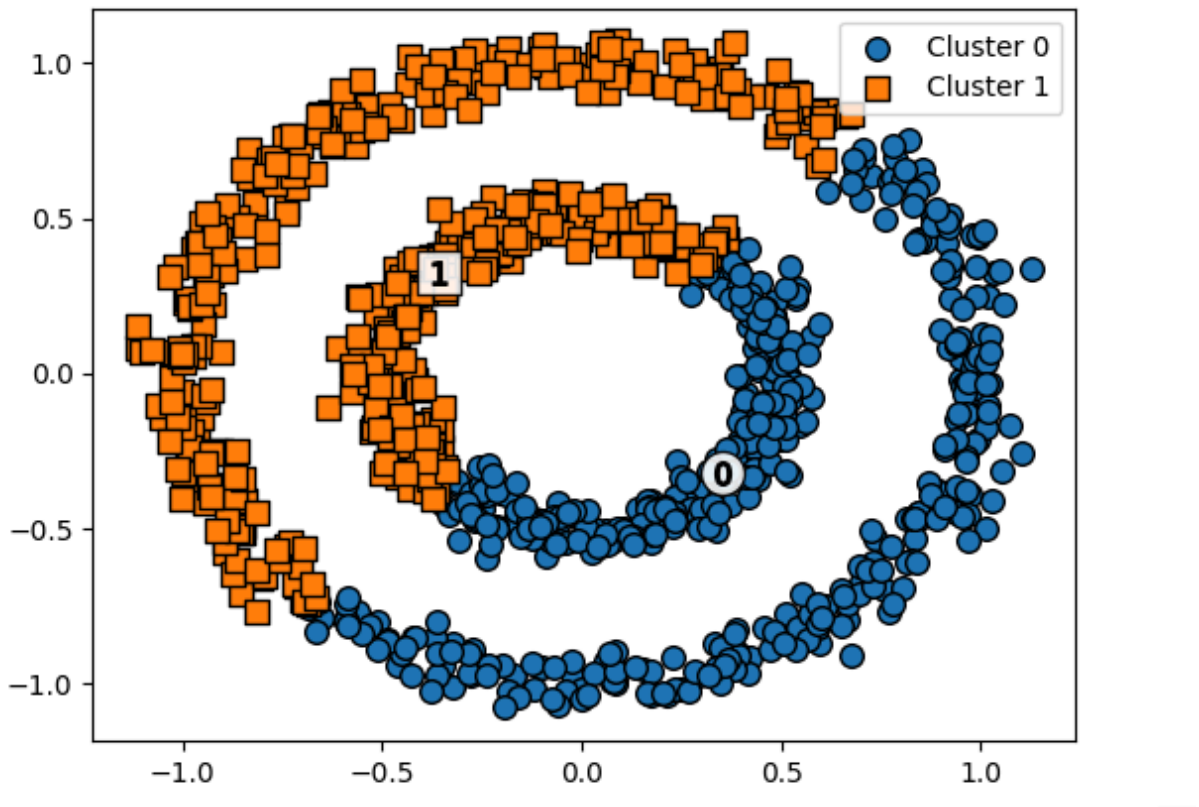
make_circles() 함수

- 내부 원과 외부 원 형태로 돼 있는 2차원 데이터 세트
- 오직 2개의 피처만을 생성

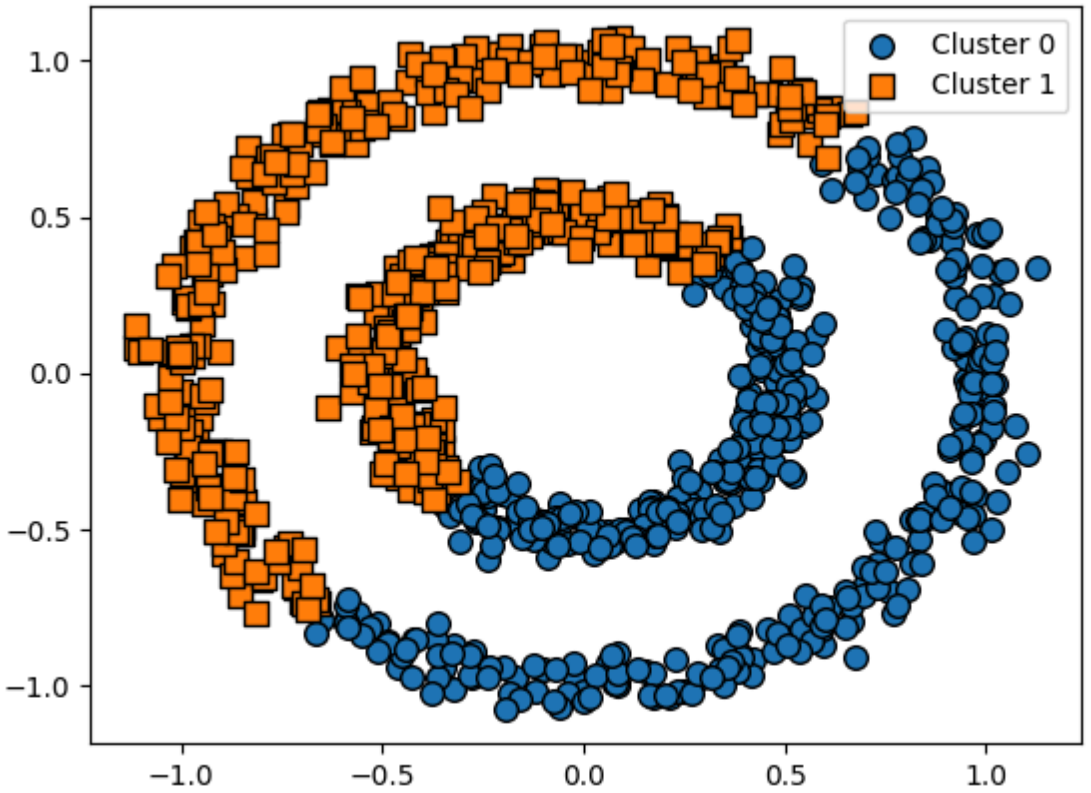
```
from sklearn.datasets import make_circles
import pandas as pd

X, y = make_circles(
    n_samples=1000,
    shuffle=True,
    noise=0.05,
    random_state=0,
    factor=0.5
)
```

```
clusterDF = pd.DataFrame(data=X, columns=['ftr1', 'ftr2'])
clusterDF['target'] = y
visualize_cluster_plot(None, clusterDF, 'target', iscenter=False)
```

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2, max_iter=1000, random_state=0)
kmeans_labels = kmeans.fit_predict(X)

clusterDF['kmeans_cluster'] = kmeans_labels

visualize_cluster_plot(kmeans, clusterDF, 'kmeans_cluster', iscenter=True)
```

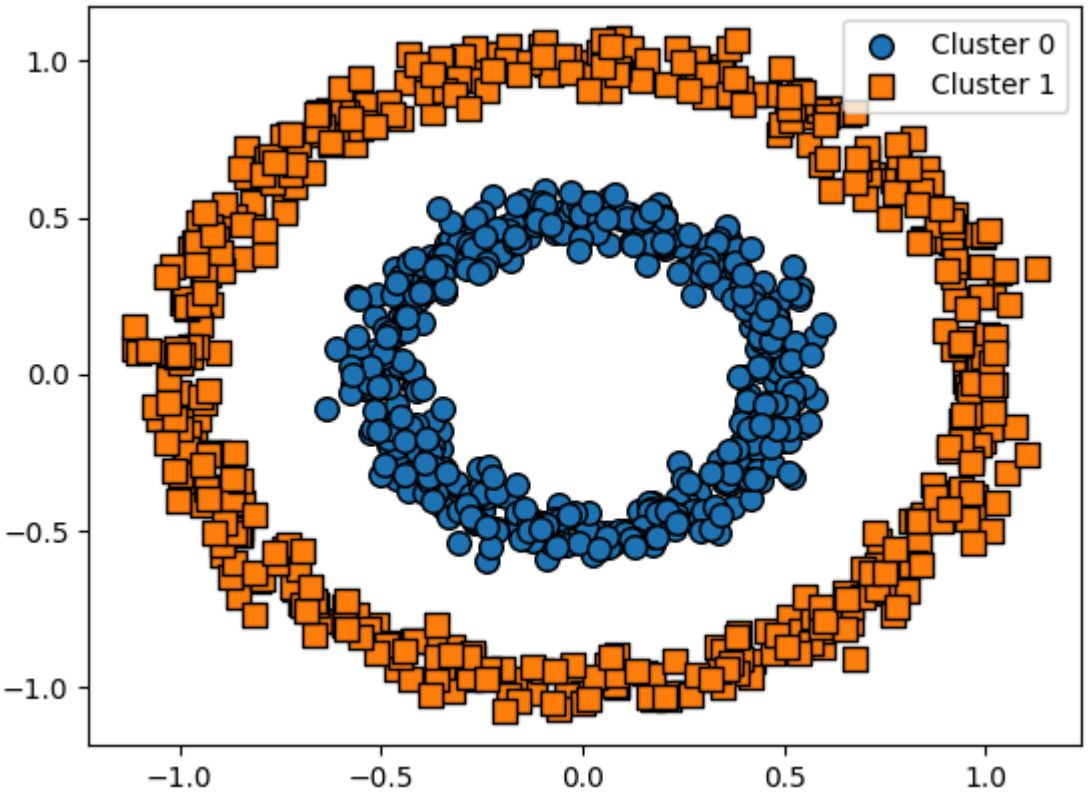


K-평균⇒ 위와 아래 절반으로 군집화



GMM ⇒ 일렬과 다르게 원형 복잡한 데이터 안됨

```
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=0.2, min_samples=10, metric='euclidean')
dbscan_labels = dbscan.fit_predict(X)

clusterDF['dbscan_cluster'] = dbscan_labels

visualize_cluster_plot(dbscan, clusterDF, 'dbscan_cluster', iscenter=False)
```



DBSCAN은 효과적 군집화

군집화 실습 - 고객 세그먼테이션 474p

고객 세그먼테이션(Customer Segmentation)은 다양한 기준으로 고객을 분류하는 기법

1) InvoiceNo

주문 번호

`'C'` 로 시작하면 **취소(Cancelled)** 주문을 의미

2) StockCode

제품 코드 (Item Code)

하나의 인보이스 안에 여러 StockCode가 있을 수 있음

3) Description

제품 설명 (제품명)

4) Quantity

주문한 제품의 수량

음수(-)일 경우 취소/반품을 의미할 수 있음

5) InvoiceDate

주문 발생 시각 (날짜 + 시간)

6) UnitPrice

제품 단가

7) CustomerID

- 고객 ID
- NA/Null이면 고객 정보가 없는 주문

8) Country

- 주문 고객의 국가명

info()

0 InvoiceNo 541909 non-null object

1 StockCode 541909 non-null object

2 Description 540455 non-null object

3 Quantity 541909 non-null int64

4 InvoiceDate 541909 non-null datetime64[ns]
5 UnitPrice 541909 non-null float64

6 CustomerID 406829 non-null float64

7 Country 541909 non-null object

dtypes: datetime64ns, float64(2), int64(1), object(4)
memory usage: 33.1+ MB

→ CustomerID의 Null 값이 너무 많음 & 오류 데이터

사전 정제: 불린 인덱싱을 적용해 **Not Null인 값만필터링 & 다수인 영국 데이터만 남기기**

```
retail_df = retail_df[retail_df['Quantity'] > 0]
retail_df = retail_df[retail_df['UnitPrice'] > 0]
retail_df = retail_df[retail_df['CustomerID'].notnull()]

print(retail_df.shape)
retail_df.isnull().sum()

```

```
(397884, 8)
```

	0
InvoiceNo	0
StockCode	0
Description	0
Quantity	0
InvoiceDate	0
UnitPrice	0
CustomerID	0
Country	0

```
retail_df = retail_df[retail_df['Country'] == 'United Kingdom']
print(retail_df.shape)
```

RFM 기반 데이터 가공

1. UnitPrice'x 'Quantity' → 주문 금액 데이터
2. CustomerNo도 더 편리한 식별성을 위해 float 형을 int 형으로

```
retail_df['sale_amount'] = retail_df['Quantity'] * retail_df['UnitPrice']
retail_df['CustomerID'] = retail_df['CustomerID'].astype(int)
```

```
print(retail_df['CustomerID'].value_counts().head(5))

print(
    retail_df.groupby('CustomerID')['sale_amount']
    .sum()
    .sort_values(ascending=False)[:5]
)

```

```
CustomerID
17841    7847
14096    5111
12748    4595
14606    2700
15311    2379
Name: count, dtype: int64
CustomerID
18102    259657.30
17450    194550.79
16446    168472.50
17511     91062.38
16029     81024.84
Name: sale_amount, dtype: float64
```

- 몇몇 특정 고객이 많은 주문 건수와 주문 금액
- InvoiceNo + StockCode로 Group by를 수행하면 거 의 1 에 가깝게 유일한 식별자 레벨이 됨

3, RFM 기반 고객 세그먼테이션은 **고객 단위**로 Recency, Frequency, Monetary 값을 계산해야 함

- 따라서 주문(Invoice) 단위 데이터인 `retail_df`를 **CustomerID 기준으로 groupby** 해서 고객별 요약 정보로 변환

1) groupby(CustomerID) 후 agg() 사용

칼럼마다 다른 집계를 한 번에 수행하기 위해

`groupby().agg({컬럼명: 집계함수})` 형태를 사용

2) Frequency (F)

고객별 주문 횟수 `InvoiceNo`의 **count()**

3) Monetary Value (M)

고객별 총 구매 금액 `sale_amount`의 **sum()**

4) Recency (R)

고객별 가장 최근 구매일 `InvoiceDate`의 **max()** 값으로 구한 뒤 기준 날짜(today)와의 차이로 Recency 계산

```
aggregations = {
    'InvoiceDate': 'max',
    'InvoiceNo': 'count',
    'sale_amount': 'sum'
}

cust_df = retail_df.groupby('CustomerID').agg(aggregations)

cust_df = cust_df.rename(columns={
    'InvoiceDate': 'Recency',
    'InvoiceNo': 'Frequency',
    'sale_amount': 'Monetary'
})

cust_df = cust_df.reset_index()
cust_df.head(3)
```

	CustomerID	Recency	Frequency	Monetary
0	12346	2011-01-18 10:01:00	1	77183.60
1	12747	2011-12-07 14:34:00	103	4196.01
2	12748	2011-12-09 12:20:00	4595	33719.73

```
import datetime as dt

cust_df['Recency'] = dt.datetime(2011, 12, 10) - cust_df['Recency']
cust_df['Recency'] = cust_df['Recency'].apply(lambda x: x.days + 1)
```

```
print("cust_df 로우와 칼럼 건수는", cust_df.shape)
cust_df.head(3)
```

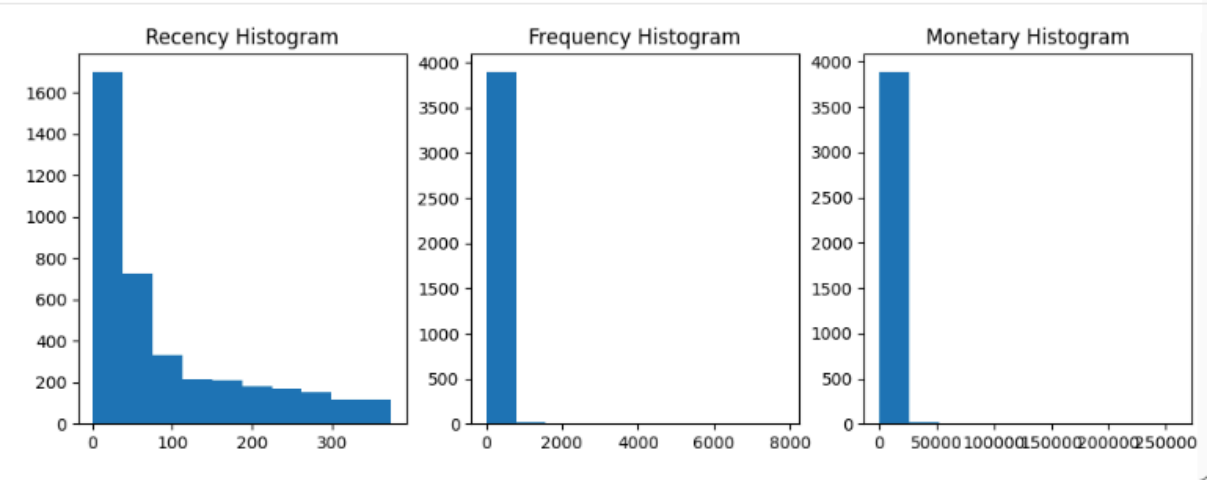
```
cust_df 로우와 칼럼 건수는 (3920, 4)
```

	CustomerID	Recency	Frequency	Monetary
0	12346	326	1	77183.60
1	12747	3	103	4196.01
2	12748	1	4595	33719.73

4. 오늘 날짜를 현재 날짜로 해서는 안 됨 → 2011년 12월 9일에서 하루 더한 2011년 12월 10일

RFM 기반 고객 세그먼테이션

```
import matplotlib.pyplot as plt

fig, (ax1, ax2, ax3) = plt.subplots(figsize=(12, 4), nrows=1, ncols=3)

ax1.set_title('Recency Histogram')
ax1.hist(cust_df['Recency'])

ax2.set_title('Frequency Histogram')
ax2.hist(cust_df['Frequency'])

ax3.set_title('Monetary Histogram')
ax3.hist(cust_df['Monetary'])

plt.show()
```



소매업체 대규모 주문: 개인 고객 주문과 큰 주문 횟수, 주문 금액 차이

→ 왜곡된 데이터 분포도 → 한쪽 군집에만 집중

```
cust_df[['Recency', 'Frequency', 'Monetary']].describe()
```

	Recency	Frequency	Monetary
count	3920.000000	3920.000000	3920.000000
mean	92.742092	90.388010	1864.385601
std	99.533485	217.808385	7482.817477
min	1.000000	1.000000	3.750000
25%	18.000000	17.000000	300.280000
50%	51.000000	41.000000	652.280000
75%	143.000000	99.250000	1576.585000
max	374.000000	7847.000000	259657.300000

- Recency는 평균이 92.7이지만, 50%（중위값 2/4 분위）인 51보다 크게 높음
- max 값 은 374로 75%（3/4 분위）인 143보다 훨씬 커서 왜곡 정도가 높음
- Frequency와 Monetary의 경우는 왜곡 정도가 더 심해서 Frequency의 평균이 90.3인데, 75%인 99.25에 가깝

K-평균 + StandardScaler로 평균과 표준편차를 재조정

```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
```

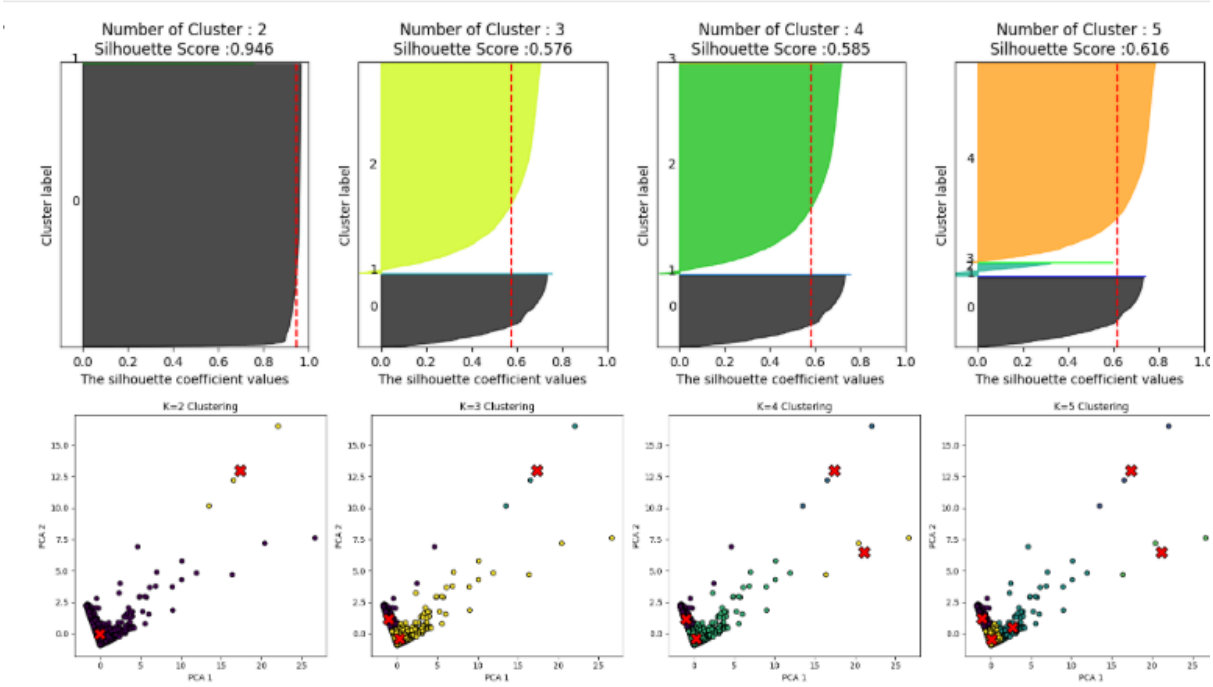
```
from sklearn.metrics import silhouette_score, silhouette_samples

X_features = cust_df[['Recency', 'Frequency', 'Monetary']].values
X_features_scaled = StandardScaler().fit_transform(X_features)

kmeans = KMeans(n_clusters=3, random_state=0)
labels = kmeans.fit_predict(X_features_scaled)

cust_df['cluster_label'] = labels

print('실루엣 스코어는 : {0:.3f}'.format(silhouette_score(X_features_scaled, la
bels)))
```

실루엣 스코어는 : 0.576

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import numpy as np

def visualize_kmeans_plot_multi(cluster_list, X):
    """
    cluster_list : 사용할 K 개수 리스트 (예: [2,3,4,5])
    X : StandardScaler로 스케일링된 feature 데이터
    """

    # PCA로 2차원 변환
    pca = PCA(n_components=2)
    X_pca = pca.fit_transform(X)

    # subplot 크기 설정
    n_cols = len(cluster_list)
    fig, axs = plt.subplots(1, n_cols, figsize=(5 * n_cols, 5))

    if len(cluster_list) == 1:
        axs = [axs]

    for idx, k in enumerate(cluster_list):
```

```
        ax = axs[idx]

        # KMeans 군집화
        kmeans = KMeans(n_clusters=k, random_state=0)
        labels = kmeans.fit_predict(X)

        # 시각화
        ax.scatter(
            X_pca[:, 0],
            X_pca[:, 1],
            c=labels,
            cmap='viridis',
            s=30,
            edgecolor='k'
        )

        # 중심 표시
        centers_pca = pca.transform(kmeans.cluster_centers_)
        ax.scatter(
            centers_pca[:, 0],
            centers_pca[:, 1],
            c='red',
            s=200,
            marker='X',
            edgecolor='k'
        )

        ax.set_title(f"K={k} Clustering")
        ax.set_xlabel("PCA 1")
        ax.set_ylabel("PCA 2")

    plt.tight_layout()
    plt.show()
```

```
visualize_silhouette([2, 3, 4, 5], X_features_scaled)
visualize_kmeans_plot_multi([2, 3, 4, 5], X_features_scaled)
```



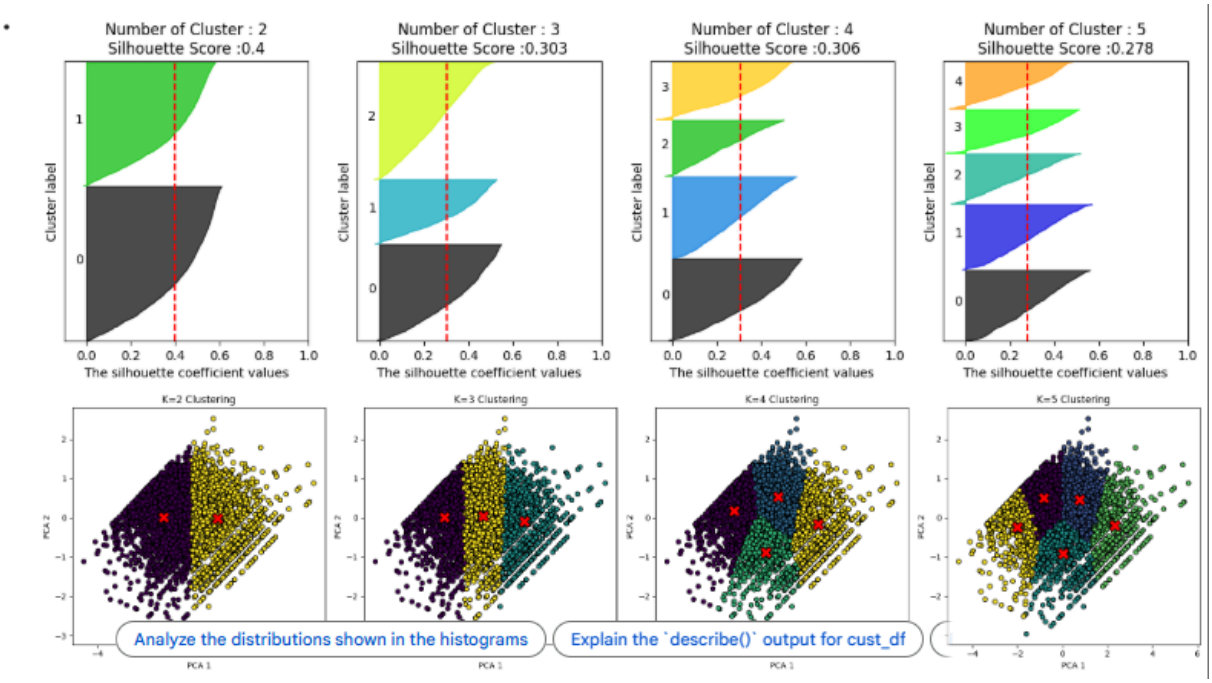
⇒ 왜곡된 극단값 때문에 KMeans는 의미 있는 고객 세그먼트를 만들지 못하며, 군집 수를 늘려도 소수·저품질 군집만 생겨 유효한 인사이트를 제공하지 못함

⇒ 데이터 값에 로그(Log)를 적용

```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import numpy as np

# Log Transformation
cust_df['Recency_log'] = np.log1p(cust_df['Recency'])
cust_df['Frequency_log'] = np.log1p(cust_df['Frequency'])
cust_df['Monetary_log'] = np.log1p(cust_df['Monetary'])

# Scaling
X_features = cust_df[['Recency_log', 'Frequency_log', 'Monetary_log']].valu
es
X_features_scaled = StandardScaler().fit_transform(X_features)

# KMeans
kmeans = KMeans(n_clusters=3, random_state=0)
```

```
labels = kmeans.fit_predict(X_features_scaled)

cust_df['cluster_label'] = labels

print('실루엣 스코어는 : {0:.3f}'.format(silhouette_score(X_features_scaled, labels)))
```

실루엣 스코어는 : 0.303



실루엣 스코어는 로그 변환하기 전보다 떨어지지만 앞의 경우보다 더 균일하게 군집화가 구성되었음