Elixirr

군집화

2025.10.24

김자영 강사

군집화

clustering

비지도학습 유사성 기반

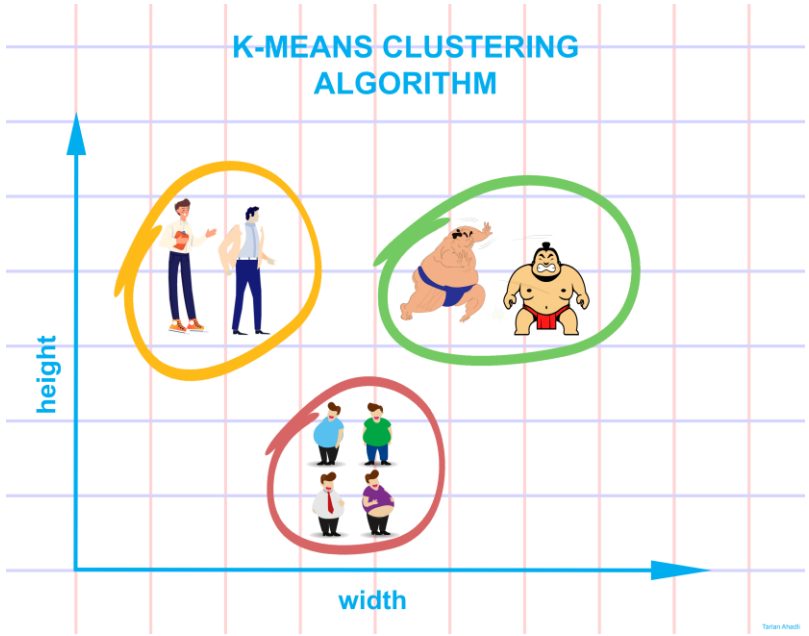
- **K-means** clustering 중심 기반 군집화
- **DBSCAN**(Density-Based Spatial Clustering of Application with Noise) 밀도 기반 군집화

- **Hierarchical** clustering 계층적 군집화

k-means clustering

■ 개념 및 특징

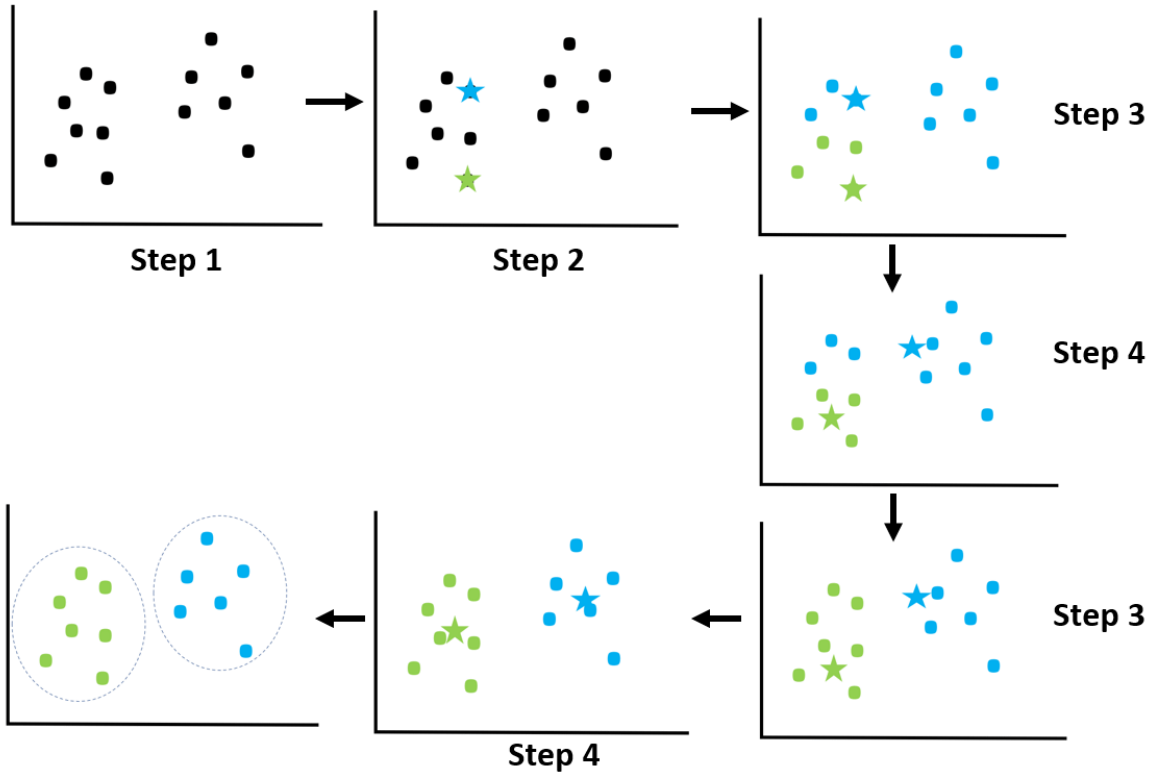
- **원형 클러스터**를 구분하는 데 적합하다.
- 작은 데이터셋에서 잘 작동한다.
- 구현하기 쉽고, 다른 군집 알고리즘에 비해 계산 효율성이 높다.
- **사전에 클러스터의 개수 k를 지정해야 한다.**
- 특성의 **스케일**이 필요하다.
- 평가방법
  - 엘보우 방법(elbow method)
  - 실루엣 그래프(silhouette plot)



https://www.lancaster.ac.uk/stor-i-student-sites/harini-jayaraman/k-means-clustering/

k-means clustering

■ 동작 원리



https://preview.redd.it/wodjl2e8ffw71.png?width=1121&format=png&auto=webp&s=1a723f05e981efee1519c4999c95c6611c587c0f

① Step1 : 군집 수(k값) 선택

② Step2 : 초기 군집 중심(Centroid) 설정

③ Step3 : 각 데이터 포인트를 가장 가까운 군집 중심에 할당

④ Step4 : 각 군집의 평균값 계산 후 군집 중심 이동

⑤ Step5 : 군집 중심이 변하지 않을 때까지 반복

k-means clustering

■ 데이터 준비

- 특성의 중요도

```
# 라이브러리 불러오기
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

k-means clustering

■ 데이터 준비

- 3개의 군집을 가지는 테스트데이터 생성

```
# 3개의 군집을 가지는 테스트 데이터 생성
from sklearn.datasets import make_blobs
features, targets = make_blobs(n_samples=200,
                    n_features=2,
                    centers=3,
                    cluster_std=0.8,
                    random_state=0)

print(features.shape, targets.shape)
```

```
(200, 2) (200,)
```

k-means clustering

■ 데이터 준비

- 데이터프레임으로 만들기

```
# 생성된 테스트데이터로 DataFrame 만들기
cluster_df = pd.DataFrame(features, columns=['feature1','feature2'])
cluster_df['target'] = targets
cluster_df.head(3)
```

	feature1	feature2	target
0	-1.692427	3.622025	2
1	0.697940	4.428867	0
2	1.100228	4.606317	0

k-means clustering

■ 데이터 준비

- 데이터프레임으로 만들기

```
# target 데이터의 빈도수
cluster_df['target'].value_counts()
```

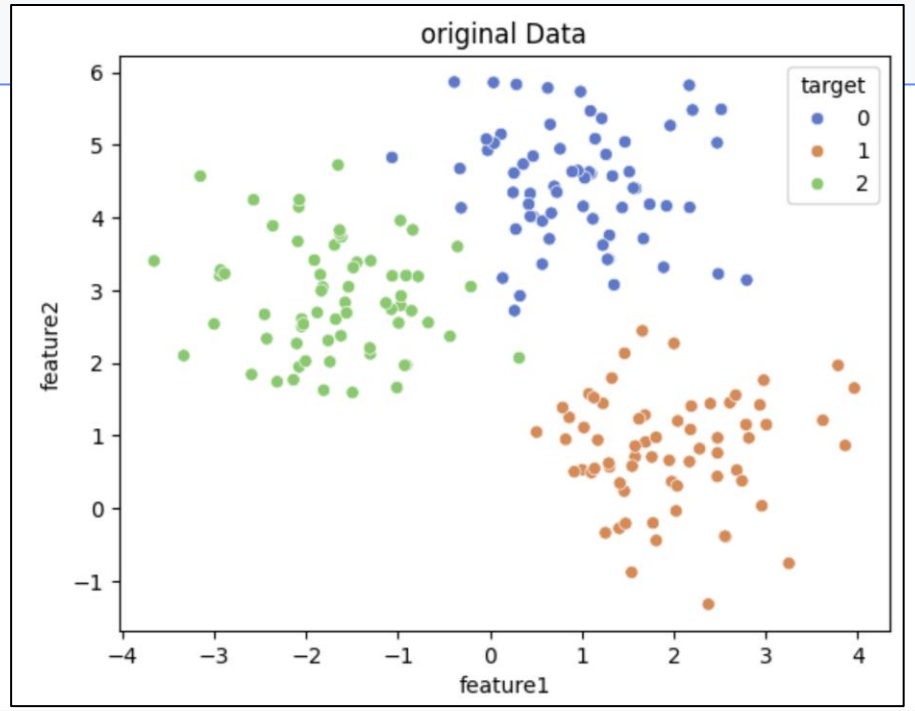
```
target
1    67
0    67
2    66
Name: count, dtype: int64
```

k-means clustering

■ 데이터 준비

- 데이터 시각화

```
# 데이터 시각화
sns.scatterplot(data=cluster_df, x='feature1', y='feature2', hue='target', palette='muted')
plt.title('original Data')
plt.show()
```



k-means clustering

■ 클러스터링

▪ 클러스터링

```
# 클러스터링 객체 생성
from sklearn.cluster import KMeans
km= KMeans(n_clusters=3)

# 학습 및 클러스터링 결과 예측
# km.fit(features)
# kmeans_cluster = km.predict(features)
kmeans_cluster = km.fit_predict(features)

print(kmeans_cluster)
```

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

k-means clustering

k-means clustering

- 클러스터링 결과 확인

```
# 데이터프레임에 군집과 결과인 kmeans_cluster 컬럼 추가
cluster_df['kmeans_cluster'] = kmeans_cluster
cluster_df.head(3)
```

	feature1	feature2	target	kmeans_cluster
0	-1.692427	3.622025	2	1
1	0.697940	4.428867	0	2
2	1.100228	4.606317	0	2

k-means clustering

k-means clustering

- 클러스터의 중심

```
# 클러스터의 중심
km.cluster_centers_
```

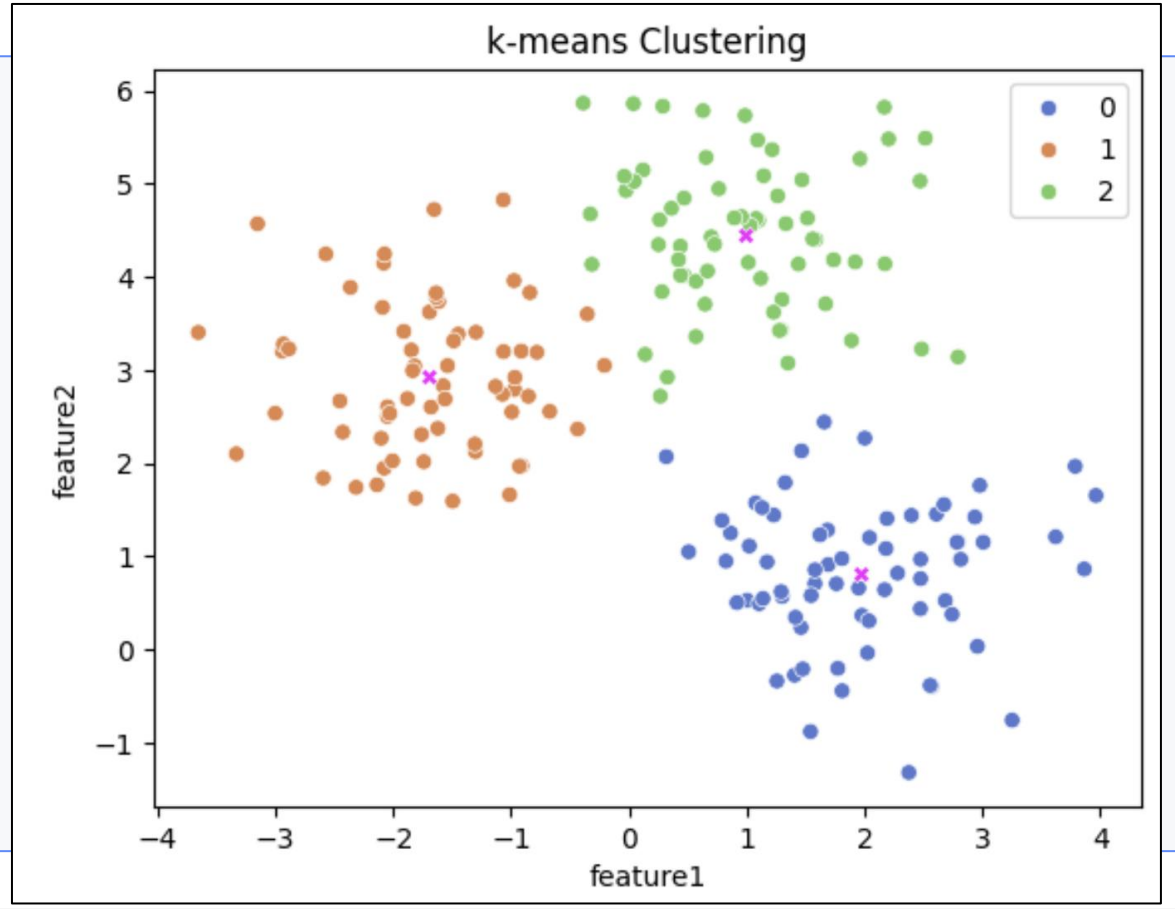
```
array([[ 1.95763312,  0.81041752],
       [-1.70636483,  2.92759224],
       [ 0.990103  ,  4.44666506]])
```

k-means clustering

k-means clustering

- 군집 결과 시각화

```
# 군집화 결과 시각화
sns.scatterplot(data=cluster_df,
                x='feature1',
                y='feature2',
                hue='kmeans_cluster',
                palette='muted')

# 개별 군집의 중심 좌표
sns.scatterplot(x=km.cluster_centers_[:,0],
                y=km.cluster_centers_[:,1],
                color='magenta',
                marker='X')

plt.title('k-means Clustering')
plt.show()
```

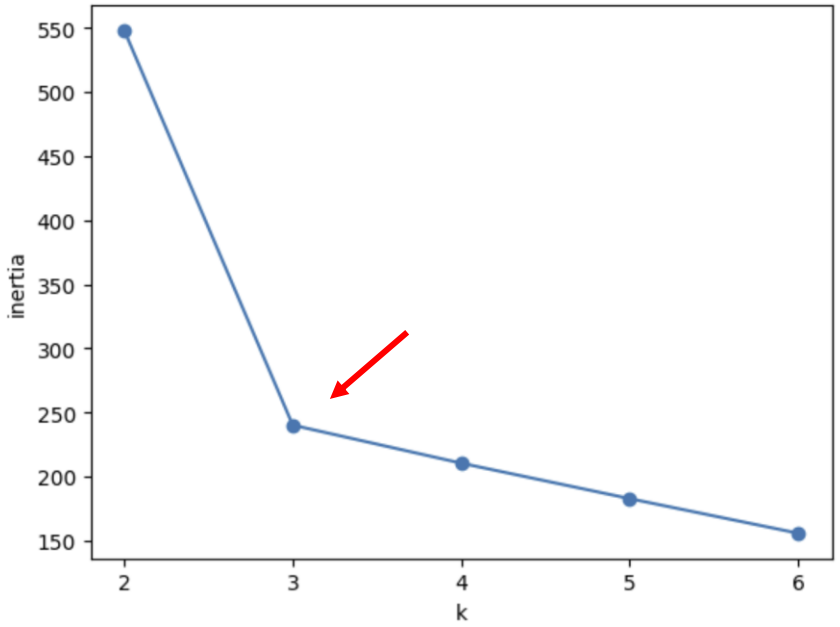


k-means clustering

■ 최적의 군집 개수를 찾는 법

▪ 엘보우(elbow) 기법

✓ 클러스터 개수를 늘려가면서 **이너셔**의 변화를 관찰하여 최적의 클러스터 개수를 찾는 방법



※ 이너셔(inertia)

- 클러스터 중심과 클러스터에 속한 샘플 사이의 거리 제곱 합
  → 클러스터의 샘플이 얼마나 가깝게 있는지를 나타내는 값
- 일반적으로 클러스터 개수가 늘어나면 클러스터 개개의 크기는 줄어들기 때문에 이너셔도 줄어든다.
- 이너셔가 감소하는 속도가 꺾이는 지점을 지나면, 클러스터 개수를 늘려도 클러스터에 밀집된 정도가 개선되지 않는다.

k-means clustering

최적의 군집 개수를 찾는 법

- 군집의 개수를 2부터 7까지 늘려가며 이너셔 구하기

```
inertia = []
for n in range(2,7):
    km = KMeans(n_clusters=n)
    km.fit(features)
    print(km.inertia_)
    inertia.append(km.inertia_)
```

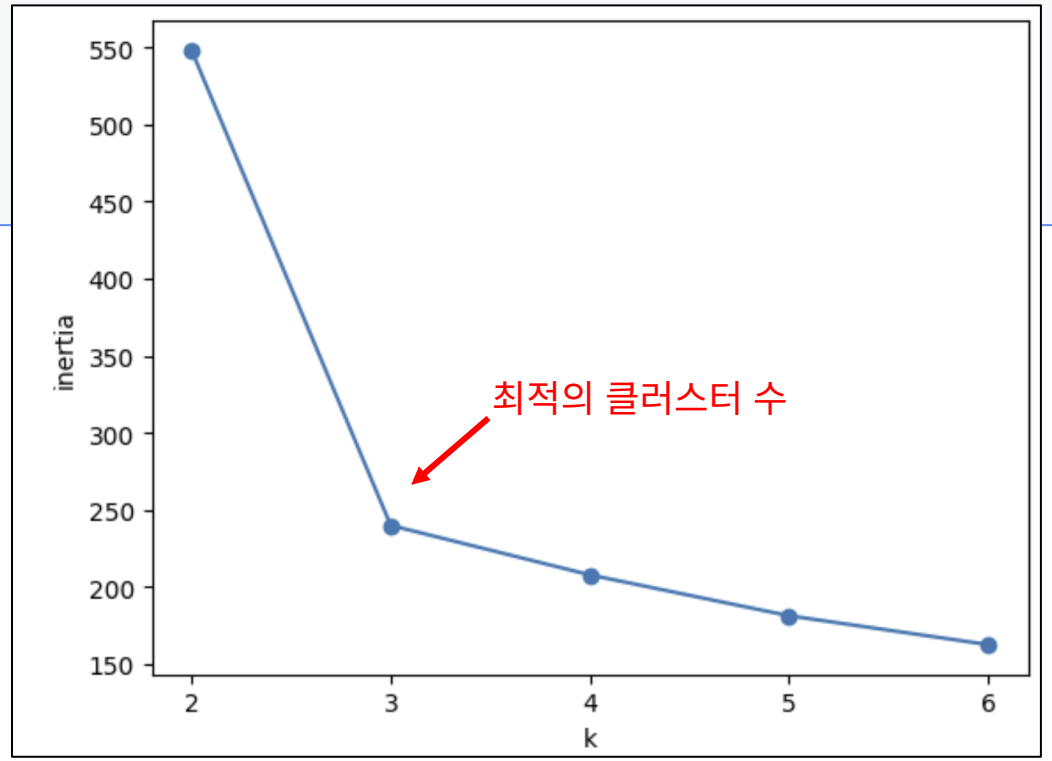
```
548.0904569461312
240.0078759459446
207.92537140645015
181.42054425243245
162.71105860290865
```

k-means clustering

■ 최적의 군집 개수를 찾는 법

- 이너셔 시각화

```
plt.plot(range(2,7), inertia, marker='o')
plt.xticks(range(2,7))
plt.xlabel('k')
plt.ylabel('inertia')
plt.show()
```



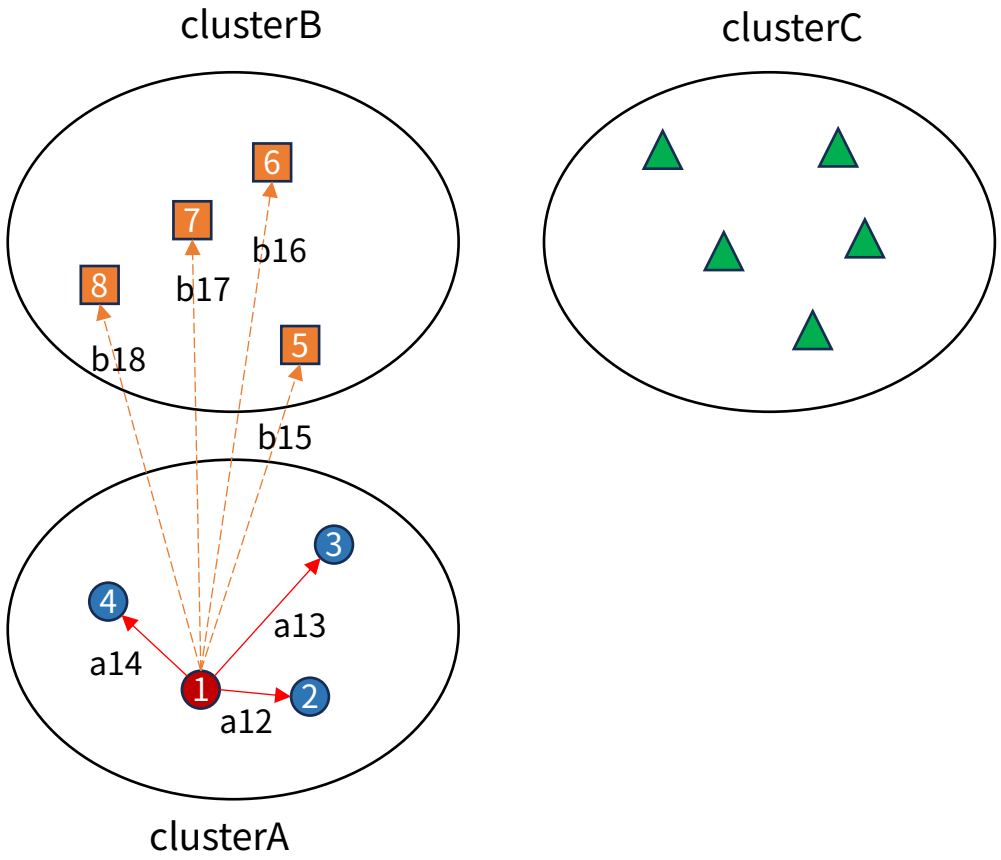
k-means clustering

실루엣 분석(Silhouette analysis)

- 각 군집들이 얼마나 효율적으로 분리되어 있는지 분석한다.
- 다른 군집과의 거리는 떨어져 있고 동일 군집끼리의 데이터는 서로 가깝게 잘 뭉쳐 있는가?
- 실루엣 분석은 실루엣 계수를 기반으로 한다.
- **실루엣 계수(Silhouette coefficient)**
  - 군집화의 품질을 평가하는 지표.
  - 각 개별 데이터 포인트가 자신의 군집에 얼마나 잘 속해 있는지, 다른 군집과는 얼마나 잘 분리되어 있는지 측정한다.
  - -1에서 1 사이의 값을 가지며, 1에 가까울수록 군집화가 잘 되었다고 해석할 수 있다.
    - ✓ 1에 가까워질수록 근처의 군집과 더 멀리 떨어져 있다는 의미이다.
    - ✓ 0에 가까워질수록 가까운 군집과 가까워진다는 의미이다.
    - ✓ -값은 아예 다른 군집에 데이터포인트가 할당되었음을 뜻한다.

k-means clustering

■ 실루엣 계수(Silhouette coefficient)



① **a(i)** : i번째 데이터에서 군집 내 다른 데이터들까지의 평균 거리

$$a(1) = mean([a12, a13, a14])$$

② **b(i)** : i번째 데이터에서 가장 가까운 다른 군집과의 평균거리

$$b(1) = mean([b15, b16, b16, b17])$$

③ **s(i)** : $= \frac{b(i)-a(i)}{\max(a(i),b(i))}$

$$s(1) = \frac{b(1)-a(1)}{b(1)}$$

k-means clustering

■ 실루엣 분석(Silhouette analysis)

- 개별 실루엣 계수

```
# 개별 실루엣 계수
from sklearn.metrics import silhouette_samples
cluster_df['silhouette'] = silhouette_samples(features, kmeans_cluster)
cluster_df.head()
```

	feature1	feature2	target	kmeans_cluster	silhouette
0	-1.692427	3.622025	2	0	0.598109
1	0.697940	4.428867	0	1	0.658958
2	1.100228	4.606317	0	1	0.704928
3	-1.448724	3.384245	2	0	0.609202
4	1.214861	5.364896	0	1	0.658057

k-means clustering

■ 실루엣 분석(Silhouette analysis)

▪ **개별** 실루엣 계수

```
# 다른 군집에 클러스터가 할당된 데이터가 있는지 확인
cond = cluster_df['silhouette'] < 0
cluster_df.loc[cond]
```

	feature1	feature2	target	kmeans_cluster	silhouette

k-means clustering

■ 실루엣 분석(Silhouette analysis)

- **평균** 실루엣 계수

```
# 평균 실루엣 계수
from sklearn.metrics import silhouette_score
silhouette_score(features, kmeans_cluster)
```

```
np.float64(0.5764726251866076)
```

k-means clustering

실루엣 분석(Silhouette analysis)

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.cluster import KMeans

def silhouette_plot(X, n_clusters):
    """
    X: 데이터 (ndarray 또는 DataFrame)
    n_clusters: 클러스터의 수 (k 값)
    """
    # KMeans 모델 학습
    kmeans = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = kmeans.fit_predict(X)

    # 실루엣 점수 계산
    silhouette_avg = silhouette_score(X, cluster_labels)
    sample_silhouette_values = silhouette_samples(X, cluster_labels)

    fig, ax1 = plt.subplots(1, 1)
    fig.set_size_inches(10, 6)

    # 그래프의 y축은 클러스터의 개수만큼 높이 설정
    ax1.set_xlim([-0.1, 1])
    ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])
```

실루엣 시각화
silhouette_analysis.py

k-means clustering

실루엣 분석(Silhouette analysis)

```
    y_lower = 10
    for i in range(n_clusters):
        ith_cluster_silhouette_values = sample_silhouette_values[cluster_labels == i]
        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = plt.cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper), 0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7)

        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        y_lower = y_upper + 10  # 10 for spacing between clusters

    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("Silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    plt.show()
```

실루엣 시각화
silhouette_analysis.py

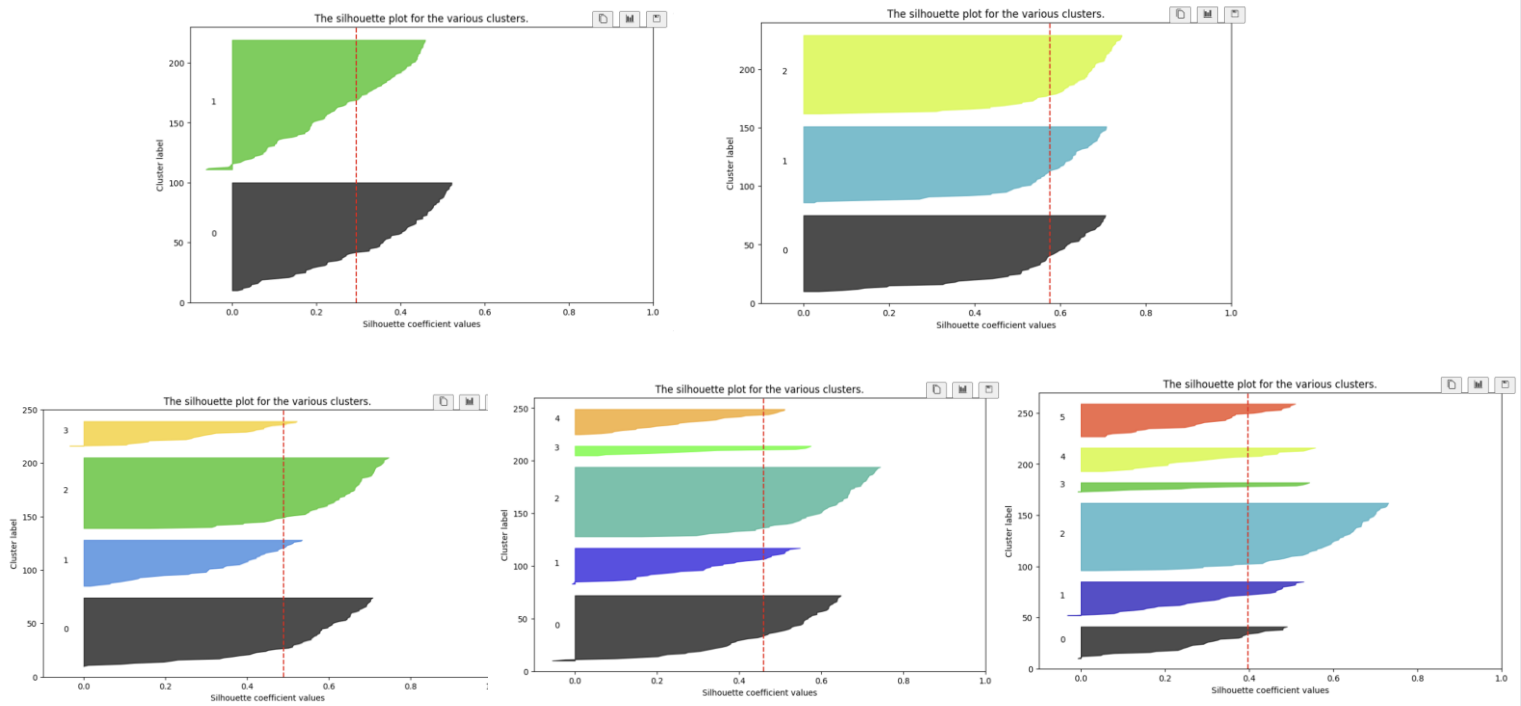
k-means clustering

실루엣 분석(Silhouette analysis)

- 최적의 실루엣 찾기

```
# 최적의 실루엣 찾기
import silhouette_analysis as s
for k in range(2,7):
    s.silhouette_plot(features, k)
```

- 전체 실루엣 계수의 평균값이 1에 가까울수록 좋다.
- 개별 군집의 평균값의 편차가 크지 않은 것이 좋다.
- 각 군집 별 실루엣 계수 평균값이 전체 실루엣 계수 평균값과 크게 벗어나지 않아야 좋다.



DBSCAN

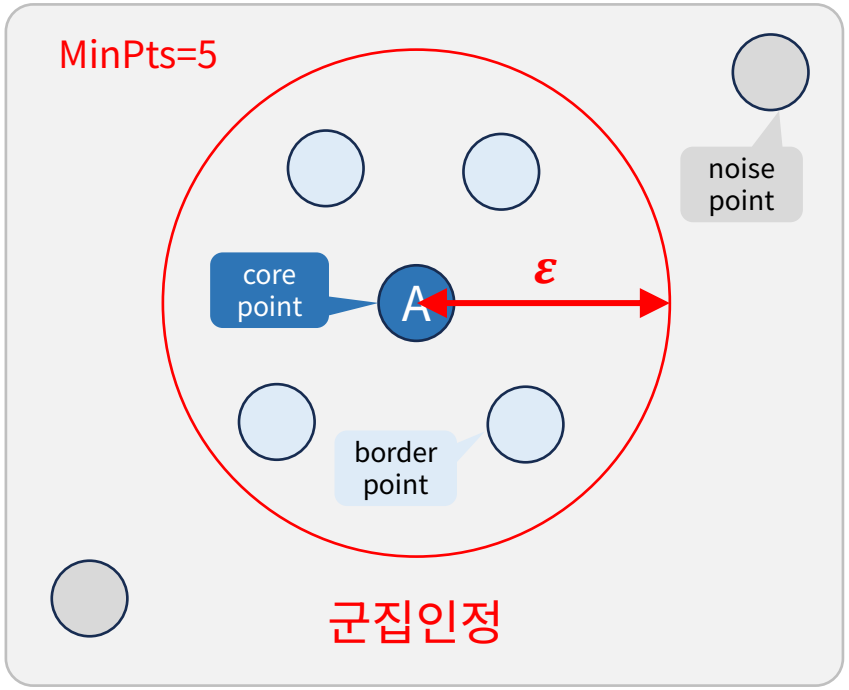
DBSCAN 기본 개념

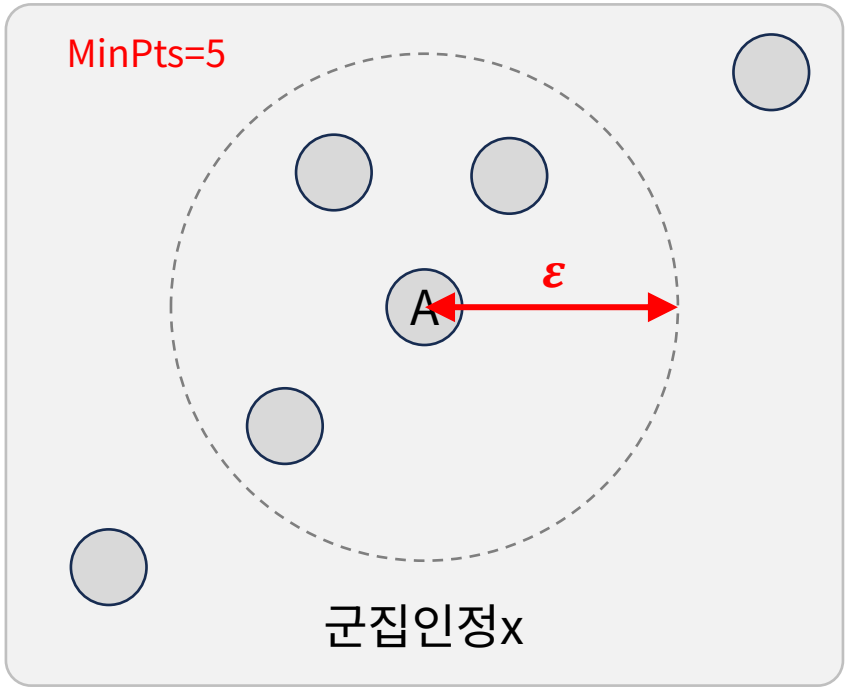
- **D**ensity-**B**ased **S**patial **C**lustering of **A**pplication with **N**oise
- 데이터의 **밀도**가 높은 영역을 클러스터로 정의
- 주요 파라미터
  - **Epsilon(ε)** : 입실론. 한 점을 중심으로 하는 반경의 크기.
  - **MinPts** : 민포인트. 입실론 주변 영역 내에 포함되어야 할 최소 데이터 포인트 수(자신 포함).
- 포인트 분류
  - **Core Point** : ε 반경 내에 MinPts 이상의 포인트가 있는 점
  - **Border Point** : Core Point는 아니지만 Core Point의 ε 반경 내에 있는 점
  - **Noise Point** : Core Point도 Border Point도 아닌 점으로, 이상치로 간주함

DBSCAN

DBSCAN 기본 개념

- ε 반경 내의 데이터 수가 자신 포함하여 **MinPts**개 이상이면 군집으로 인정





DBSCAN

알고리즘 작동 방식

1. 임의의 포인트에서 시작하여 ε 반경 내의 이웃을 찾는다.
2. 이웃의 수가 MinPts 이상이면 새로운 클러스터를 형성한다.
3. 클러스터에 속한 모든 포인트에 대해 같은 과정을 반복하여 클러스터를 확장한다.
4. 모든 포인트를 처리할 때까지 1~3 과정을 반복한다.

DBSCAN

DBSDAN의 장점 및 한계

- 장점
  - ✓ 클러스터 개수를 사전에 지정할 필요가 없다.
  - ✓ 불규칙한 모양의 클러스터도 찾아낸다.
  - ✓ 이상치(Noise)를 효과적으로 식별한다.

- 한계
  - ✓ 밀도가 다양한(밀도가 오밀조밀한 클러스터와 듬성듬성한 클러스터가 함께 존재하는 경우) 클러스터를 처리하기 어려울 수 있다.
  - ✓ 고차원 데이터에서는 성능이 저하될 수 있다.
  - ✓ ε과 MinPts 파라미터 선택에 결과에 큰 영향을 미친다.

DBSCAN

DBSDAN의 선택

- 이상치가 있는 대규모 데이터셋에 유용
- 클러스터의 모양이 불규칙하거나 사전에 클러스터 수를 알기 어려운 경우 적합하다.

DBSCAN

■ 데이터 준비

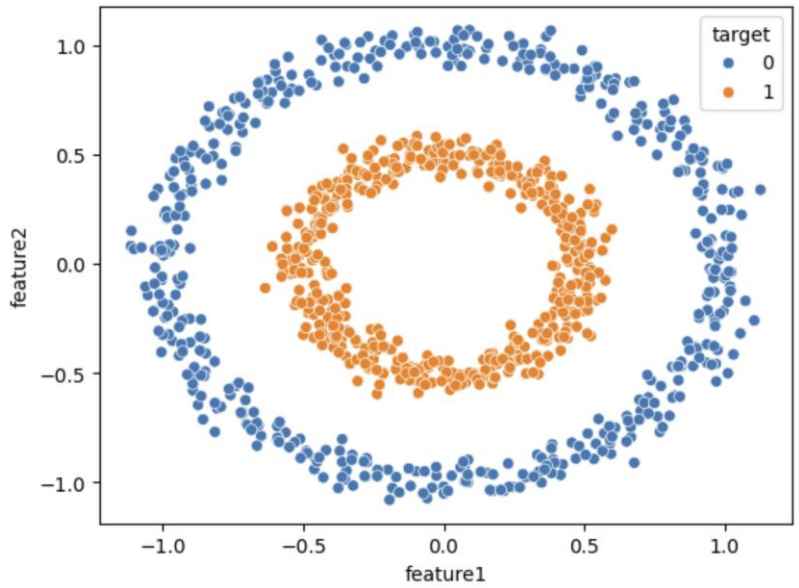
```
from sklearn.datasets import make_circles

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

df = pd.DataFrame(X, columns=['feature1', 'feature2'])
df['target'] = y

df.head()
```

	feature1	feature2	target
0	0.519781	-0.015981	1
1	-0.057719	-0.420279	1
2	-0.805155	-0.662227	0
3	0.316549	0.312730	1
4	-0.304804	0.407563	1

DBSCAN

■ 데이터 준비

```
sns.scatterplot(data=df, x='feature1', y='feature2', hue='target', palette='muted’)
plt.show()
```



DBSCAN

■ k-means 클러스터링

- 클러스터링

```
from sklearn.cluster import KMeans
km = KMeans(n_clusters=2)
df['kmeans'] = km.fit_predict(X)

df.head()
```

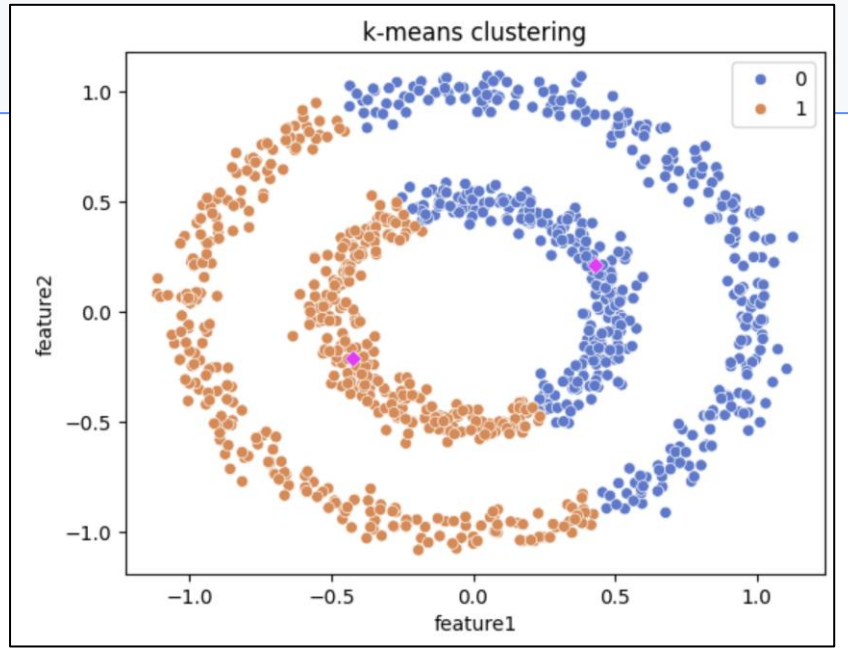
	feature1	feature2	target	kmeans
0	0.519781	-0.015981	1	0
1	-0.057719	-0.420279	1	1
2	-0.805155	-0.662227	0	1
3	0.316549	0.312730	1	0
4	-0.304804	0.407563	1	1

DBSCAN

k-means 클러스터링

- 클러스터링 결과 확인

```
sns.scatterplot(df, x='feature1', y='feature2', hue='kmeans', palette='muted')
sns.scatterplot(x=km.cluster_centers_[:,0], y=km.cluster_centers_[:,1],
                marker='D', color='magenta')
plt.title('k-means clustering')
plt.show()
```



DBSCAN

DBSCAN

- 클러스터링

```
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps=0.2,
                min_samples=10)
df['dbscan'] = dbscan.fit_predict(X)
df.head()
```

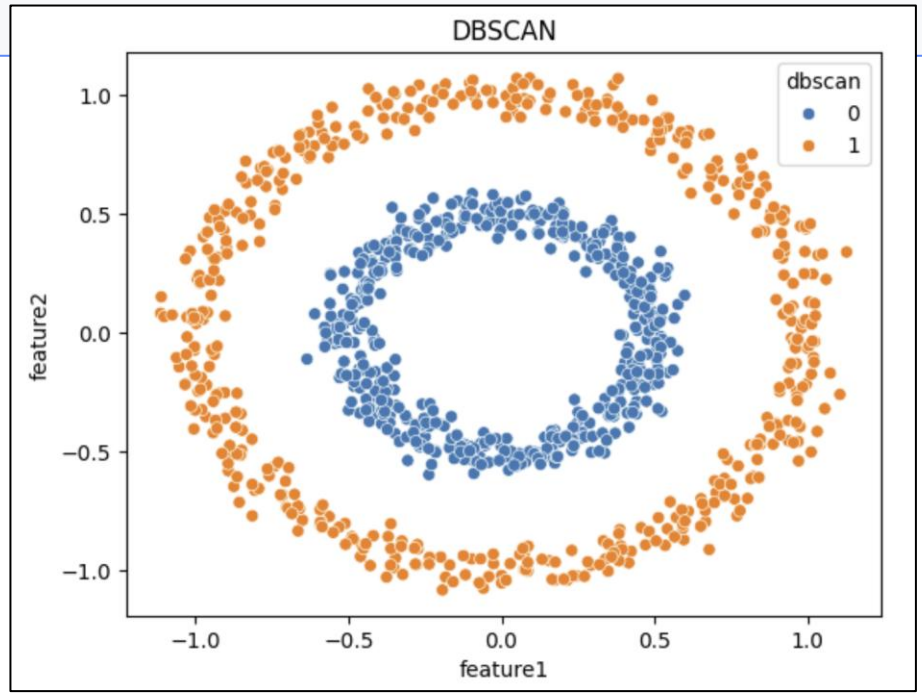
	feature1	feature2	target	kmeans	dbscan
0	0.519781	-0.015981	1	0	0
1	-0.057719	-0.420279	1	1	0
2	-0.805155	-0.662227	0	1	1
3	0.316549	0.312730	1	0	0
4	-0.304804	0.407563	1	1	0

DBSCAN

DBSCAN

- 클러스터링 결과 확인

```
sns.scatterplot(df, x='feature1', y='feature2', hue='dbscan')
plt.title('DBSCAN')
plt.show()
```



프로야구 타자 군집화

데이터 준비

■ 라이브러리 불러오기

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


np.set_printoptions(suppress=True,  # 지수 표기법 억제 (예: 1e-10 대신 0.0000000001로 표시)
                    precision=8)    # 소수점 이하 자리수를 8자리로 표현
```

데이터 준비

■ 데이터 불러오기

```
hitter1 = pd.read_csv('data/2000_2001_hitter.csv')
hitter2 = pd.read_csv('data/2002_2013_hitter.csv')
hitter3 = pd.read_csv('data/2014_hitter.csv')

print(f'hitter1 >>> {hitter1.shape}')
print(f'hitter2 >>> {hitter2.shape}')
print(f'hitter3 >>> {hitter3.shape}')

display(hitter1.head(3))
display(hitter2.head(3))
display(hitter3.head(3))
```

```
hitter1 >>> (89, 36)
hitter2 >>> (505, 39)
hitter3 >>> (55, 39)
```

	YrPlayer	Year	Rank	Player	Team	AVG	G	PA	AB	H	...	OBP	SLG	OPS	ISO	SECA	TA	RC	RC/27	wOBA	XR
0	2000박종호	2000	1	박종호	현대	0.340	121	541	441	150	...	0.428023	0.489796	0.917819	0.149660	0.297052	0.976667	100.354529	8.286154	0.400138	92.804
1	2000김동주	2000	2	김동주	두산	0.339	127	539	469	159	...	0.413729	0.603412	1.017141	0.264392	0.379531	1.063830	115.473840	9.306847	0.431993	105.000
2	2000브리또	2000	3	브리또	SK	0.338	103	452	405	137	...	0.398664	0.533333	0.931997	0.195062	0.274074	0.931655	86.204956	8.224501	0.400969	77.820

3 rows × 36 columns

	YrPlayer	Year	Rank	Player	Team	AVG	G	PA	AB	R	...	MH	RISP	PH-BA	ISO	SECA	TA	RC	RC/27	wOBA	XR
0	2002장성호	2002	1	장성호	KIA	0.343	133	586	481	82	...	48	0.365	0.0	0.178794	0.365904	1.047904	116.003072	9.104892	0.412689	106.706
1	2002마해영	2002	2	마해영	삼성	0.323	133	596	532	92	...	53	0.313	0.0	0.268797	0.359023	1.002674	120.662819	8.573411	0.413068	110.578
2	2002이승엽	2002	3	이승엽	삼성	0.323	133	617	511	123	...	47	0.331	0.0	0.365949	0.538160	1.267409	157.375689	11.770481	0.468948	136.108

3 rows × 39 columns

	YrPlayer	Year	Rank	Player	Team	AVG	G	PA	AB	R	...	MH	RISP	PH-BA	ISO	SECA	TA	RC	RC/27	wOBA	XR
0	2014서건창	2014	1	서건창	넥센	0.370	128	616	543	135	...	66	0.390	0.0	0.176796	0.342541	1.097222	139.001623	10.254218	0.424498	120.634
1	2014김태균	2014	2	김태균	한화	0.365	118	508	422	66	...	44	0.354	0.0	0.199052	0.360190	1.100694	110.631890	10.194748	0.441970	97.816
2	2014손아섭	2014	3	손아섭	롯데	0.362	122	570	483	105	...	54	0.336	0.0	0.175983	0.356108	1.106918	125.666667	10.603125	0.428663	110.456

3 rows × 39 columns

데이터 준비

■ 테이블 병합

```
# 3개의 데이터프레임 모두 병합
hitter_concat = pd.concat([hitter1, hitter2, hitter3],
                          ignore_index=True) # 기존 인덱스 무시
hitter_concat.tail()
```

	YrPlayer	Year	Rank	Player	Team	AVG	G	PA	AB	H	...	TA	RC	RC/27	wOBA	XR	R	SAC	MH	RISP	PH-BA
644	2014이범호	2014	51	이범호	KIA	0.269	105	406	350	94	...	0.884328	64.894286	6.441712	0.369291	63.468	47.0	0.0	24.0	0.293	0.000
645	2014모창민	2014	52	모창민	NC	0.263	122	468	419	110	...	0.676647	53.028291	4.138046	0.319970	55.644	62.0	8.0	22.0	0.263	0.667
646	2014오지환	2014	53	오지환	LG	0.262	113	464	397	104	...	0.818182	64.147759	5.463689	0.336344	63.404	72.0	6.0	21.0	0.359	0.000
647	2014조동화	2014	54	조동화	SK	0.262	125	522	443	116	...	0.539359	48.898621	3.520701	0.293759	48.544	74.0	28.0	26.0	0.328	0.143
648	2014김재호	2014	55	김재호	두산	0.252	122	421	341	86	...	0.620818	41.969406	3.867488	0.304498	44.468	50.0	13.0	21.0	0.298	0.000

5 rows × 41 columns

데이터 준비

■ 변수 선택

```
# 변수 선택
X = hitter_concat[['OPS','ISO','SECA','TA','RC','RC/27','wOBA','XR']]
y = hitter_concat['YrPlayer']

print(X.shape, y.shape)
display(X.head())
display(y.head())
```

```
(649, 8) (649,)
```

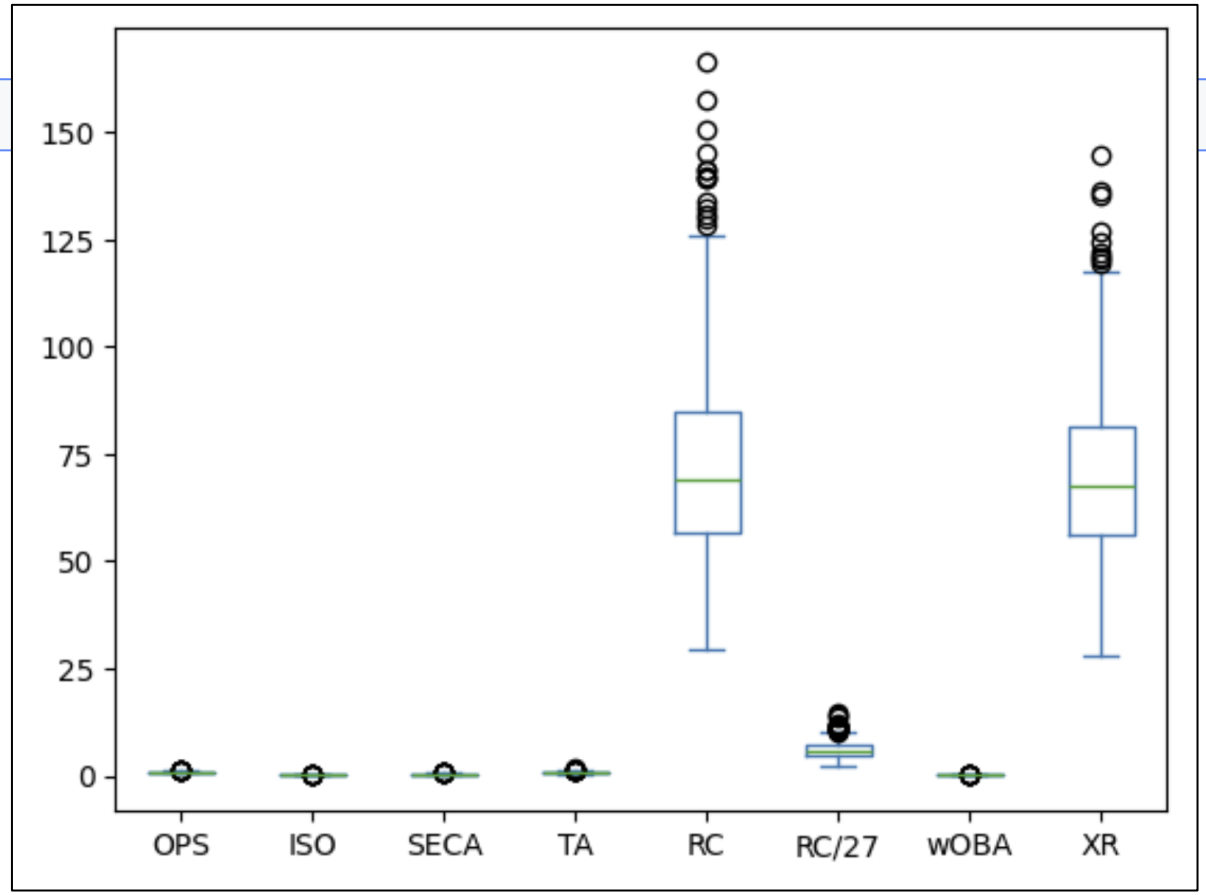
	OPS	ISO	SECA	TA	RC	RC/27	wOBA	XR
0	0.917819	0.149660	0.297052	0.976667	100.354529	8.286154	0.400138	92.804
1	1.017141	0.264392	0.379531	1.063830	115.473840	9.306847	0.431993	105.000
2	0.931997	0.195062	0.274074	0.931655	86.204956	8.224501	0.400969	77.820
3	1.031229	0.284188	0.423077	1.110092	119.916981	9.811389	0.436688	107.594
4	0.932665	0.231504	0.324582	0.923588	82.605677	7.125729	0.394586	80.284

```
0     2000박종호
1     2000김동주
2     2000브리또
3     2000송지만
4    2000데이비스
Name: YrPlayer, dtype: object
```

데이터 준비

■ 데이터 분포 확인

```
X.plot(kind='box')
```



데이터 준비

■ 데이터 스케일링

```
# 데이터 스케일링
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X.loc[:,'OPS':'XR'] = scaler.fit_transform(X)
X.head()
```

	OPS	ISO	SECA	TA	RC	RC/27	wOBA	XR
0	0.976149	-0.074229	0.104176	1.108646	1.248725	1.204841	1.118807	1.214334
1	1.897116	1.679891	1.054165	1.654227	1.929967	1.758231	1.894866	1.857449
2	1.107616	0.619910	-0.160485	0.826902	0.611178	1.171415	1.139040	0.424202
3	2.027750	1.982543	1.555727	1.943794	2.130165	2.031780	2.009248	1.994235
4	1.113814	1.177062	0.421269	0.776410	0.449003	0.575692	0.983538	0.554133

k-means clustering

■ 엘보우 기법

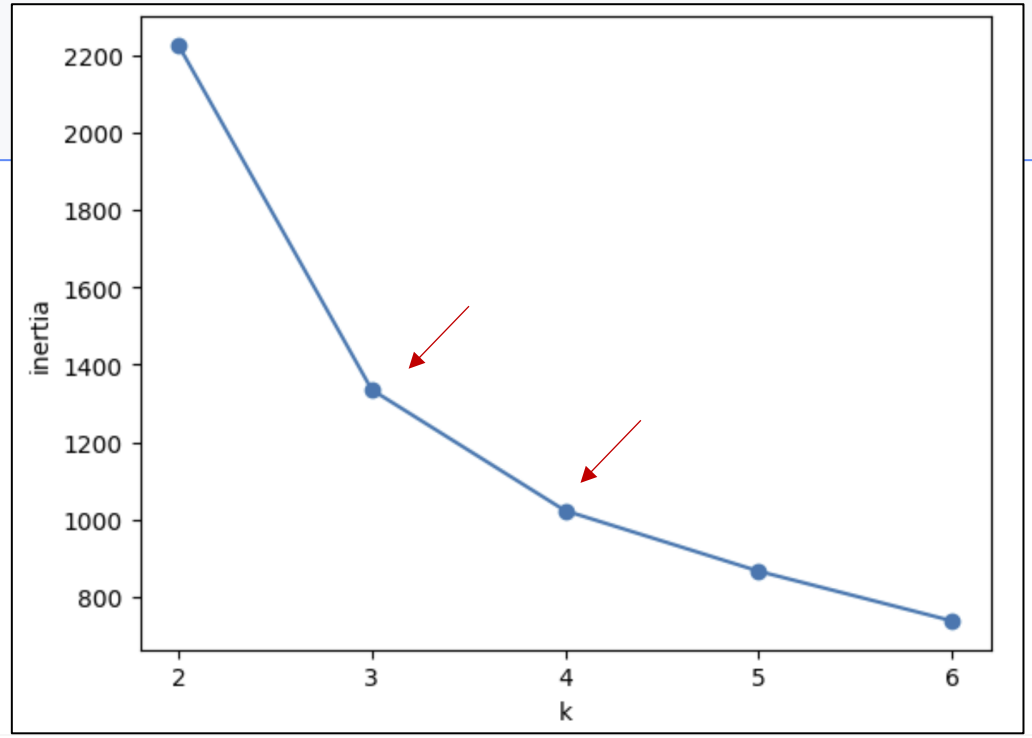
```
# 엘보우 기법으로 최적의 k 찾기
from sklearn.cluster import KMeans
inertia = []
for n in range(2,7):
    km = KMeans(n_clusters=n)
    km.fit(X)
    print(km.inertia_)
    inertia.append(km.inertia_)
```

```
2225.953572555236
1335.4311472249044
1022.911428117714
866.5429996546825
738.1526654441549
```

k-means clustering

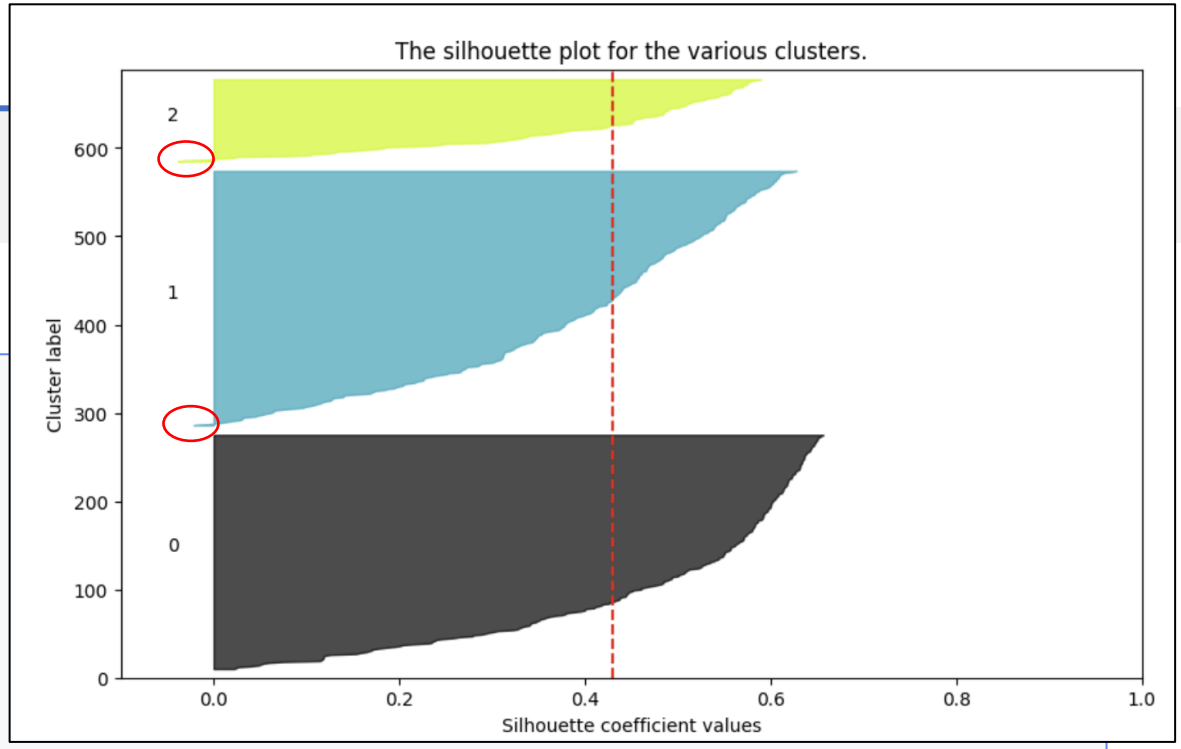
■ 엘보우 기법

```
plt.plot(range(2,7), inertia, marker='o')
plt.xticks(range(2,7))
plt.xlabel('k')
plt.ylabel('inertia')
plt.show()
```



k-means clustering

군집화

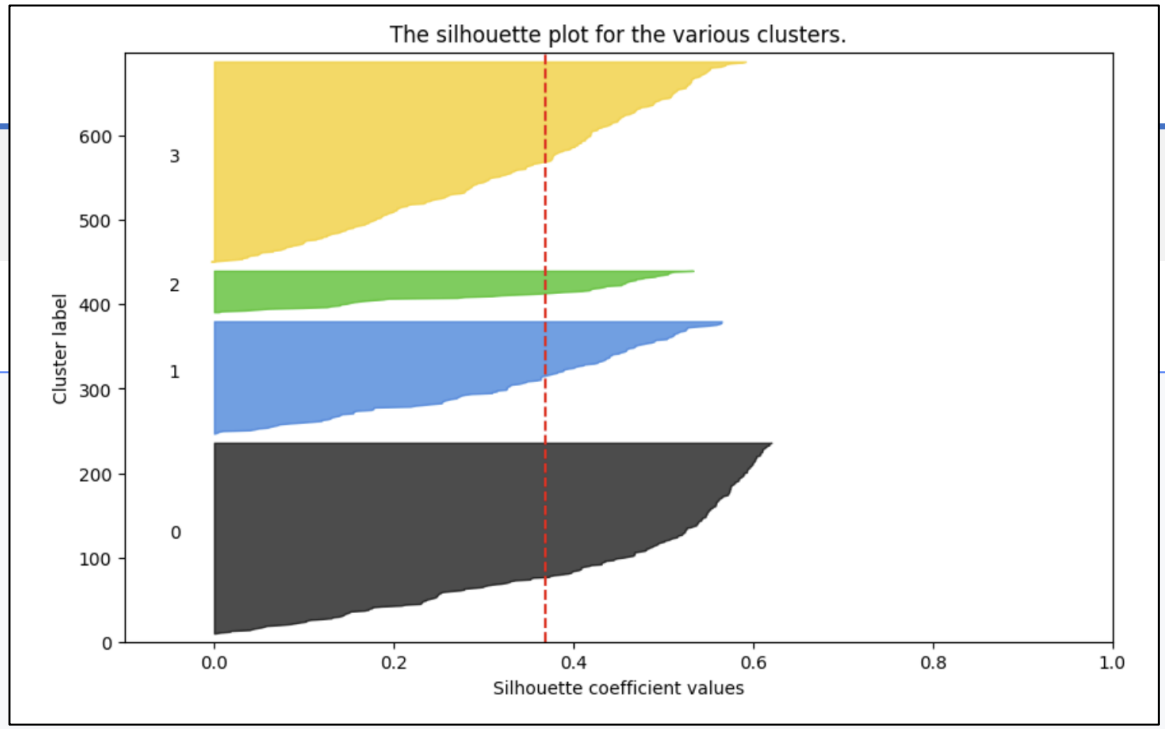



```
k = 3
# k-means clustering
from sklearn.cluster import KMeans
km = KMeans(n_clusters=k, random_state=10)
kmeans_cluster = km.fit_predict(X)

# 실루엣 점수
from sklearn.metrics import silhouette_score
print(f'실루엣 점수:{silhouette_score(X, kmeans_cluster)}')

# 실루엣 시각화
import silhouette_analysis as s
s.silhouette_plot(X, k)
```

```
실루엣 점수:0.4295317550526677
```

k-means clustering

군집화




```
k = 4
# k-means clustering
from sklearn.cluster import KMeans
km = KMeans(n_clusters=k, random_state=10)
kmeans_cluster = km.fit_predict(X)

# 실루엣 점수
from sklearn.metrics import silhouette_score
print(f'실루엣 점수:{silhouette_score(X, kmeans_cluster)}')

# 실루엣 시각화
import silhouette_analysis as s
s.silhouette_plot(X, k)
```

```
실루엣 점수:0.3686437769255018
```

k-means clustering

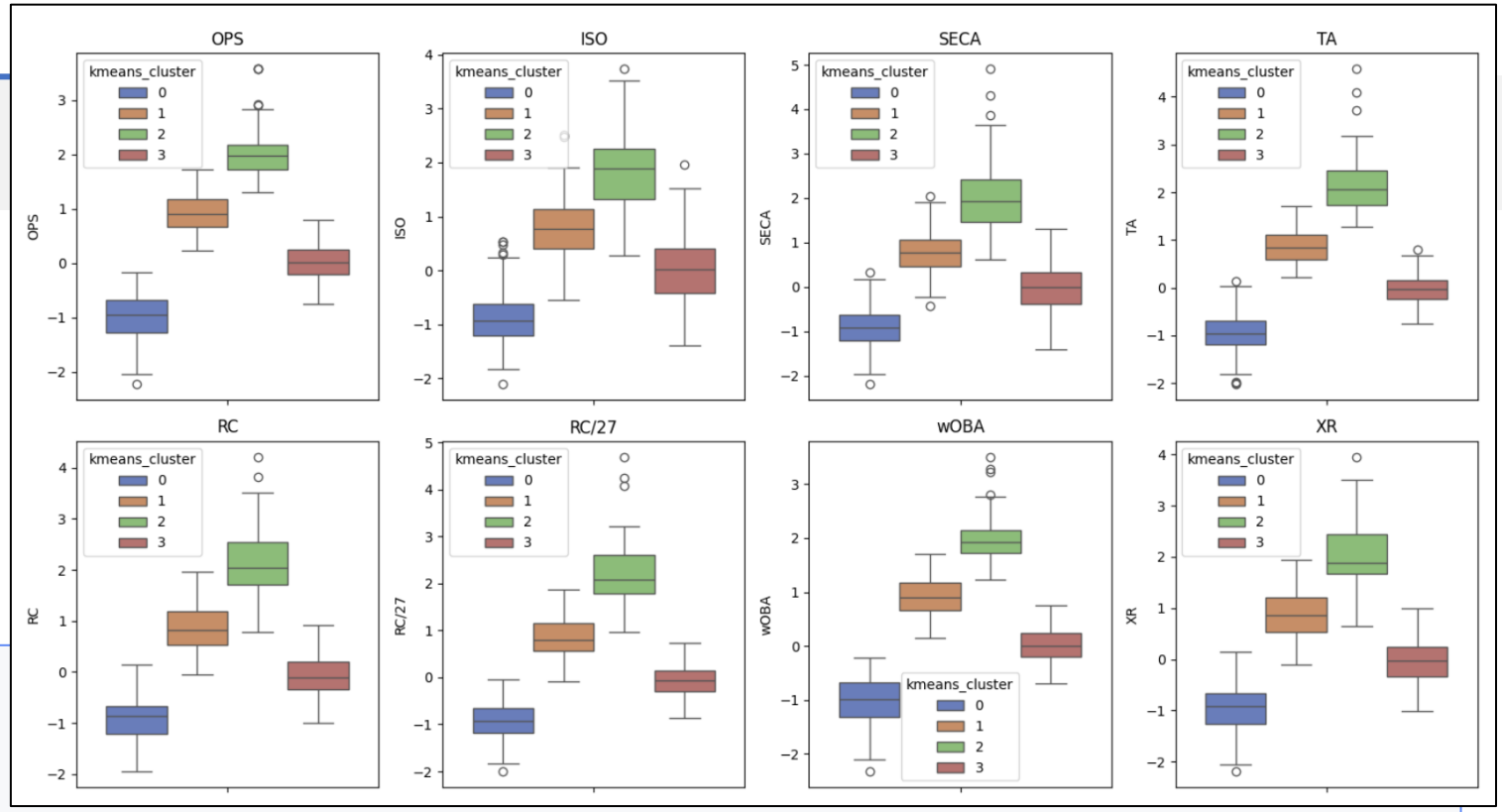
■ 군집화 결과 분석

```
# 데이터프레임 생성
df = pd.DataFrame(X, columns=X.columns)
df['kmeans_cluster'] = kmeans_cluster
df.head()
```

	OPS	ISO	SECA	TA	RC	RC/27	wOBA	XR	kmeans_cluster
0	0.976149	-0.074229	0.104176	1.108646	1.248725	1.204841	1.118807	1.214334	1
1	1.897116	1.679891	1.054165	1.654227	1.929967	1.758231	1.894866	1.857449	2
2	1.107616	0.619910	-0.160485	0.826902	0.611178	1.171415	1.139040	0.424202	1
3	2.027750	1.982543	1.555727	1.943794	2.130165	2.031780	2.009248	1.994235	2
4	1.113814	1.177062	0.421269	0.776410	0.449003	0.575692	0.983538	0.554133	1

k-means clustering

군집화 결과 분석




```
# 군집별 데이터 특징 분석
cols = df.columns[:-1]

plt.figure(figsize=(15,8))
for i, col in enumerate(cols):
    plt.subplot(2,4,i+1)
    sns.boxplot(data=df, y=col, hue='kmeans_cluster', palette='muted')
    plt.title(col)
plt.tight_layout()
```

k-means clustering

■ 실루엣 분석

```
# 개별 데이터 실루엣 계수 컬럼 추가
from sklearn.metrics import silhouette_samples
df['silhouette'] = silhouette_samples(X, cluster)
df.head(3)
```

	OPS	ISO	SECA	TA	RC	RC/27	wOBA	XR	cluster	silhouette
0	0.976149	-0.074229	0.104176	1.108646	1.248725	1.204841	1.118807	1.214334	1	0.433784
1	1.897116	1.679891	1.054165	1.654227	1.929967	1.758231	1.894866	1.857449	2	0.278949
2	1.107616	0.619910	-0.160485	0.826902	0.611178	1.171415	1.139040	0.424202	1	0.365366

k-means clustering

■ 실루엣 분석

```
# 잘못 군집화 된 클러스트
df.loc[df['silhouette']<0]
```

	OPS	ISO	SECA	TA	RC	RC/27	wOBA	XR	cluster	silhouette
106	-0.487238	-0.525068	-0.672578	-0.639331	-0.228069	-0.50102	-0.465956	-0.251185	3	-0.003318

k-means clustering

■ 실루엣 분석

```
# 클러스터 별 중심 player

df['player'] = y # player 컬럼 추가
max_idx = df.groupby('cluster')['silhouette'].idxmax()
df.loc[max_idx]
```

	OPS	ISO	SECA	TA	RC	RC/27	wOBA	XR	cluster	silhouette	player
325	-1.099228	-1.007645	-1.334422	-1.194599	-1.091820	-1.073170	-1.098797	-1.185063	0	0.620514	2007김민재
475	1.070553	0.938210	0.769761	1.190167	0.779519	0.959467	1.112819	0.801445	1	0.565102	2011최정
553	2.099809	1.986463	2.518512	2.502596	2.497083	2.534148	2.085315	2.440240	2	0.533266	2013박병호
192	-0.023610	0.068359	-0.132569	-0.101811	-0.008476	-0.149790	-0.020237	0.084294	3	0.591658	2004박진만

주성분 분석(PCA)

① 전체 주성분 계산

② 누적 분산 비율로 중요도 확인

③ 주요 성분만 선택하여 차원 축소

주성분 분석(PCA)

① 전체 주성분 계산

```
# 모든 특성(컬럼)에 대해 주성분 계산하여 각 데이터포인트를 주성분 공간으로 변환
from sklearn.decomposition import PCA
pca = PCA()
principal_components = pca.fit_transform(X)
principal_components
```

```
array([[ 2.48864775, -1.29871071, -0.18369185, ..., 0.11512947, -0.05012084, -0.00542238], [ 4.8638416 , -
0.32843484, 0.04378699, ..., 0.02815584, -0.05298553, 0.00816747], [ 2.05850718, -0.4397998 , -
0.79470761, ..., 0.02887054, -0.04552095, -0.02742987], ..., [-0.3971905 , 0.69084294, 0.14211226, ...,
0.09606087, -0.0169874 , 0.04994242], [-3.94087277, -0.42488303, 0.27501545, ..., -0.11415355, 0.00769498,
0.00666951], [-3.33227219, 0.22550401, -0.14325161, ..., -0.0513692 , 0.01871363, -0.02078552]], shape=(649,
8))
```

주성분 분석(PCA)

② 누적 분산 비율로 중요도 확인

```
# 컬럼별 설명된 분산 비율 확인
explained_variance = pca.explained_variance_ratio_
explained_variance
```

```
array([0.90514573, 0.05062617, 0.02188854, 0.0184561 , 0.00299299, 0.00055624, 0.00018392, 0.00015031])
```

```
# 누적분산 확인
cumulative_variance = explained_variance.cumsum()
cumulative_variance
```

```
array([0.90514573, 0.95577189, 0.97766044, 0.99611654, 0.99910953, 0.99966577, 0.99984969, 1. ])
```

주성분 분석(PCA)

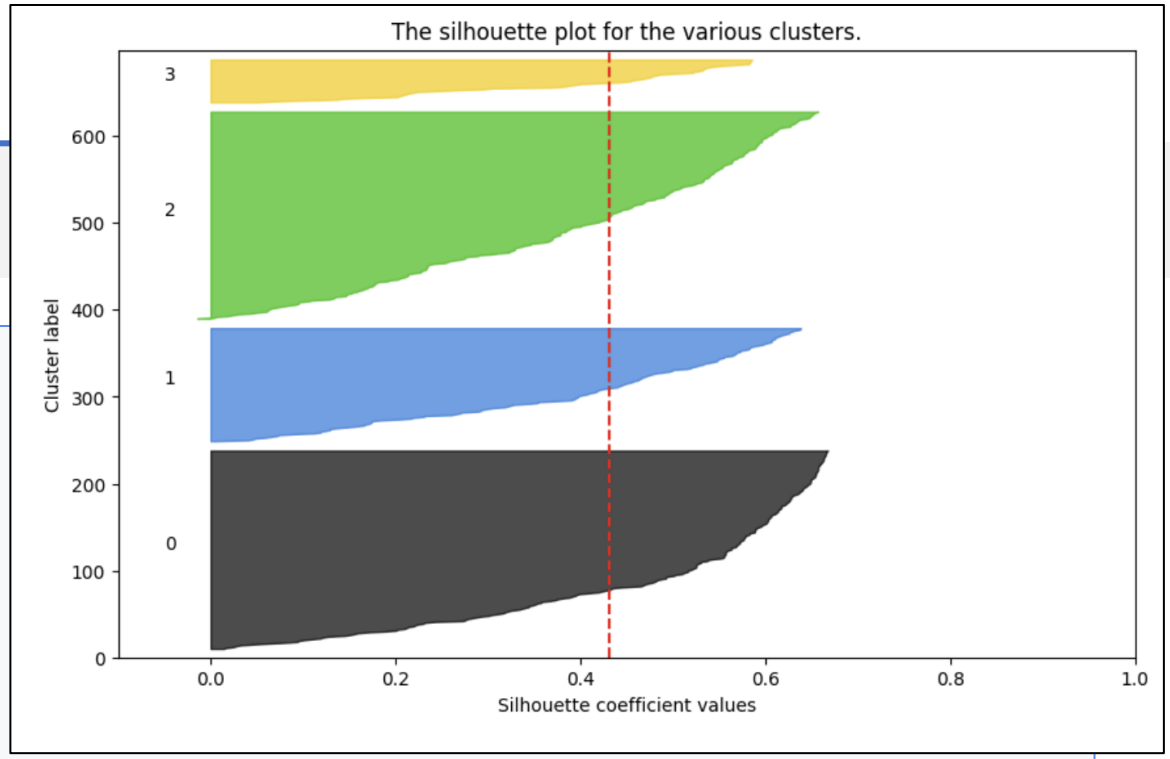
③ 주요 성분만 선택하여 차원 축소

```
# 2개의 주성분 추출
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principal_components = pca.fit_transform(X_scaled)
principal_components
```

```
array([[ 2.48864775, -1.29871071],
       [ 4.8638416 , -0.32843484],
       [ 2.05850718, -0.4397998 ],
       ...,
       [-0.3971905 ,  0.69084294],
       [-3.94087277, -0.42488303],
       [-3.33227219,  0.22550401]])
```

주성분 분석(PCA)

■ 주성분으로 군집화



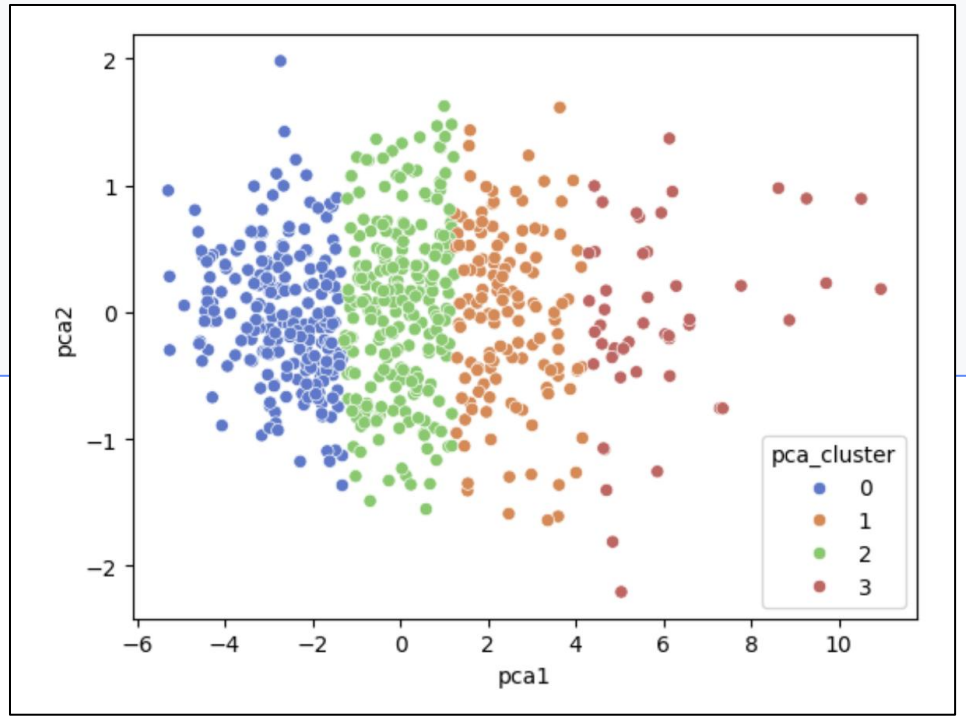

```
# 주성분으로 군집화

k = 4
# k-means clustering
from sklearn.cluster import KMeans
km = KMeans(n_clusters=k, random_state=10)
pca_cluster = km.fit_predict(principal_components)

# 실루엣 점수
from sklearn.metrics import silhouette_score
print(f'실루엣 점수:{silhouette_score(principal_components, pca_cluster)}')

# 실루엣 시각화
import silhouette_analysis as s
s.silhouette_plot(principal_components, k)
```

```
실루엣 점수:0.43183378066194306
```

주성분 분석(PCA)

■ 주성분으로 군집화

```
# 군집 결과 시각화
df_pca = pd.DataFrame(principal_components,
                      columns=['pca1','pca2'])
df_pca['pca_cluster'] = pca_cluster

sns.scatterplot(data=df_pca, x='pca1', y='pca2',
                hue='pca_cluster', palette='muted');
```



주성분 분석(PCA)

■ 군집 결과 비교

```
# pca_cluster 컬럼 추가
df['pca_cluster'] = pca_cluster
df.head()
```

	OPS	ISO	SECA	TA	RC	RC/27	wOBA	XR	cluster	silhouette	player	pca_cluster
0	0.976149	-0.074229	0.104176	1.108646	1.248725	1.204841	1.118807	1.214334	1	0.433784	2000박종호	1
1	1.897116	1.679891	1.054165	1.654227	1.929967	1.758231	1.894866	1.857449	2	0.278949	2000김동주	3
2	1.107616	0.619910	-0.160485	0.826902	0.611178	1.171415	1.139040	0.424202	1	0.365366	2000브리또	1
3	2.027750	1.982543	1.555727	1.943794	2.130165	2.031780	2.009248	1.994235	2	0.469111	2000송지만	3
4	1.113814	1.177062	0.421269	0.776410	0.449003	0.575692	0.983538	0.554133	1	0.430869	2000데이비스	1

주성분 분석(PCA)

■ 군집 결과 비교

```
df[['cluster','pca_cluster']].value_counts()
```

```
cluster  pca_cluster
3        2              236
0        0              227
1        1              131
2        3               50
1        2                3
3        0                2
Name: count, dtype: int64
```