비지도학습 - 군집분석

> ## K-means clustering
>
> m^(k), i 차원 x = (x0, ..., xi)
>
> > 1. 그룹 평균 m^(k) 초기화
> > 2. 그룹 할당 (argmin[dm, x])
> > 3. 평균 업데이트
> > 4. 반복
>
> 가중치를 주지 않아 밀도 차이가 있을 경우 클러스터링이 잘 되지 않음. 초기 K 값은 사용자가 정해야 함.

In [11]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs
```

In [15]:
```
X, y = make_blobs(n_samples=100, n_features =2, centers=5, random_state=10)
```

In [17]:
```
print(X.shape, y.shape)
```

```
(100, 2) (100,)
```

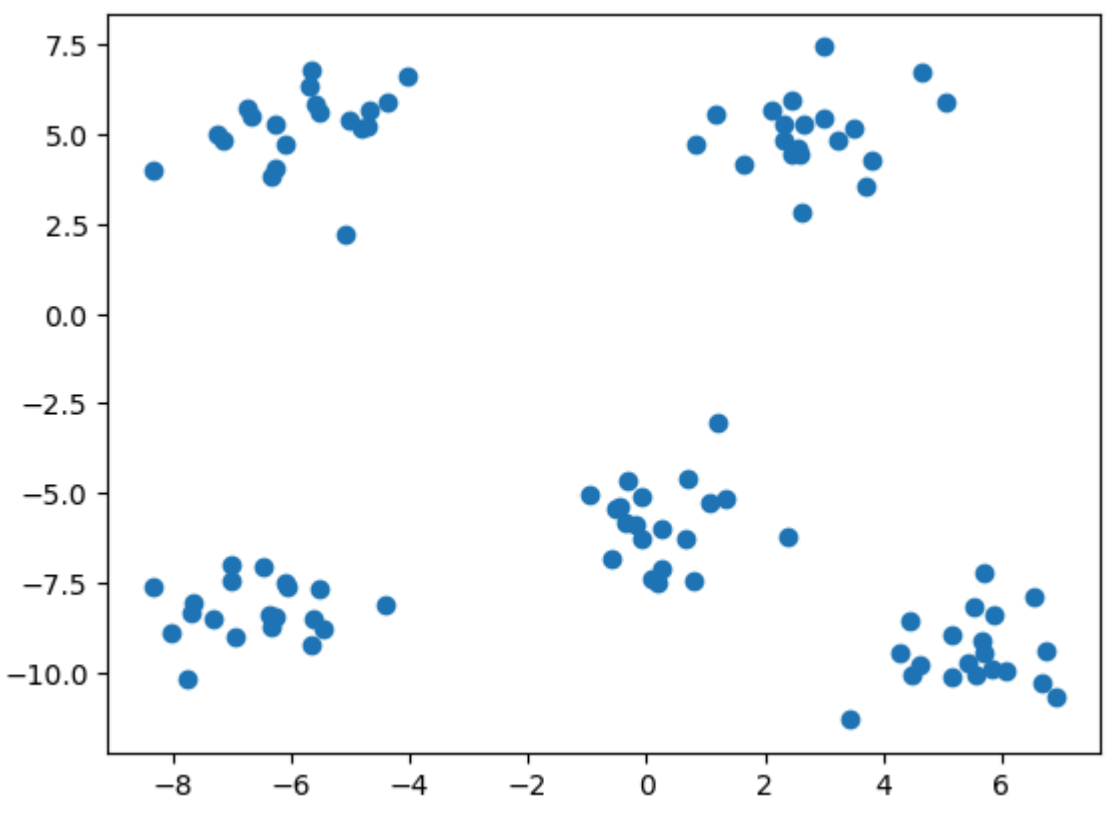
In [27]:
```
X[:10]
```

Out[27]:
```
array([[ -5.57785425,   5.87298826],
       [  1.62783216,   4.17806883],
       [ -6.95088443,  -9.02459449],
       [ -0.95276902,  -5.03431566],
       [  6.55010412,  -7.9123388 ],
       [ -6.67870531,   5.52444477],
       [ -5.52802829,   5.62491189],
       [ -8.03866378,  -8.91215049],
       [  5.55912116, -10.06110303],
       [ -7.26457869,   4.98882111]])
```

In [31]:
```
y[:20]
```

Out[31]:
```
array([3, 1, 4, 2, 0, 3, 3, 4, 0, 3, 4, 3, 0, 2, 0, 3, 1, 2, 0, 3])
```

In [33]:
```
plt.scatter(X[:,0], X[:,1])
plt.show()
```



In [61]:
```
# k = 5로 설정해야지~!

from sklearn.cluster import KMeans
kmc = KMeans(n_clusters=5, init='random', max_iter=200, random_state=222)
kmc.fit(X)
label_kmc = kmc.labels_
```

In [63]:
```
print(label_kmc)
```

```
[0 3 4 2 1 0 0 4 1 0 4 0 1 2 1 0 3 2 1 0 0 2 3 4 0 0 2 1 4 3 2 2 1 2 0 1 4
 2 4 3 2 2 2 1 4 4 0 1 3 1 3 1 0 4 1 1 2 3 2 4 1 3 3 2 0 3 1 3 1 4 4 0 3 1
 2 2 4 3 0 0 1 1 3 2 3 4 2 4 2 4 3 4 0 3 3 3 4 0 0 4]
```

In [65]:
```
kmc_columns = ['kmc_comp1', 'kmc_comp2']
X_kmc_df = pd.DataFrame(X, columns=kmc_columns)
X_kmc_df['target'] = y
X_kmc_df['label_kmc'] = label_kmc
X_kmc_df.head()
```

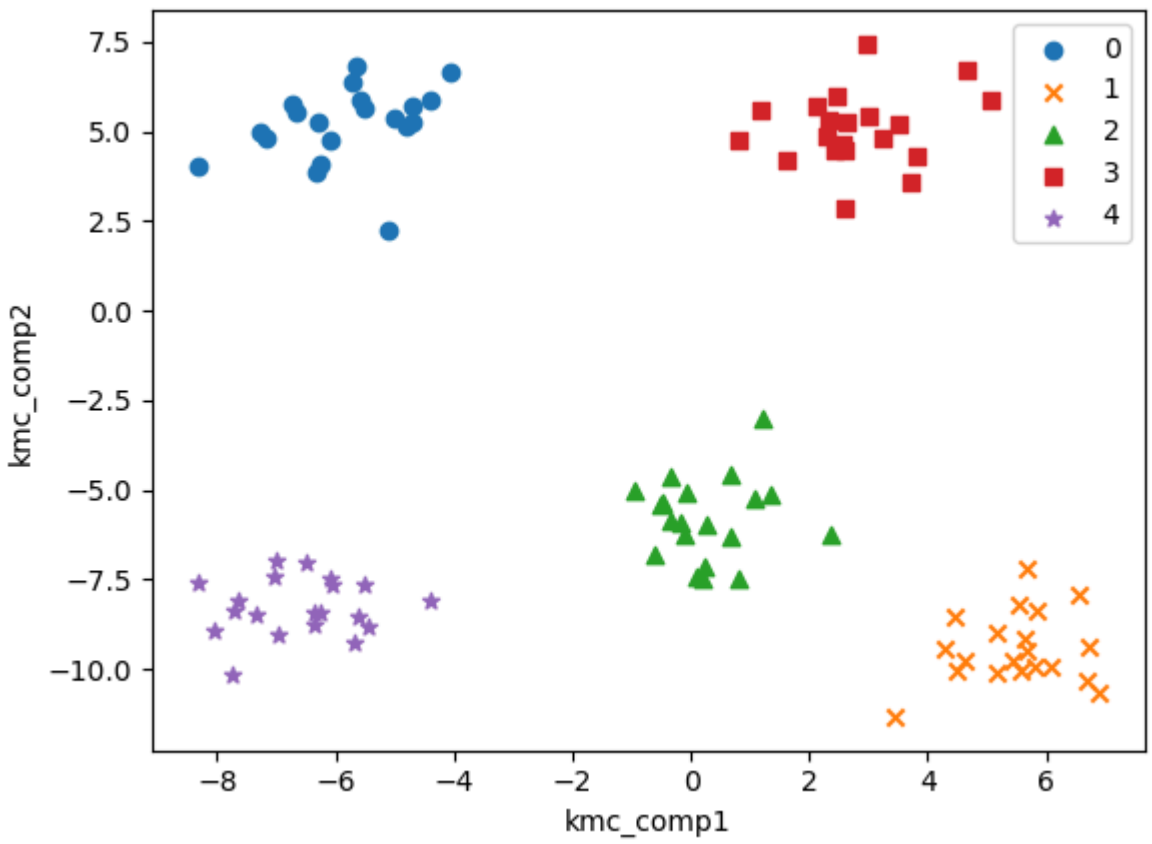
Out[65]:

	kmc_comp1	kmc_comp2	target	label_kmc
0	-5.577854	5.872988	3	0
1	1.627832	4.178069	1	3
2	-6.950884	-9.024594	4	4
3	-0.952769	-5.034316	2	2
4	6.550104	-7.912339	0	1

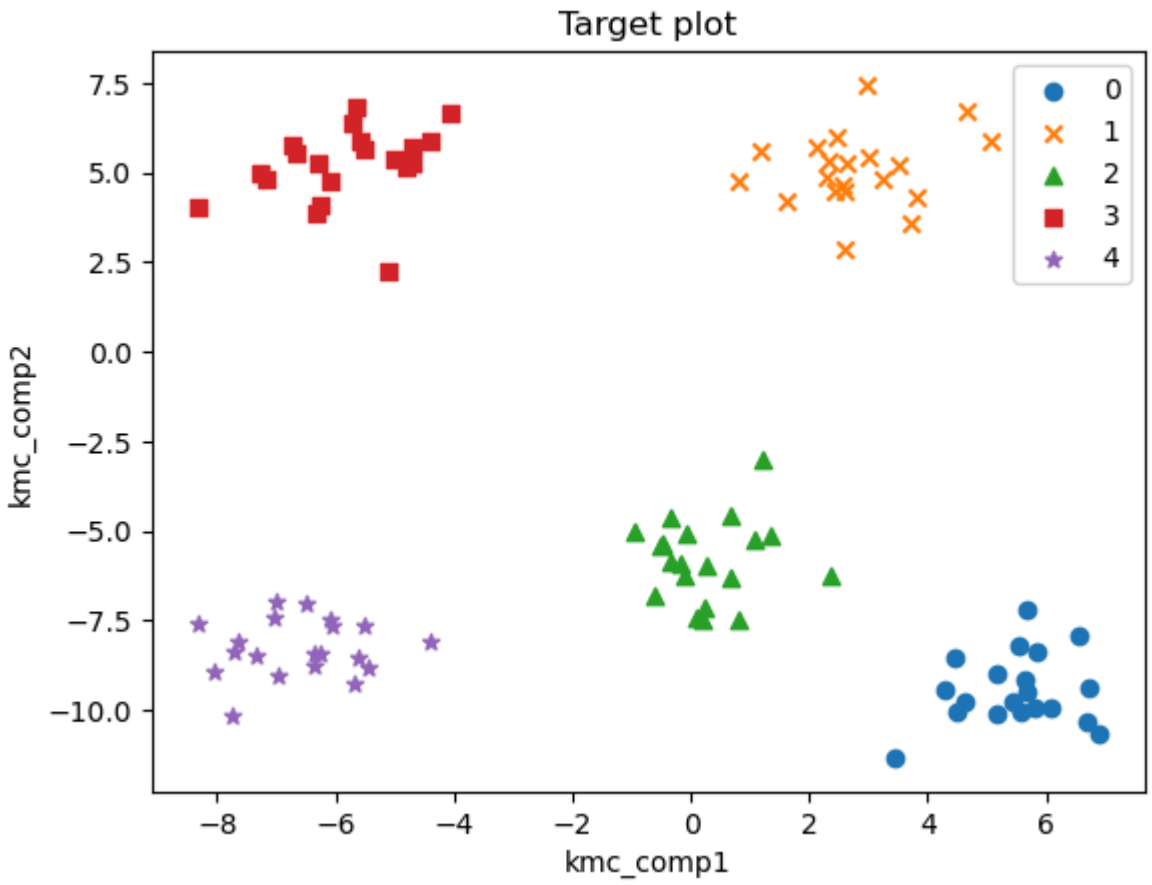
In [69]:
```
print(set(X_kmc_df['target']))
print(set(X_kmc_df['label_kmc']))
```

```
{0, 1, 2, 3, 4}
{0, 1, 2, 3, 4}
```

In [73]:
```
df = X_kmc_df
markers = ['o', 'x', '^', 's', '*']

for i, mark in enumerate(markers):
    df_i = df[df['label_kmc']==i]
    target_i = i
    X1 = df_i['kmc_comp1']
    X2 = df_i['kmc_comp2']
    plt.scatter(X1, X2, marker=mark, label=target_i)

plt.xlabel('kmc_comp1')
plt.ylabel('kmc_comp2')
plt.legend()
plt.show()
```



In [75]:
```
# 실제 target plot

for i, mark in enumerate(markers):
    df_i = df[df['target']==i]
    target_i = i
    X1 = df_i['kmc_comp1']
    X2 = df_i['kmc_comp2']
    plt.scatter(X1, X2, marker=mark, label=target_i)

plt.title('Target plot')
plt.xlabel('kmc_comp1')
plt.ylabel('kmc_comp2')
plt.legend()
plt.show()
```



In [77]:
```
from sklearn.metrics import silhouette_score
sil_score = silhouette_score(X, label_kmc)
print(sil_score)
```

```
0.7598181300128782
```

계층 클러스터링

1. 병합 계층 클러스터링 (Agglomerative) : 개별 데이터 포인트를 하나의 클러스터로 설정하고 시작, 이후 유사도에 따라 합침.

   > 연결방법 (Linkage Method) : 단일연결(Single Method), 가장 가까운 거리의 데이터를 비교 / 완전연결(Complete Method) 가장 먼거리의 데이터를 비교

2. 분할 계층 클러스터링 (Divisive) : 전체 데이터 셋을 하나의 클러스터로 놓고 시작, 가장 멀리 떨어진 데이터를 다른 클러스터로 분리.

Dedrogram 으로 시각화 가능

> 병합계층클러스터링
>
> 1. 전체 n 개 데이터로 구성된 데이터 셋을 n 개의 클러스터로 설정
> 2. 각 데이터간 거리행렬 D(n*m) 계산. D 는 대칭행렬
> 3. 거리행렬 D 에서 가장 가까운 클러스터 쌍을 찾아서 합침. (u, v -> uv)
> 4. 거리 행렬에 새로운 클러스터(uv)를 설정하고 행, 열을 추가 및 기존 u, v 행, 열 은 삭제함.
> 5. 단일연결의 경우 min / 완전연결의 경우 max 를 이용해 거리계산을 하여 3-4를 반복함.

> Ward's 계층 클러스터링 정보의 손실을 최소화 하는 방향으로 클러스터링, 오차제곱합을 최소화하는 방법 이용
> 오차제곱합 > 각 데이터포인트가 클러스터 중심에서 얼마나 떨여져 있는 지를 의미함.

1. 각 데이터포인트를 하나의 클러스터로 가정, 이 때 오차제곱합은 0이 됨.
2. 모든 가능한 클러스터쌍을 고려하여 오차제곱합을 계산 후 가장 작은 오차 제곱합에 해당하는 클러스터를 만듦.
3. 반복

In [105…
```
X, y = make_blobs(n_samples =10, n_features = 2, random_state=1)
```

In [107…
```
print(X.shape, y.shape)
```

```
(10, 2) (10,)
```

In [113…
```
from sklearn.cluster import AgglomerativeClustering
aggc = AgglomerativeClustering(n_clusters = None, distance_threshold=0, linkage='complete')
label_aggc = aggc.fit_predict(X)
print(label_aggc)
```

```
[5 7 9 4 6 2 8 3 1 0]
```

In [123…
```
aggc2 = AgglomerativeClustering(n_clusters = 2, linkage='complete')
label_aggc2 = aggc2.fit_predict(X)

print(label_aggc2)
```

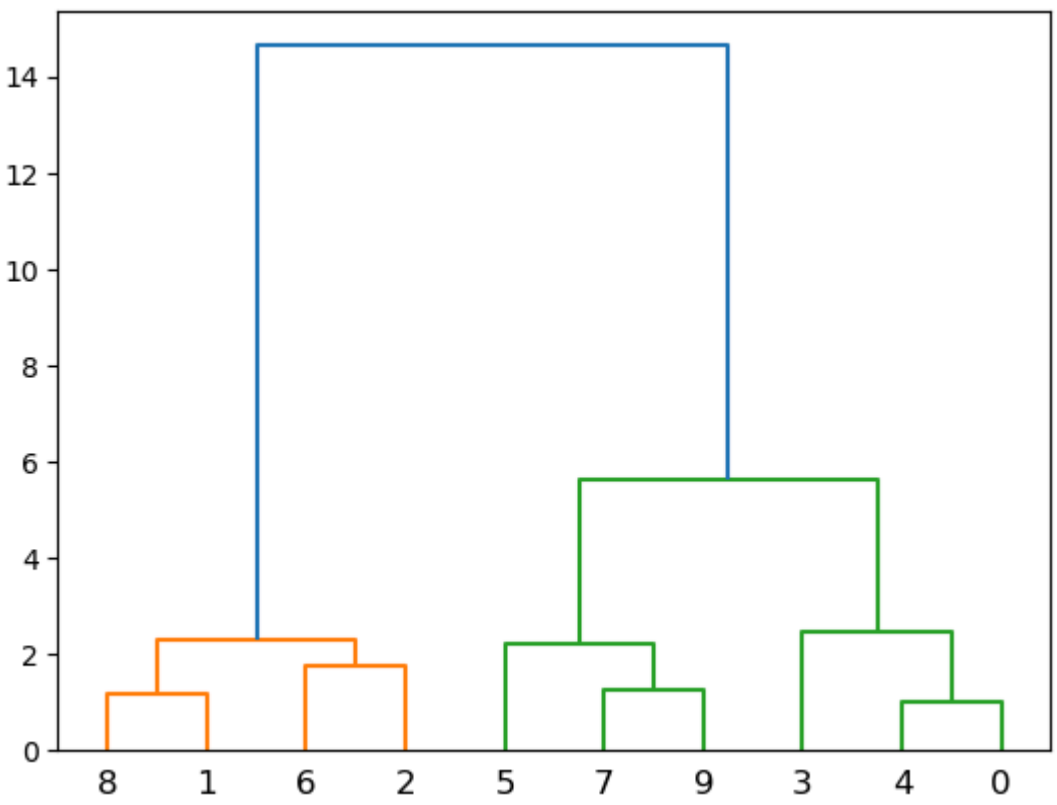
```
[0 0 0 0 1 1 1 0 1 0]
```

In [121…
```
aggc3 = AgglomerativeClustering(n_clusters = 3, linkage='complete')
label_aggc3 = aggc3.fit_predict(X)

print(label_aggc3)
```

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

In [129…
```
from scipy.cluster.hierarchy import dendrogram
from scipy.cluster.hierarchy import linkage
import matplotlib.pyplot as plt

linked = linkage(X, 'complete')
labels = label_aggc
dendrogram(linked, orientation='top', labels=labels, show_leaf_counts=True)
plt.show
```

Out[129…
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



DBSCAN

밀집된 영역 : 클러스터 vs. 밀집 정도가 낮은 영역 : 클러스터 외부 영역 클러스터 개수를 미리 정하지 않음. 포인트간 거리 측정 -> 클러스터의 밀집 정도, 기본:Euclidean

** 반경 내에 외소한의 데이터 포인트 개수 이상이 존재한다. min_samples : 최소한의 데이터 포인트 개수 (hyperparameter) eps : 데이터 포인트 개수를 측정할 반경

eps 내에 min_samples 개수 이상 존재하면 core point. eps 내 존재하는 데이터 포인트들 (border point)을 하나의 클러스터로 분류. 초기 핵심 데이터포인트는 무작위로 선정. eps 내 데이터포인트 개수가 min_samples 이하일 경우 noise로 labeling 함. core point / border point / noise point

1. Eps-neighborhood of a point: Neps(x1)
2. Directly density-reachable: xn이 Neps(x1)에 속하고 Neps(x1)의 개수가 min_samples 이상. 항상 대칭적이지 않다.
3. Density-reachable
4. Density-connected

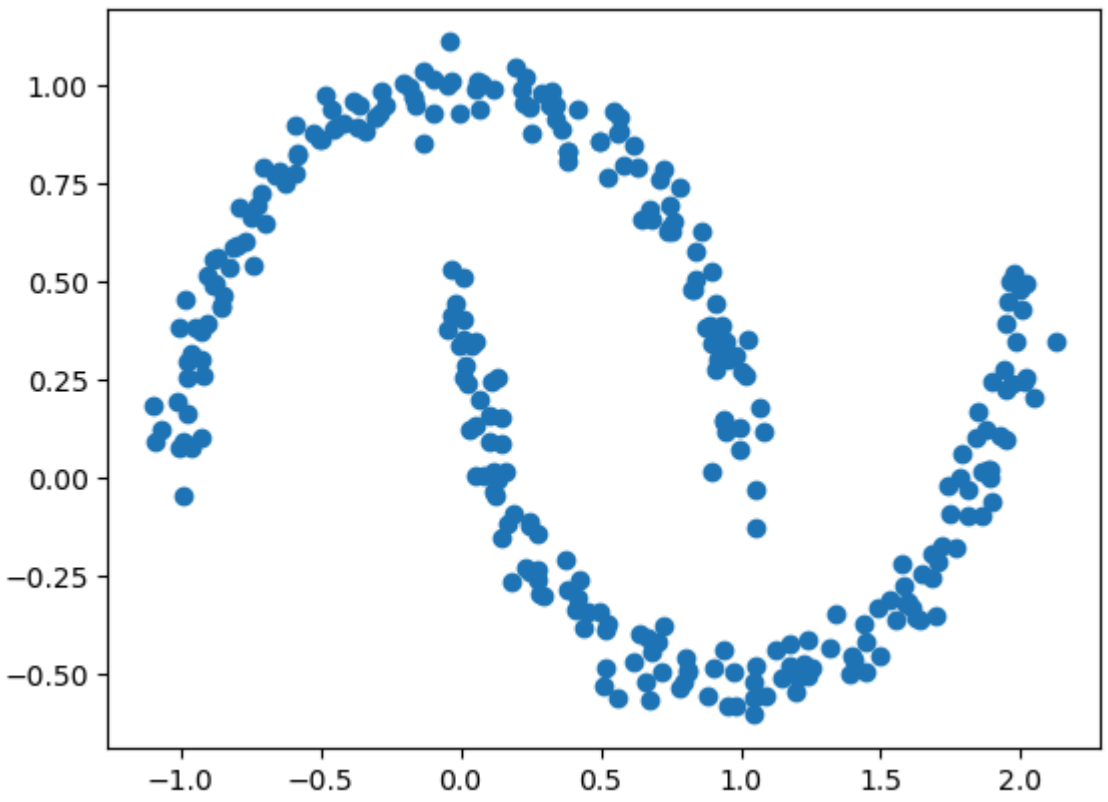
   Cluster C: 데이터포인트 x1이 C에 속할 때 x1에 대해 xn이 density-reachable 하면 xn도 C 에 속하는 것. 이때 x1, xn 은 density connected 하다. Noise: 어떤 C에도 속하지 못한 포인트

In [137…
```
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=300, noise=0.05, random_state=0)
```

In [139…
```
print(X.shape)
print(y.shape)
```

```
(300, 2)
(300,)
```

In [143…
```
plt.scatter(X[:,0], X[:,1])

plt.show()
```



In [145…
```
from sklearn.cluster import DBSCAN
dbs = DBSCAN(eps=0.2)
dbs.fit(X)
label_dbs = dbs.labels_
print(label_dbs)
```

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

In [147…
```
dbs_columns = ['dbs_comp1', 'dbs_comp2']
X_dbs_df = pd.DataFrame(X, columns=dbs_columns)
X_dbs_df['target'] = y
X_dbs_df['label_dbs'] = label_dbs
X_dbs_df.head()
```

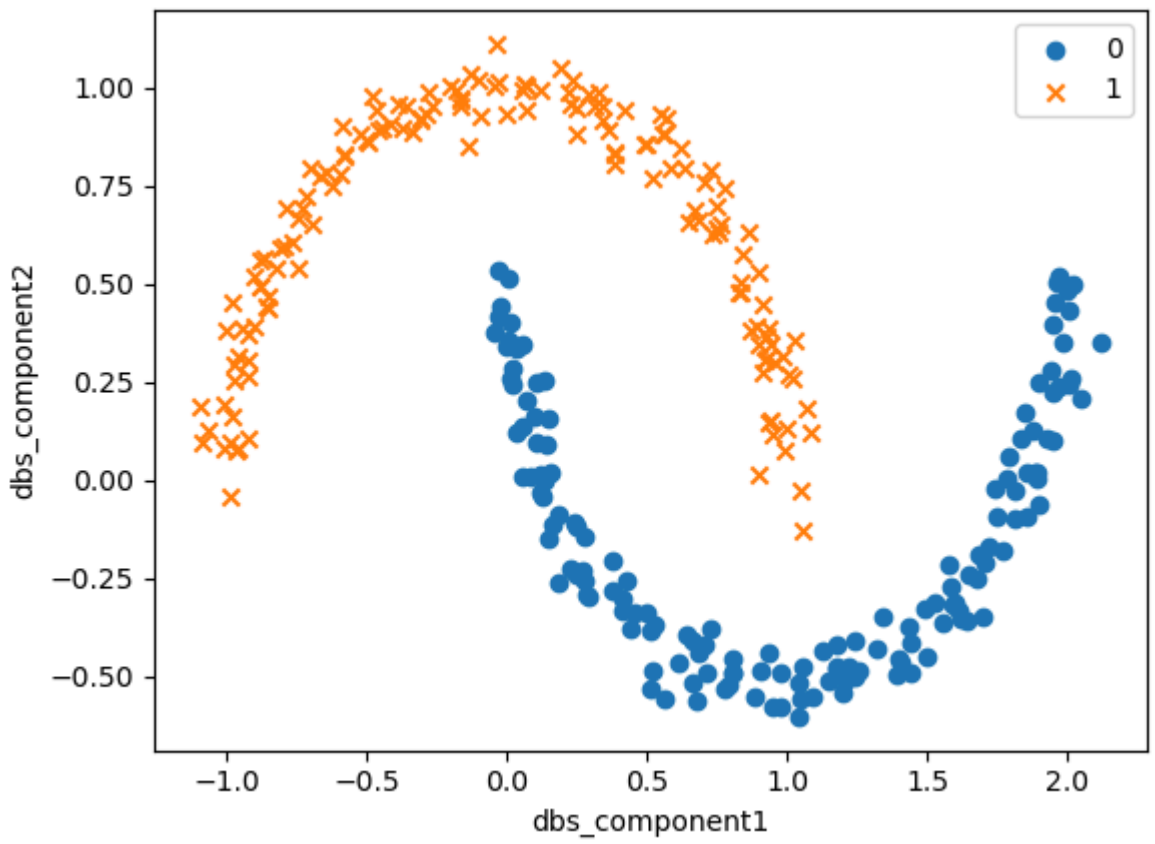
Out[147…

	dbs_comp1	dbs_comp2	target	label_dbs
0	0.715413	-0.494089	1	0
1	0.246789	-0.240091	1	0
2	0.943261	0.346800	0	1
3	0.973742	-0.492901	1	0
4	1.239713	-0.411411	1	0

In [149…
```
print(set(X_dbs_df['target']))
print(set(X_dbs_df['label_dbs']))
```

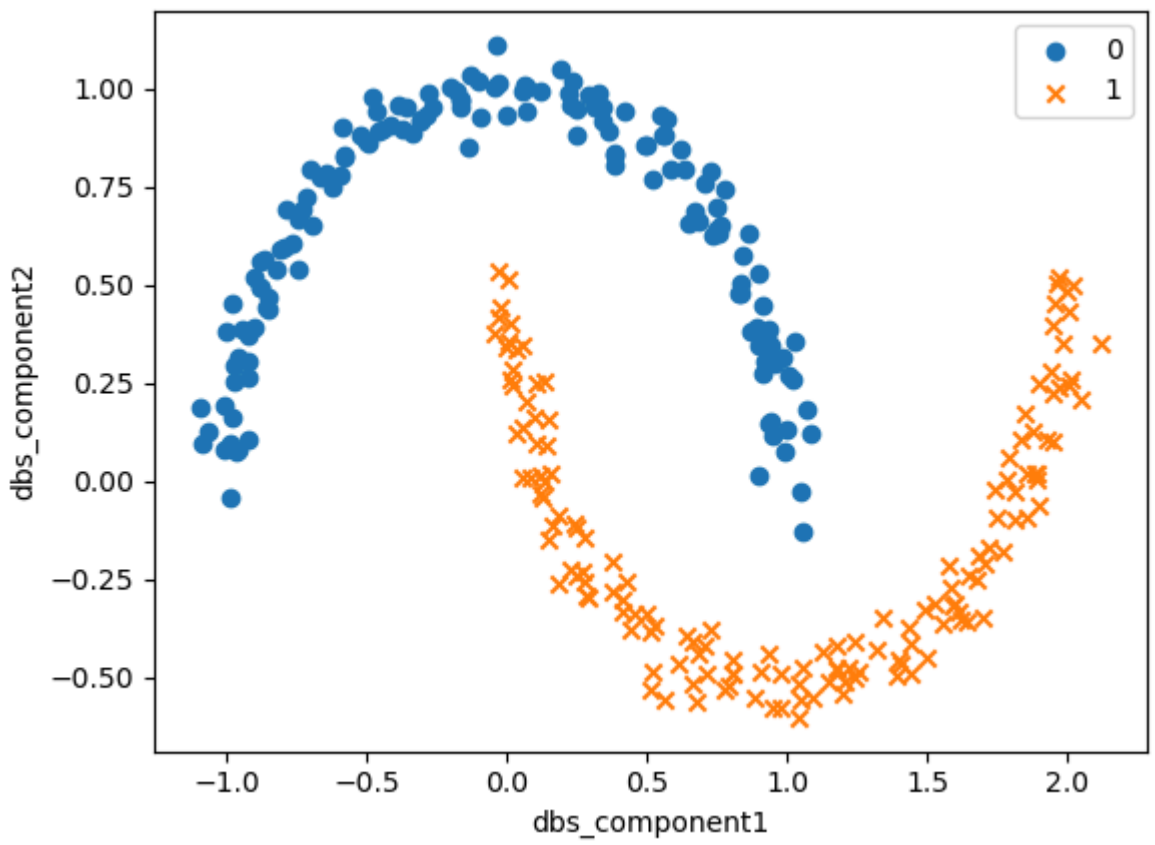
```
{0, 1}
{0, 1}
```

In [155…
```
df = X_dbs_df
markers = ['o', 'x']

for i, mark in enumerate (markers):
    df_i = df[df['label_dbs']==i]
    target_i = i
    X1 = df_i['dbs_comp1']
    X2 = df_i['dbs_comp2']
    plt.scatter(X1, X2, marker=mark, label=target_i)

plt.xlabel('dbs_component1')
plt.ylabel('dbs_component2')

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



In [157…

```
# 실제 타깃
for i, mark in enumerate (markers):
    df_i = df[df['target']==i]
    target_i = i
    X1 = df_i['dbs_comp1']
    X2 = df_i['dbs_comp2']
    plt.scatter(X1, X2, marker=mark, label=target_i)

plt.xlabel('dbs_component1')
plt.ylabel('dbs_component2')

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



In [159…

```
from sklearn.metrics import silhouette_score
sil_score = silhouette_score(X, label_dbs)
print(sil_score)
```

```
0.3284782012631504
```

가우시안 혼합 모형

전체 집단 내부에 속한 하위 집단의 존재를 가정한 확률 모델.

한 하위 집단 당 parameter = (평균μ, 표준편차(다변량Σ/단변량σ), 전체분포에 대한 해당 하위분호의 비율π) 데이터가 특정 집단에 속하는 경우 ~ 잠재변수(latent variable) z (unobservable)

> Parameter를 추정해보자! EM 알고리즘 = E-step & M-step 을 반복하여 로그가능도함수를 증가시키는 방법.
>
> 1. Initialization - μ, σ, π
> 2. E-step: r_ic (i 번째 데이터가 그룹 c 에서 추출되었을 확률, responsibility) 구하기
> 3. M-setp: 구하고 싶었던 parameter 모두를 구함 (μ, σ, π) * 그룹개수, parameter update

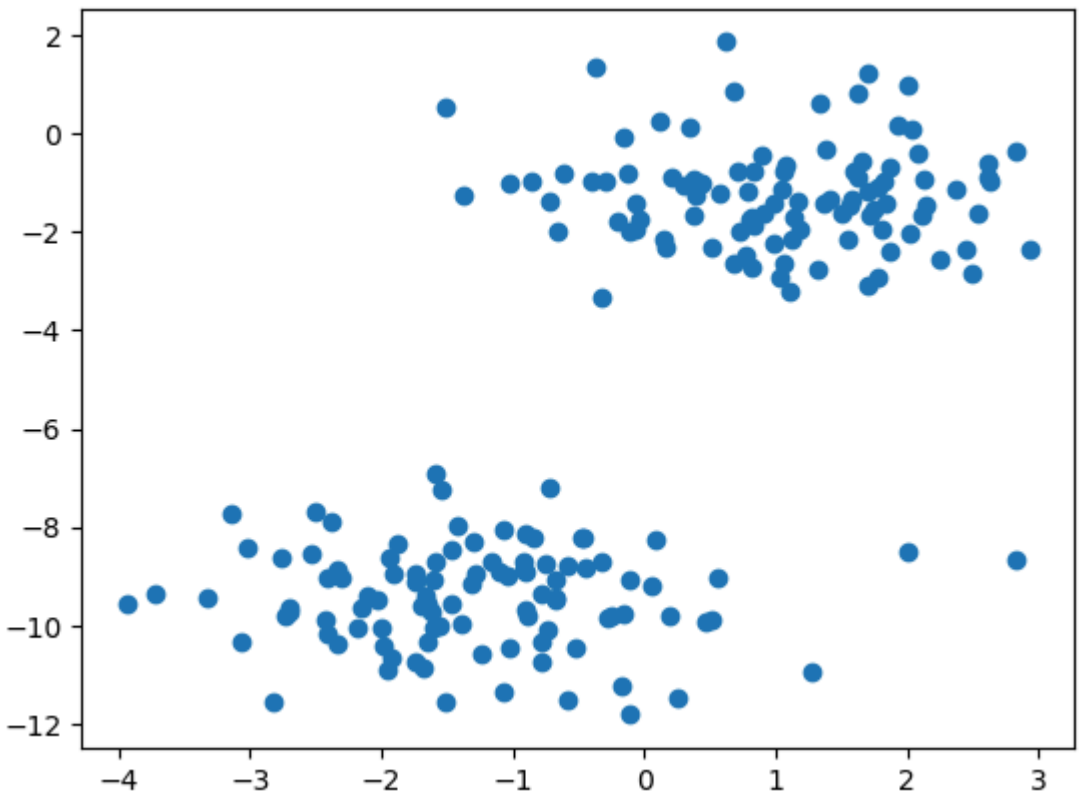
> 단점: 수렴속도가 느리고 local optimization 문제가 발생할 수 있음. K-nn과 마찬가지로 초기에 그룹개수를 정해줘야 함.

In [165…
```
X, y = make_blobs(n_samples=200, n_features=2, centers=2, random_state=2)
```

In [167…
```
print(X.shape, y.shape)
```

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

In [173…
```
plt.scatter(X[:,0], X[:,1])
plt.show()
```



In [177…
```
X
```

Out[177…

```
array([[  0.38115481,  -1.64481461],
       [ -0.58390398,  -8.78405909],
       [  1.83086535,  -1.42523455],
       [  1.81313467,  -1.92467083],
       [ -1.65577138,  -9.55594613],
       [ -0.13265422,  -0.80505548],
       [ -2.69891251,  -9.73324948],
       [  0.18757605,  -9.8171527 ],
       [  1.92513692,   0.17477999],
       [ -0.24707289,  -9.8104778 ],
       [  0.80420551,  -1.74288075],
       [ -1.39452325,  -9.97964956],
       [ -1.60515159, -10.04170987],
       [  0.83352543,  -0.74460559],
       [ -1.46957122,  -9.55869403],
       [  1.55591864,  -1.4580672 ],
       [  1.05937597,  -0.7579938 ],
       [  2.13810493,  -1.44566983],
       [  1.1848232 ,  -1.95576116],
       [ -0.37638912,   1.3767851 ],
       [ -0.6172497 ,  -0.79361238],
       [ -1.68399423, -10.86599403],
       [  0.42186795,  -1.00112008],
       [  0.81825271,  -2.73069558],
       [ -0.16344961,  -0.05951747],
       [  0.67755635,  -2.64355425],
       [ -0.90965742,  -8.1218415 ],
       [ -0.65895073,  -1.96921394],
       [  0.44030012,  -0.99402533],
       [ -0.91504844,  -8.70739333],
       [ -3.01806146,  -8.44065141],
       [ -1.02353151, -10.47025441],
       [  1.59306999,  -0.74416768],
       [ -0.66995787,  -9.05797846],
       [ -2.50153112,  -7.67699872],
       [  1.10320057,  -3.20707537],
       [ -0.58498235, -11.51494191],
       [ -0.4075959 ,  -0.98602662],
       [  1.506501  ,  -1.62839627],
       [ -1.93335222,  -8.63901908],
       [  0.20550002,  -0.86879848],
       [ -2.41442044, -10.18889625],
       [  1.62504848,  -0.87595942],
       [  0.29507856,  -1.05360095],
       [ -1.95777753, -10.92091439],
       [ -1.46335853,  -8.46162063],
       [ -1.42824915,  -7.98003845],
       [ -0.84660563,  -8.20309613],
       [ -0.45064353,  -8.8508534 ],
       [  1.71860282,  -1.65302661],
       [ -1.5841884 ,  -6.8961805 ],
       [ -0.72081785,  -1.37019171],
       [ -2.76256743,  -8.63516347],
       [ -1.55628145,  -9.99835926],
       [  1.02581503,  -2.93411237],
       [  0.33970811,   0.11528184],
       [  0.82608078,  -1.87016308],
       [ -3.93955141,  -9.57292799],
       [  2.12599563,  -0.9232473 ],
       [ -1.2800922 ,  -8.93912279],
       [  1.36979547,  -1.42903191],
       [ -1.06398595, -11.34008775],
       [  0.78454593,  -1.15593416],
       [ -1.50834952, -11.55108763],
       [ -2.34090099,  -8.88980884],
       [  0.88989739,  -0.44520158],
       [ -1.9993558 , -10.06488996],
       [  1.99964344,  -8.51061404],
       [  0.70973004,  -0.75512788],
       [  1.70677401,   1.23618273],
       [  1.8230386 ,  -0.9574862 ],
       [ -1.66161844,  -9.41498635],
       [ -2.30151669,  -9.04907966],
       [  1.77939563,  -1.09938345],
       [ -0.11858047,  -9.09539732],
       [  1.86284776,  -2.38064272],
       [  0.15873435,  -2.29051183],
       [  1.57945366,  -1.31393944],
       [ -0.32431771,  -3.31914574],
       [ -1.91775697, -10.66908765],
       [  0.98564842,  -2.22738281],
       [ -0.78216181,  -9.35497119],
       [ -3.71486953,  -9.36874886],
       [ -1.69941844,  -9.61380426],
       [ -1.03866233,  -8.98404971],
       [  0.82017839,  -1.68889855],
       [  2.82419621,  -0.35337615],
```

```
[ -1.60368203,  -9.05765066],
[  0.62173043,   1.86741382],
[  2.49020135,  -2.82466326],
[  0.55523077,  -9.04078549],
[ -0.45539895,  -8.23326244],
[ -0.74104364, -10.07763506],
[ -1.29923245,  -8.30647414],
[  0.77192861,  -2.46670777],
[ -1.09969215,  -8.92831109],
[  0.91100418,  -1.59821873],
[ -0.06811619,  -1.40206897],
[ -0.10604775,  -1.97508279],
[ -0.77824475, -10.32568907],
[  0.05546421,  -9.17884603],
[  1.86409032,  -0.70074535],
[ -0.75580553,  -8.74619579],
[ -1.97869191, -10.41036729],
[  0.99104311,  -1.43284403],
[ -3.07353754, -10.32322273],
[ -2.02797291,  -9.47245011],
[ -1.54915892,  -7.25010857],
[ -0.85715442,  -0.96306408],
[ -3.14819261,  -7.7502907 ],
[  0.08713347,  -8.26358973],
[  2.54154025,  -1.60938019],
[  1.65747097,  -0.55866748],
[  1.07861399,  -0.62821787],
[  1.13908963,  -1.6936582 ],
[ -2.18234803, -10.07244764],
[  1.40927131,  -1.35139941],
[  1.11435514,  -2.1403809 ],
[  2.07905596,  -0.39137275],
[  2.37703059,  -1.1452029 ],
[  2.02143446,  -2.01543187],
[  0.14325405,  -2.14589394],
[ -0.66876118,  -9.43350477],
[ -2.82677657, -11.5641273 ],
[ -0.31932001,  -8.72310502],
[  0.37487407,  -0.91475768],
[ -0.10657046, -11.82507855],
[ -0.67858614,  -9.47781587],
[  0.39010877,  -1.25159452],
[  1.62861332,   0.80867712],
[  1.05371201,  -2.63214103],
[ -0.52813711, -10.47523635],
[ -1.3196722 ,  -9.15547193],
[  0.51326445,  -2.31320125],
[ -0.89962999,  -9.69861063],
[ -0.89823572,  -8.91519992],
[ -2.53552918,  -8.54331169],
[ -2.73346631,  -9.81949314],
[ -1.03011438,  -1.02557957],
[  2.0119666 ,   1.00882614],
[ -1.07589398,  -8.07477912],
[ -0.20175034,  -1.78148269],
[ -0.03026543,  -1.74172697],
[ -1.74210731,  -9.13058687],
[ -0.17095868, -11.24639309],
[ -1.59361015,  -8.71046363],
[ -2.15820985,  -9.63790953],
[  1.70084242,  -1.17949827],
[ -0.28764562,  -0.98370587],
[ -1.87385754,  -8.3319748 ],
[  1.7408485 ,  -1.52644915],
[ -2.10923725,  -9.39376515],
[  2.61434254,  -0.58086887],
[  1.38354601,  -0.31804274],
[  2.82859067,  -8.66035449],
[ -1.74844822, -10.73619567],
[  1.17331624,  -1.36048319],
[  1.70531079,  -3.09426819],
[ -2.33805418, -10.39048298],
[ -0.48092196,  -8.2188617 ],
[  1.54591325,  -2.13399516],
[  0.68097889,   0.85112594],
[  2.60842583,  -0.86794594],
[ -1.64458105, -10.35745484],
[ -2.41323523,  -9.04838281],
[  2.92640652,  -2.36434847],
[  2.10389484,  -1.66689096],
[ -1.73432981,  -8.96710465],
[ -1.15527831,  -8.72497322],
[  2.24169986,  -2.5392787 ],
[  1.0505005 ,  -1.13446366],
[ -2.70131918,  -9.63497056],
[ -3.32042501,  -9.43521984],
[ -1.61892392,  -9.71765939],
```

```
       [ -0.92096863,  -8.85925495],
       [ -0.8835818 ,  -9.7960928 ],
       [  0.72986148,  -1.97091264],
       [  1.3203166 ,  -2.74891159],
       [ -2.4311049 ,  -9.90799783],
       [  0.57884096,  -1.22623874],
       [  0.46171023,  -9.92897624],
       [ -1.38115313,  -1.23503221],
       [ -0.77722054, -10.72676345],
       [  2.44441946,  -2.35848003],
       [ -0.07381757,  -1.94648329],
       [  1.27033628, -10.95464861],
       [ -0.72864791,  -7.18926735],
       [  2.03872755,   0.07546388],
       [ -1.23856256, -10.59940081],
       [ -2.3788409 ,  -7.89698831],
       [ -1.51220857,   0.53244231],
       [  1.78230921,  -2.91522595],
       [ -0.27973607,  -9.86256788],
       [  0.110124  ,   0.25183468],
       [  2.6351642 ,  -0.98185444],
       [ -1.91478126,  -8.97307912],
       [ -0.16221522,  -9.75571745],
       [  0.5124909 ,  -9.91048868],
       [  1.32915795,   0.61082376],
       [  0.25165836, -11.46732114]])
```

In [179…

```
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=2, random_state=0)
gmm.fit(X)
label_gmm = gmm.predict(X)
print(label_gmm)
```

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

In [181…

```
gmm_columns = ['gmm_comp1', 'gmm_comp2']
X_gmm_df = pd.DataFrame(X, columns=gmm_columns)
X_gmm_df['target'] = y
X_gmm_df['label_gmm'] = label_gmm
X_gmm_df.head()
```

Out[181…

	gmm_comp1	gmm_comp2	target	label_gmm
0	0.381155	-1.644815	1	1
1	-0.583904	-8.784059	0	0
2	1.830865	-1.425235	1	1
3	1.813135	-1.924671	1	1
4	-1.655771	-9.555946	0	0

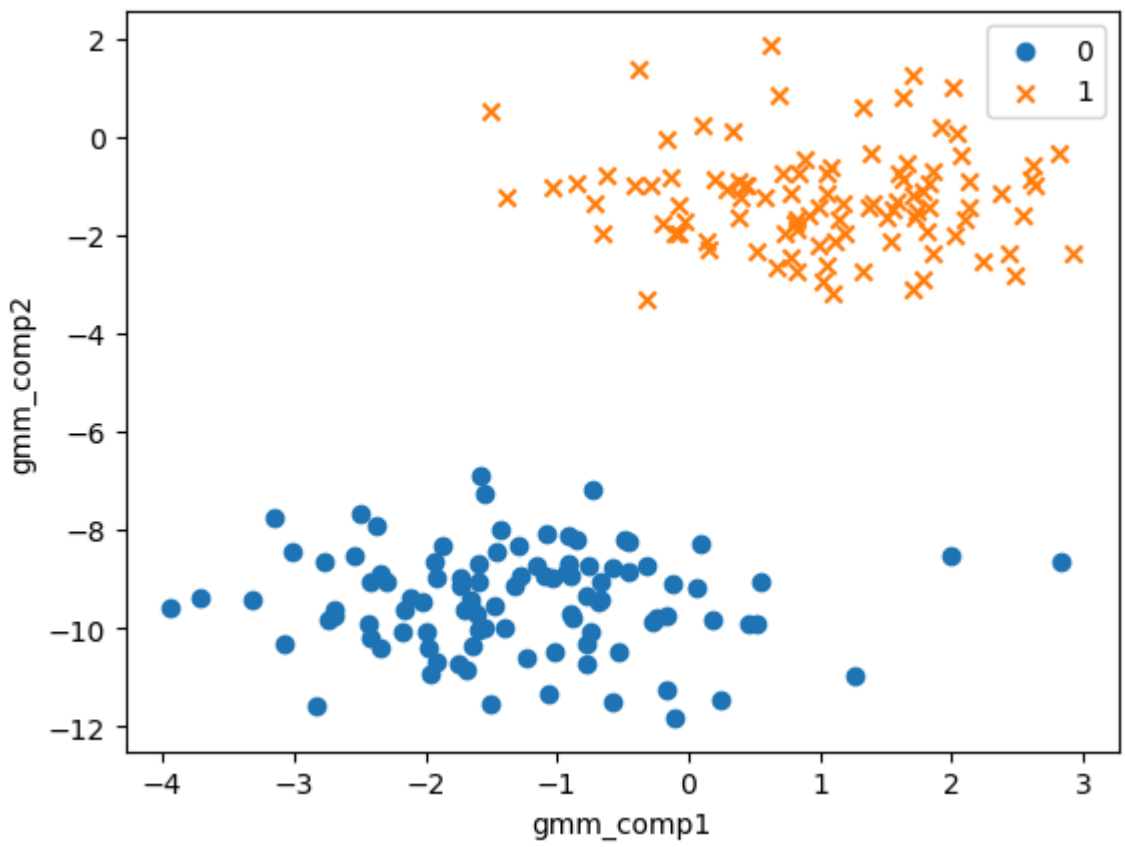
In [183…

```
print(set(X_gmm_df['target']))
print(set(X_gmm_df['label_gmm']))
```

```
{0, 1}
{0, 1}
```

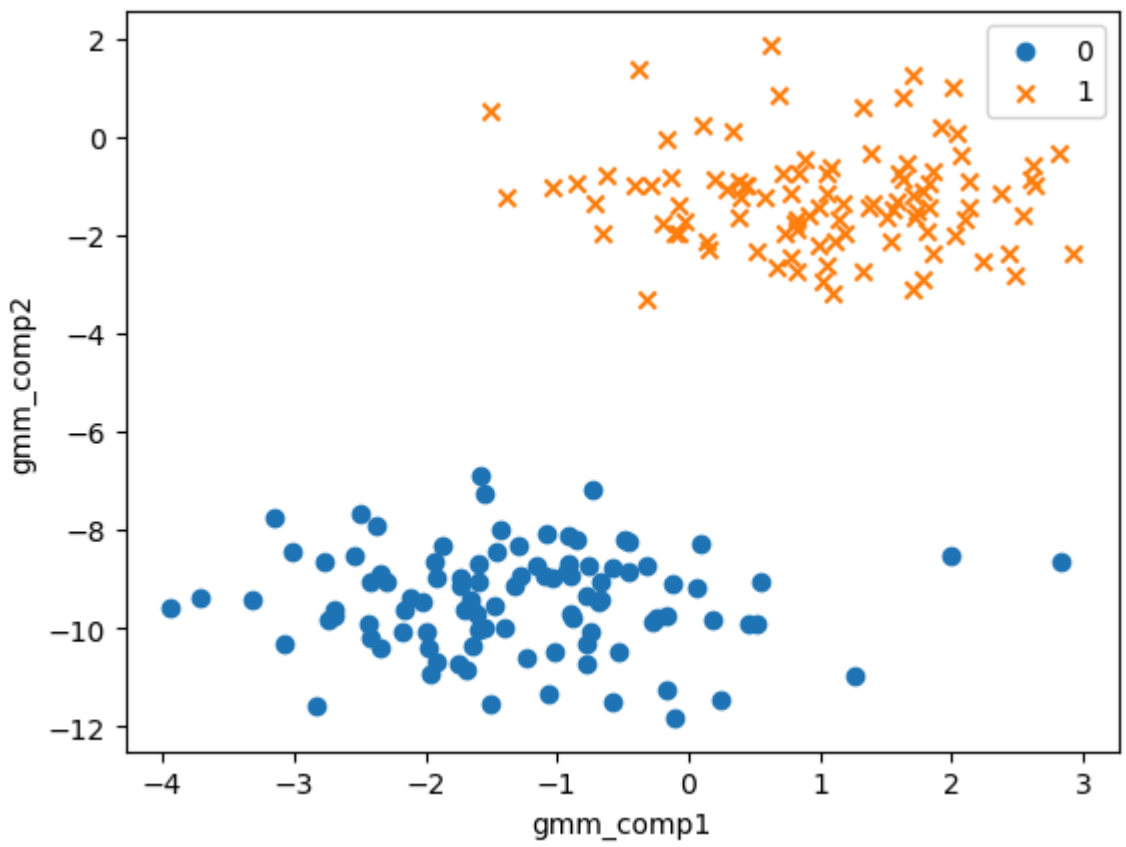
In [187…

```
df = X_gmm_df
markers = ['o', 'x']

for i, mark in enumerate(markers):
    df_i = df[df['label_gmm']==i]
    target_i = i
    X1 = df_i['gmm_comp1']
    X2 = df_i['gmm_comp2']
    plt.scatter(X1, X2, marker = mark, label = target_i)

plt.xlabel('gmm_comp1')
plt.ylabel('gmm_comp2')
plt.legend()
plt.show()
```



In [189…

```
# 실제 target plot
for i, mark in enumerate(markers):
    df_i = df[df['target']==i]
    target_i = i
    X1 = df_i['gmm_comp1']
    X2 = df_i['gmm_comp2']
    plt.scatter(X1, X2, marker = mark, label = target_i)

plt.xlabel('gmm_comp1')
plt.ylabel('gmm_comp2')
plt.legend()
plt.show()
```



In [193…

```
from sklearn.metrics import silhouette_score
sil_score = silhouette_score(X, label_gmm)
print(sil_score)
```

```
0.7842908753561848
```

In [ ]: