비지도 학습(Unsupervised Learning)

비지도 학습은 학습 알고리즘에 결과물이라고 할 수 있는 출력을 미리 제공하지 않고 인공지능(AI)이 입력 세트에서 패턴과 상관관계를 찾아내야 하는 머신러닝 알고리즘이다. 예를 들면, 새로 출시할 상품 유형에 맞는 타겟 시장을 정의하려고 할 때 필요한 기술이다. 비지도 학습은 데이터 자체가 부족하거나 훈련 데이터를 수집하기에는 비용이 너무 높은 등의 이유로 출력에 대해 알 수 없거나 활용할 수 없을 때 주로 사용된다. AI를 어린 아이라고 가정할 때, 지도 학습은 색깔, 숫자 또는 어휘와 같이 인간이 이미 알고 있는 것을 아이에게 가르치는 것과 같다. 비지도 학습은 아이가 스스로 문제를 풀고 추론할 수 있도록 내버려두는 방식이라고 할 수 있다. 보통 아이가 상상을 통한 놀이 또는 글쓰기와 그림 그리기 등의 창의적인 활동을 하면서 스스로 배우도록 한다.

- 정답이 없는 데이터를 통해
- 데이터의 유의미한 패턴 / 구조 발견

대표적인 비지도 학습으로는 **군집화(Clusteering)**라는게 있습니다. 군집화는 레이블이 지정되지 않은 데이터를 유사점 또는 차이점에 따라 그룹화하는 데이터 마이닝 기술입니다. 군집화 알고리즘은 분류되지 않은 원시 데이터 객체를 정보의 구조 또는 패턴으로 표현되는 그룹으로 처리하는 데 사용됩니다. 군집화 알고리즘은 몇 가지 유형, 특히 배타적, 중첩적, 계층적 및 확률적 유형으로 분류할 수 있습니다.
ex) 고객 세분화, 소셜 네트워크 분석, 기사 그룹 분류,...
참고로 클러스터링은 우리가 지도학습에서 배웠던 클래시피케이션(분류)와는 성격이 다릅니다. 뷴류는 정답이 있고, 군집화는 정답이 없습니다.

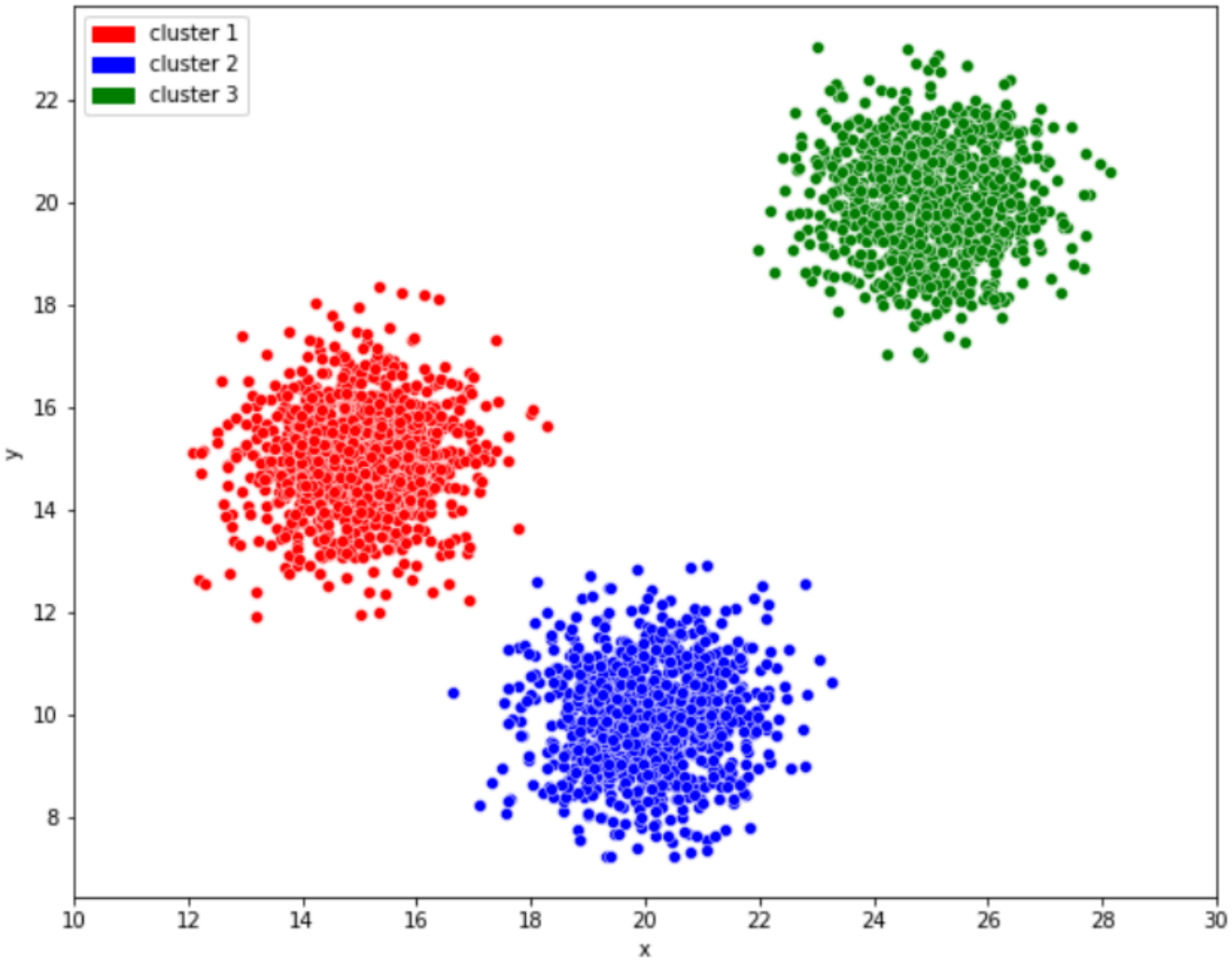
5. K-Means

비지도 합습의 클러스터리ㅏㅇ에서 대표적으로 사용되는 알고리즘이 바로 K-Means, 우리말로는 K-평균 알고리즘입니다.
K-평균 군집화 K개의 군집(Cluster)으로 묶는(Clusting) 알고리즘이다.

군집이란 쉽게 말해서 비슷한 특성을 지닌 데이터들을 모아놓은 그룹(Group)이다. 마찬가지로 군집화는 군집으로 묶는다는 의미로 해석할 수 있다.

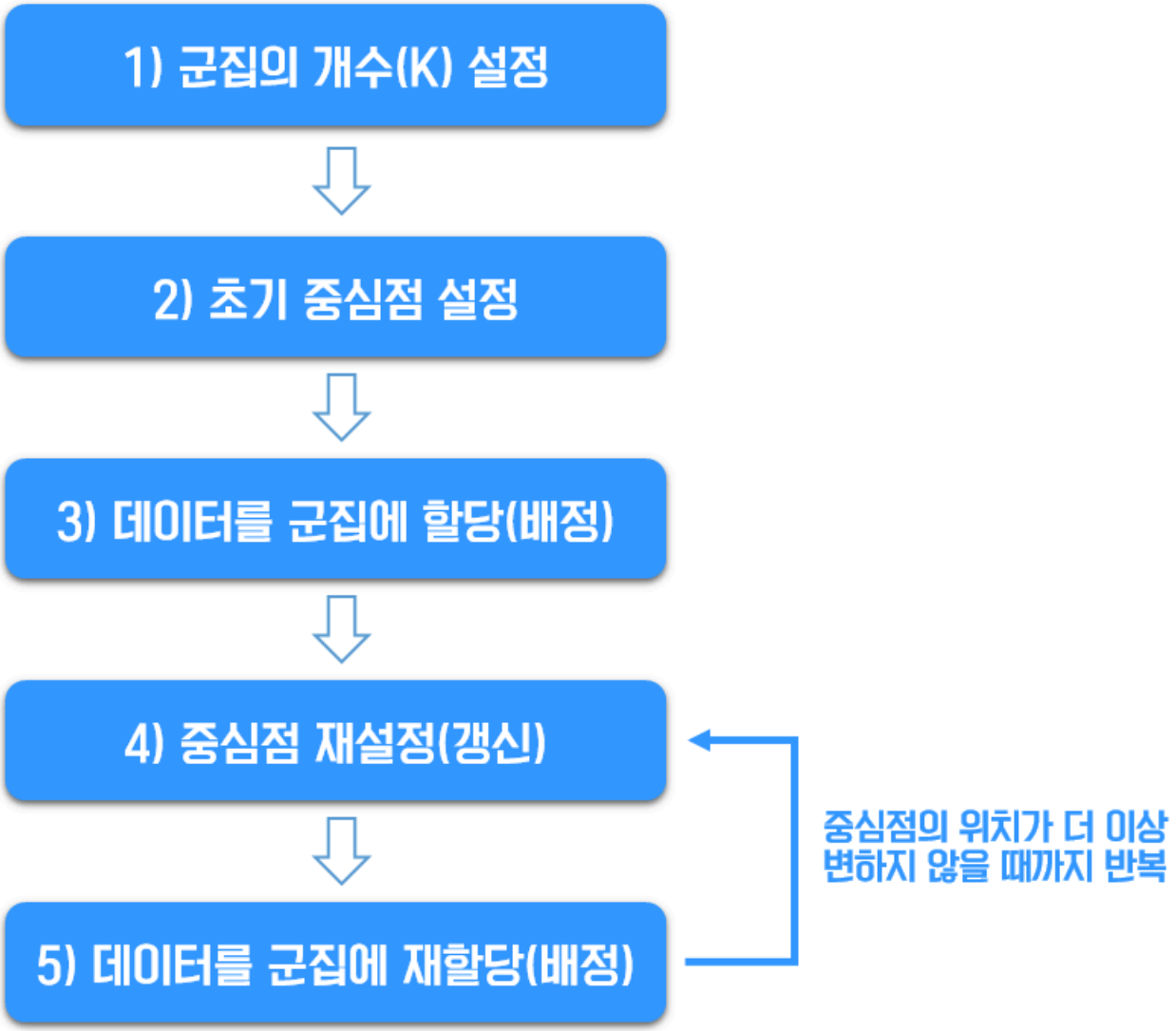
K-means 알고리즘에서 K는 묶을 군집(클러스터)의 개수를 의미하고 means는 평균을 의미한다. 단어 그대로의 의미를 해석해보면 각 군집의 평균(mean)을 활용하여 K개의 군집으로 묶는다는 의미다. 여기서 평균(Means)이란 각 클러스터의 중심점(Centroid)과 데이터들의 평균 거리를 의미한다.

아래의 그림이 주어졌다고 했을 때 어떻게 군집화를 할 수 있을까?
대부분의 사람이라면 아래의 그림과 같이 3개의 군집으로 묶을 것이며 이는 K-means 알고리즘에서 K가 3인 경우에 속한다.



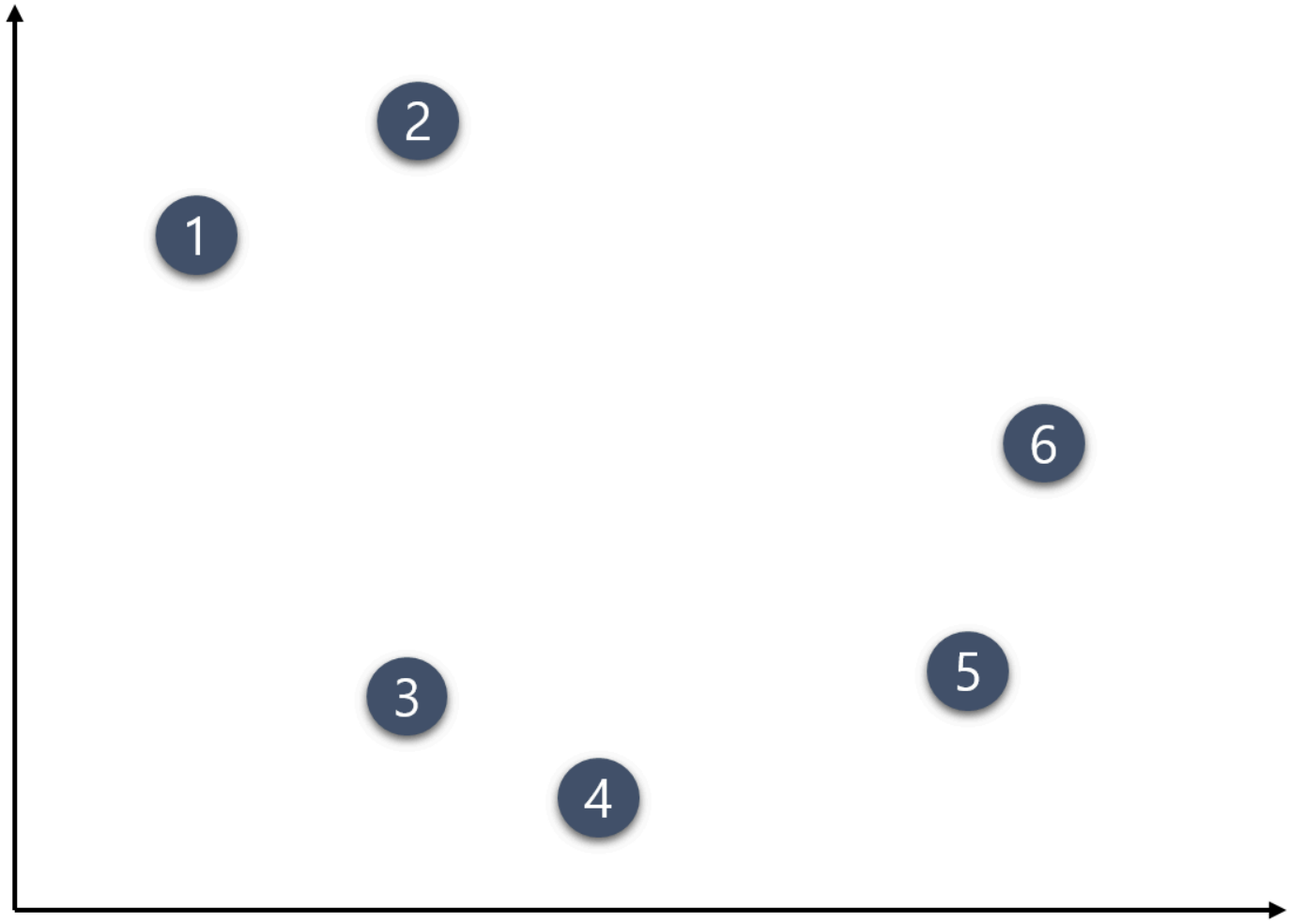
다시 말해, K-means 알고리즘은 비슷한 특성을 지닌 데이터들끼리 묶어 K개의 군집으로 군집화하는 대표적인 군집화 기법이다. 하지만 비슷한 특성을 지닌 데이터들끼리 묶는 방법은 여러가지가 있을 수 있다.

K-Means 동작 순서



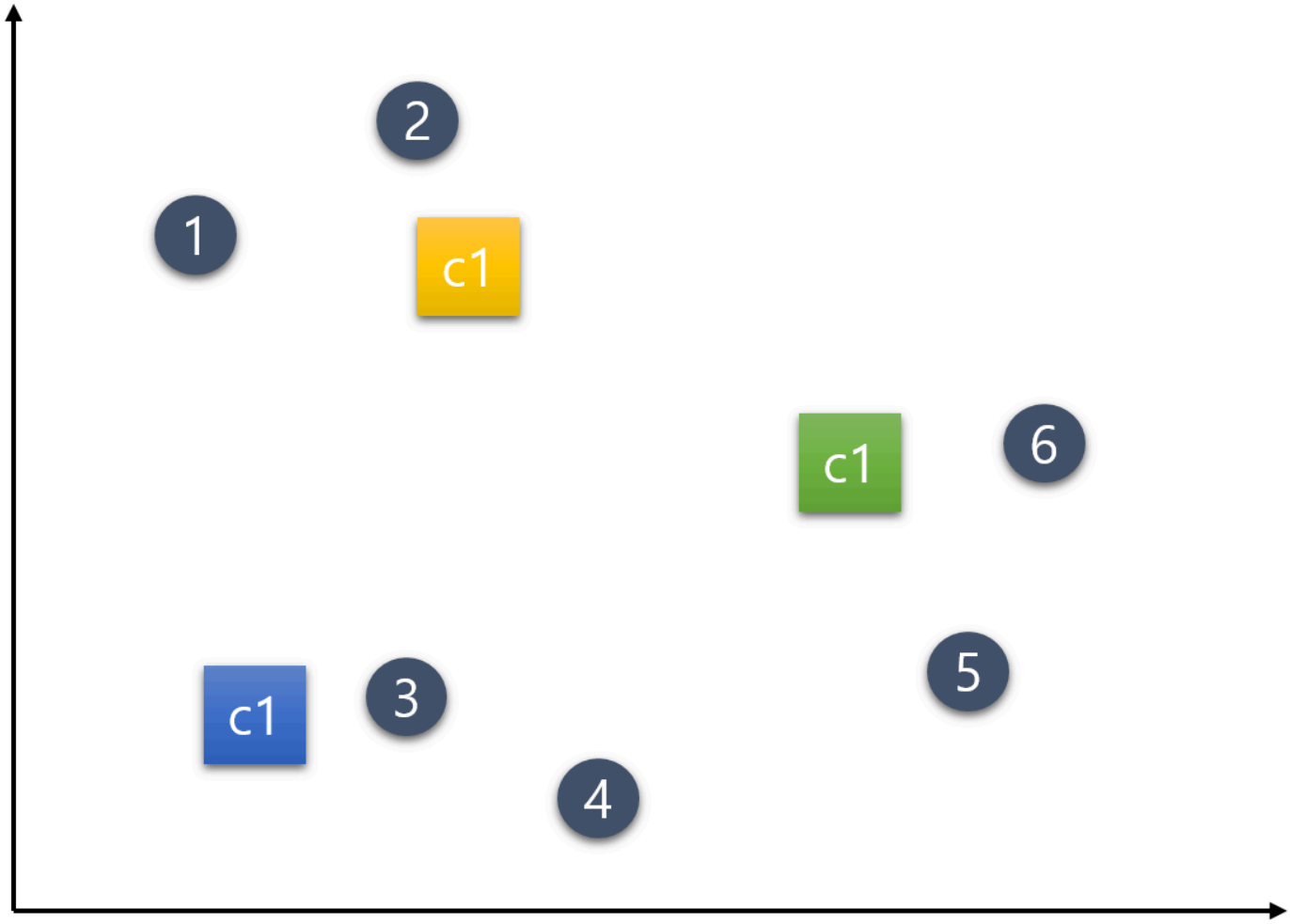
K-means 알고리즘 원리 확인하기 앞서 살펴본 K-means 알고리즘의 원리를 단계별로 이해한 뒤 이제 실제 그림으로 각 단계를 확인해보자.

다음과 같이 6개의 데이터가 주어졌다고 하자. 이 주어진 데이터를 K-means 알고리즘의 작동 과정에 따라 군집화 해보자.



step 1) 군집의 개수(K) 설정하기 군집의 개수는 K는 임의대로 3으로 설정하기로 하자.

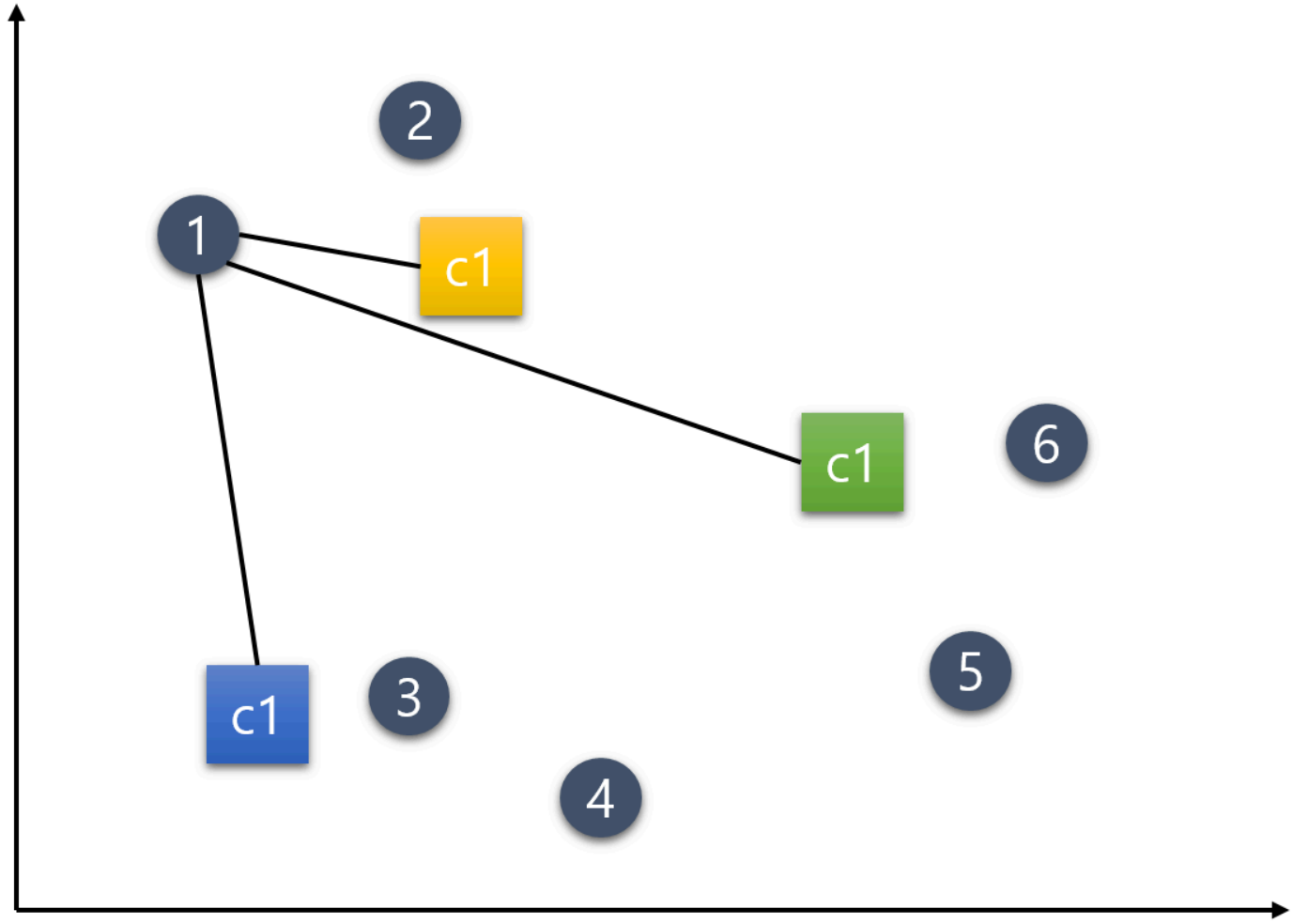
step 2) 초기 중심점 설정하기 여러가지 설정 방법이 있지만 여기서는 편의상 초기 중심점 (Centroid) c1, c2, c3는 다음과 같이 랜덤으로 설정한다.



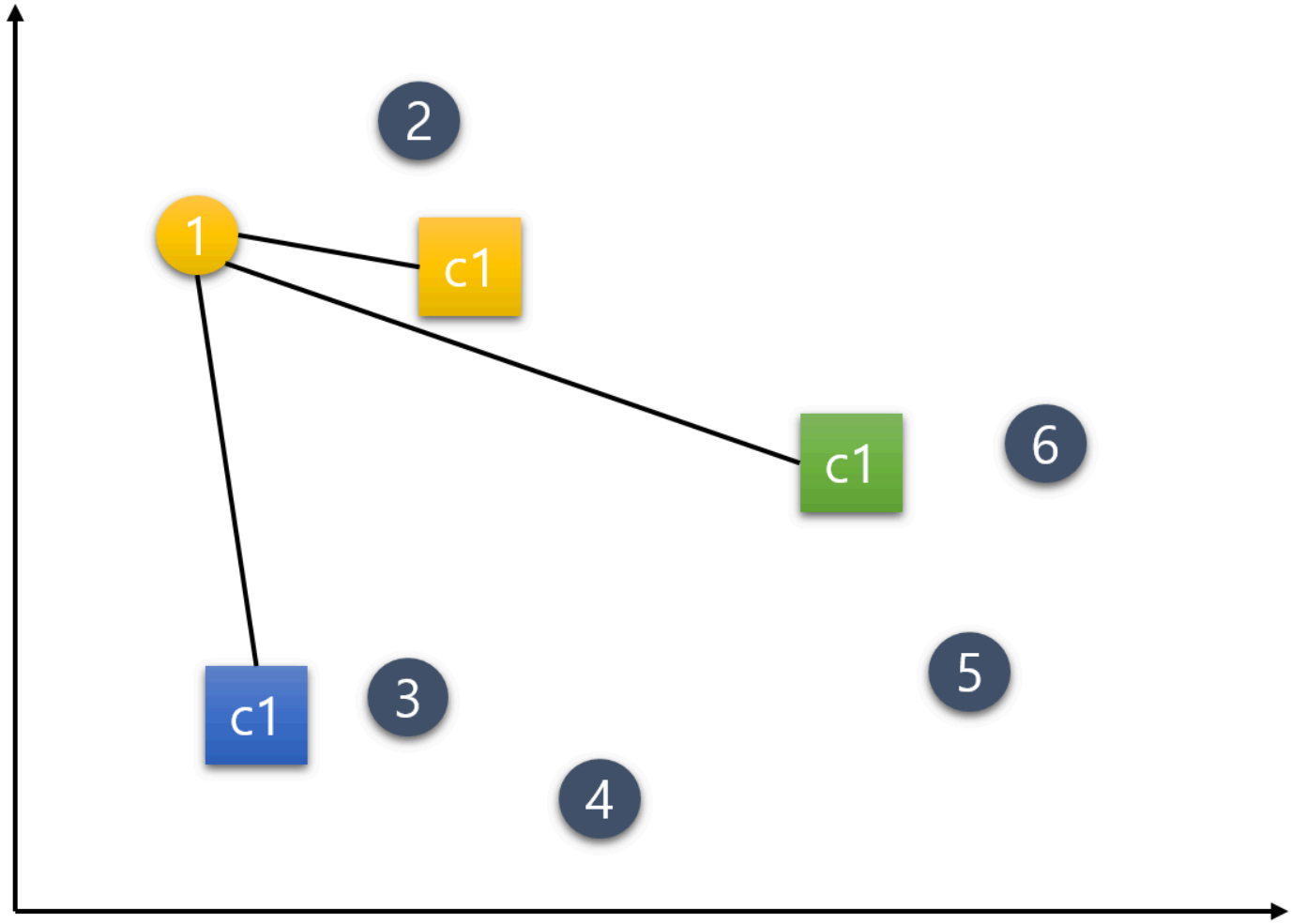
step 3) 데이터를 군집에 할당(배정)하기 거리 상 가장 가까운 군집(중심점)으로 주어진 데이터를 할당 또는 배정하는 단계이며 거리 측정 방법은 일반적으로 유클리드 거리로 측정한다.

1번 데이터부터 시작해서 6번 데이터까지 각각의 중심점에 가까운 군집으로 배정한다.

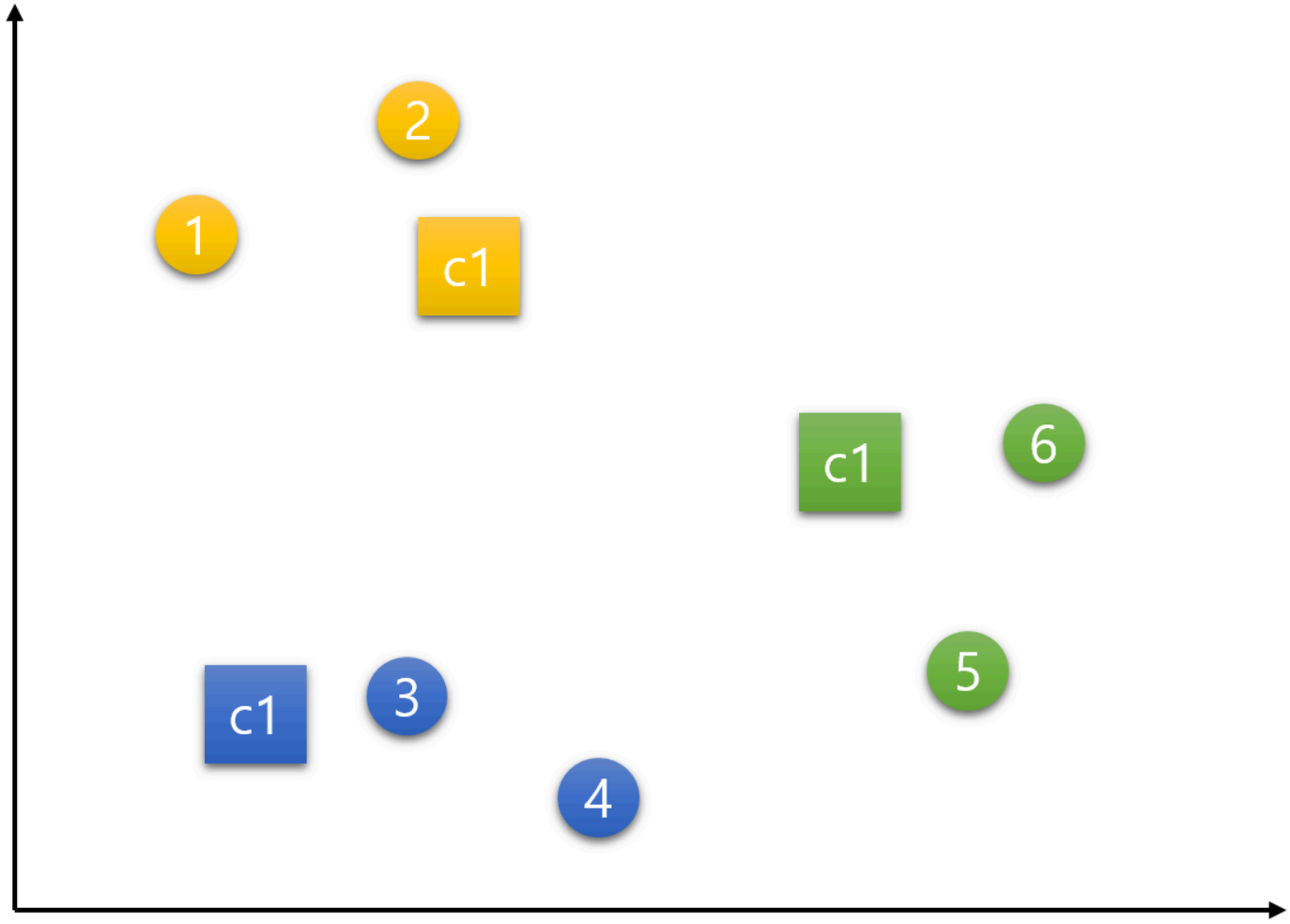
1번 데이터. 아래의 그림과 같이 c1, c2, c3 중심점으로부터 1번의 유클리드 거리를 측정한다.



가장 가까이 위치하는 중심점의 군집으로 배정한다. 1번 데이터의 경우 c1 중심점과 가장 가까우므로 노란색으로 바뀐다.

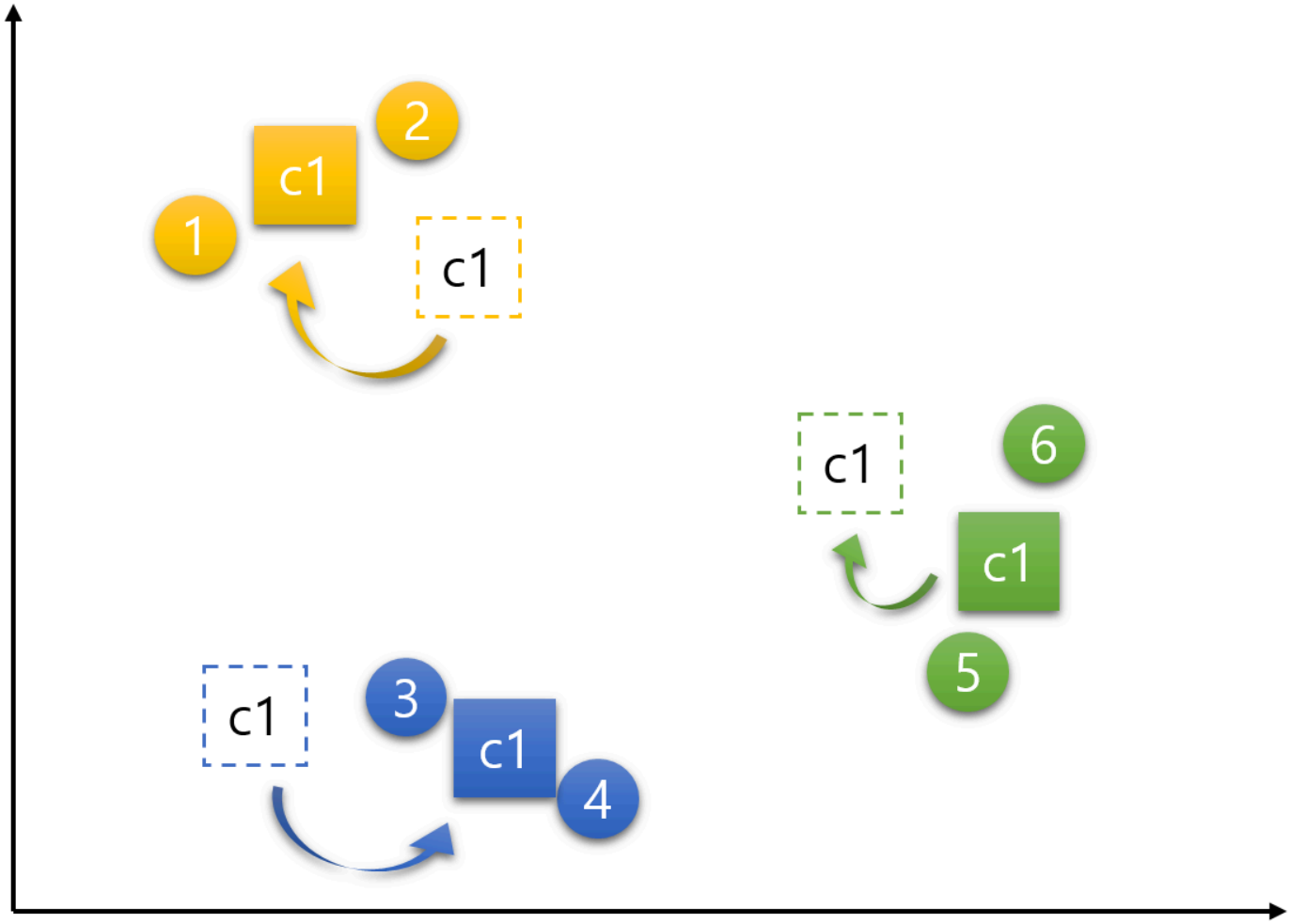


1번 외 나머지 데이터들도 같은 방식으로 배정한다. 이제 모든 데이터들이 한 번씩 각각의 군집들로 배정이 되었다.

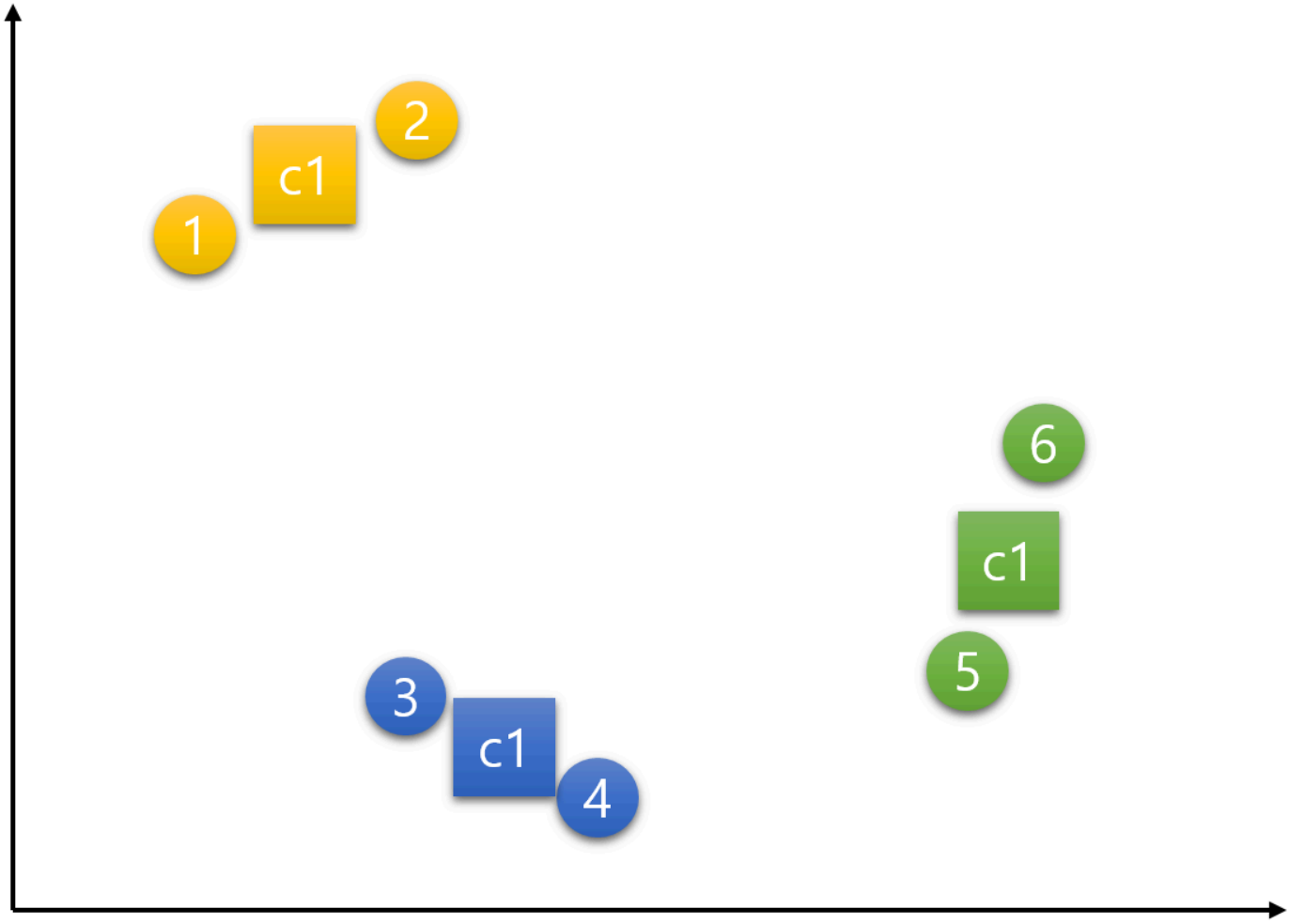


step 4) 중심점 재설정(갱신)하기 이제 중심점을 재설정하는 단계이다.

c1, c2, c3 각각의 중심점은 그 군집의 속하는 데이터들의 가장 중간(평균)에 위치한 지점으로 재설정한다. 중심점 c1은 데이터 1, 2의 평균인 지점으로, 중심점 c2는 데이터 3, 4의 평균인 지점으로, 중심점 c3는 데이터 5, 6의 평균인 지점으로 갱신된다. 결국 다음과 같이 중심점들이 옮겨진다.

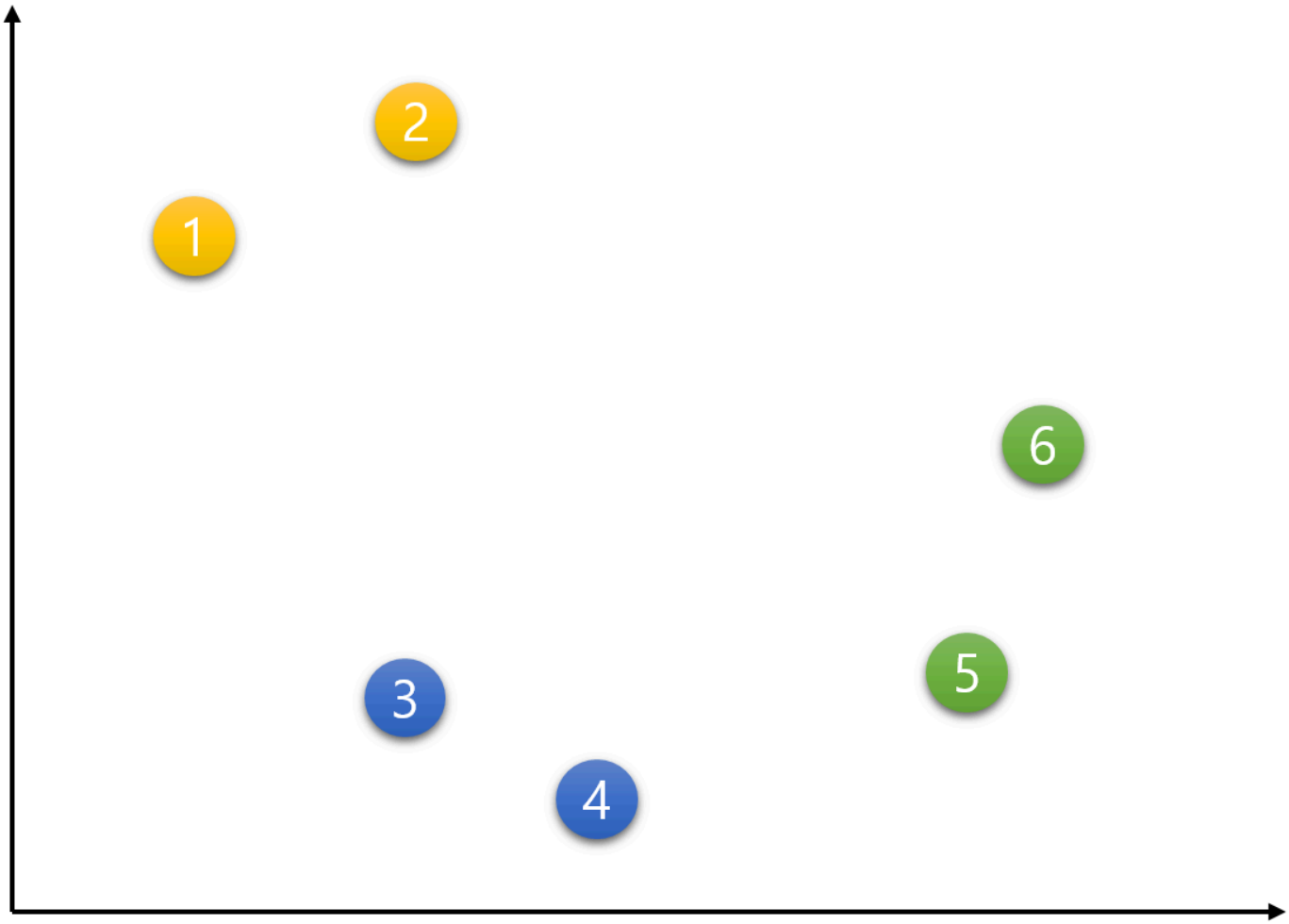


결국 다음과 같이 중심점들이 옮겨진다.



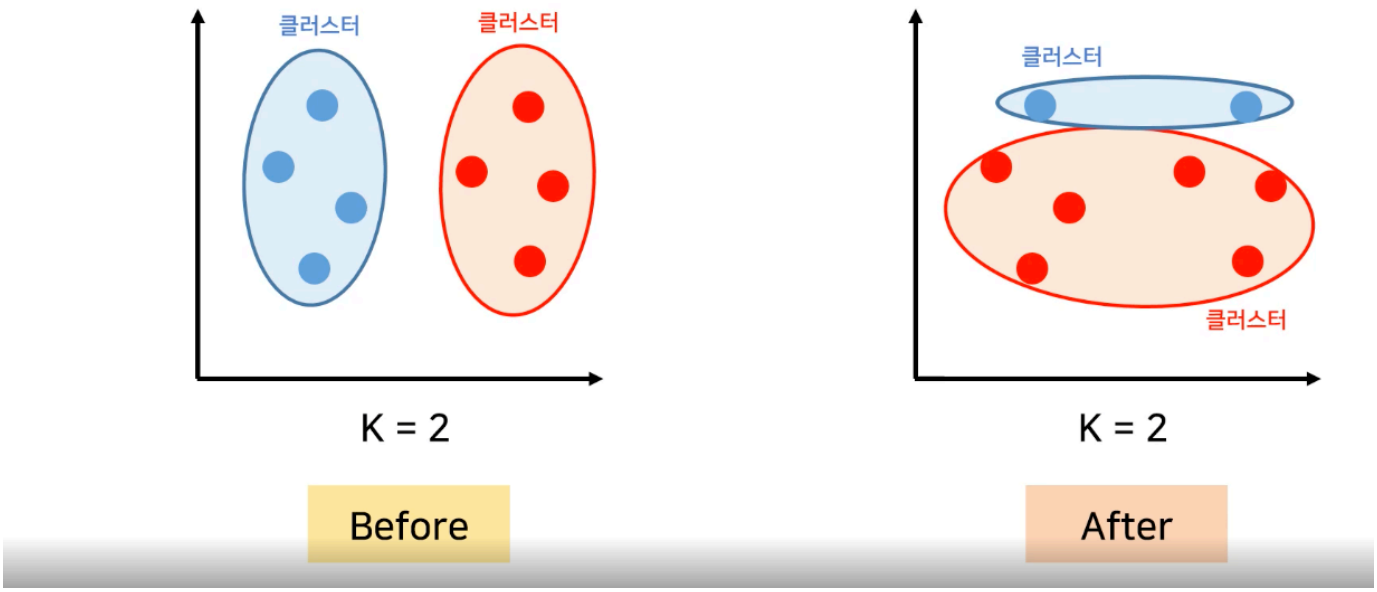
이 세 점이 무게중심을 의미하는 Centroid 라고 불리는 이유는 데이터들의 무게중심의 위치로 이동하기 때문임을 알 수 있다.

step 5) 데이터를 군집에 재할당(배정)하기 더 이상 중심점의 이동이 없을 때까지 step 4와 step 5를 반복한다고 했는데 위의 예시는 더 이상 이동이 없어 종료된다. 즉, 아래의 그림이 최종 결과이다.



그런데 사실은 K-Means의 동작 중에 하나의 큰 문제점이 있는데 바로 두번째 순서인 랜덤 좌표 설정입니다. K-Means는 최초 설정된 자표들, 즉 센트로이드들이 아주 민감한데 이값이 아무런 제한 없이 그냥 랜덤으로 설정되다 보니까 매번 결과가 달라지는 원치 않는 결과가 나올수 있다. 또는 각 센트로이드 사이의 거리가 짧으면 클러스터링이 제대로 이루어지지 않을 수도 있습니다. 이걸 **Random Initialization Trap**((중심점) 무작위 선정 문제)라고 한다. 중심점을 잘못 잡으면 아래 그림과 같이 전혀 다른 군집화 결과를 초래할 수 있다.

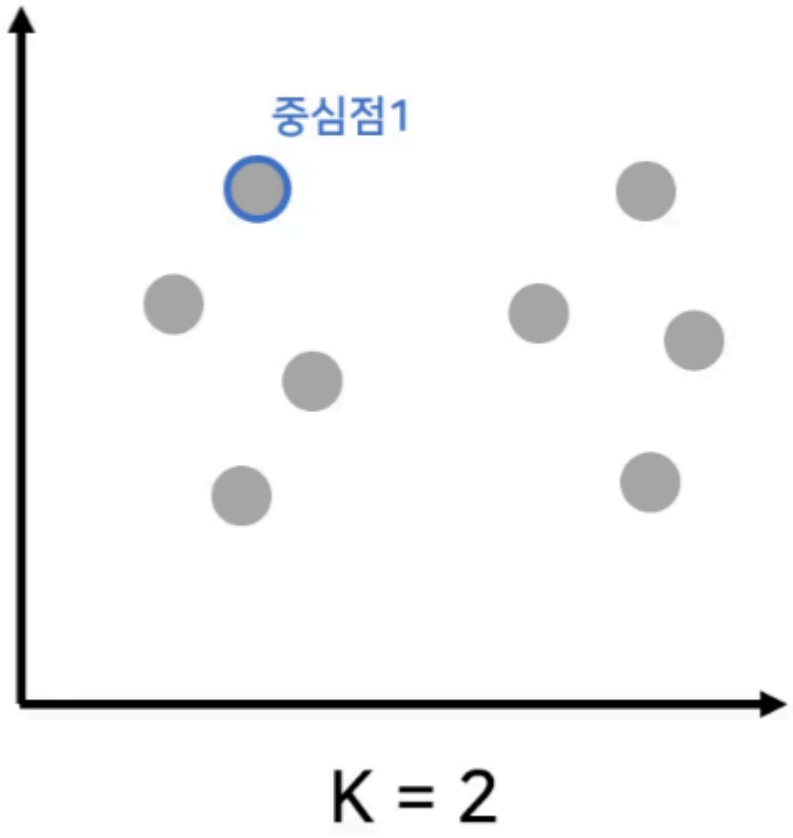
Random Initialization Trap



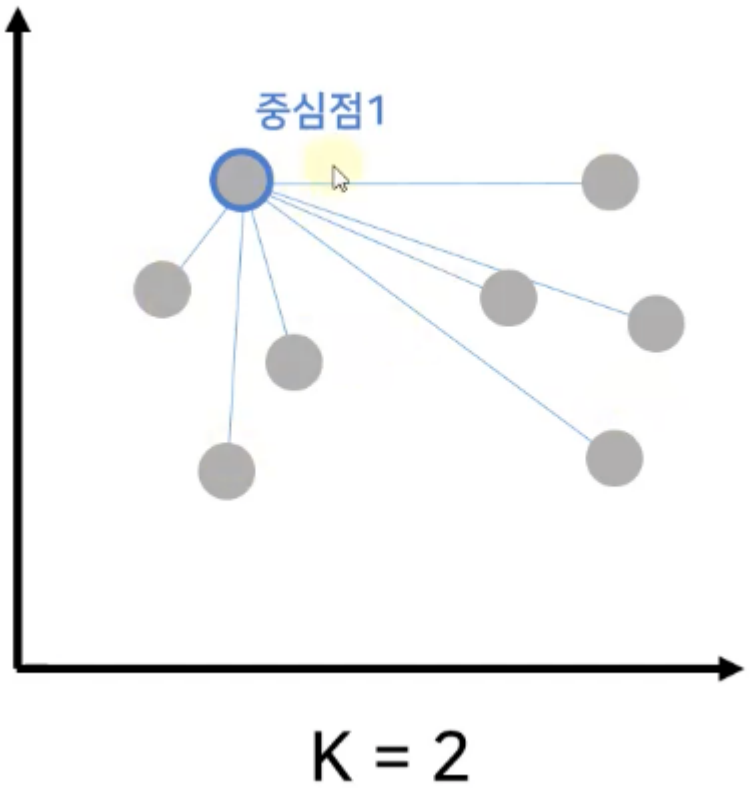
위와 같은 문제를 해결하기 위하여, K-Means++ 입니다.

K-Means++

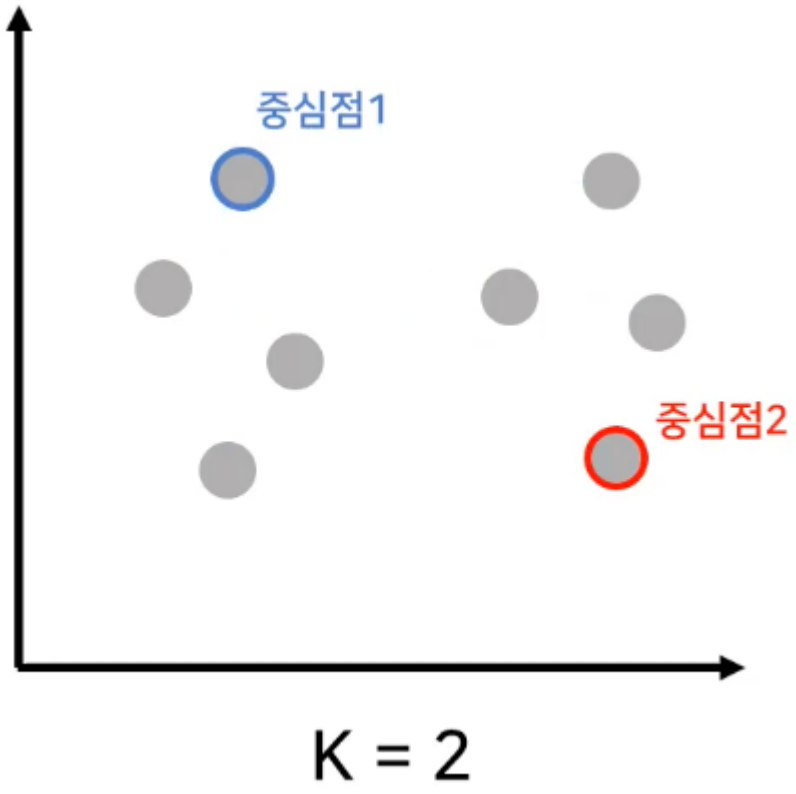
1. k계의 ㅈ붕심점을 바로 선택하지 않고 데이터 중에서 랜덤으로 1개를 중심점으로 선택해서 그 점을 첫번째 중심점으로 지정한다. 즉 아래 그림의 왼쪽위에 있는 데이터를 중심점으로 정해준다.



2. 나머지 데이터로부터 중심점까지의 거리 계산



3. 이 중심점과 가장 먼 지점의 데이터를 다음 중심점으로 선택

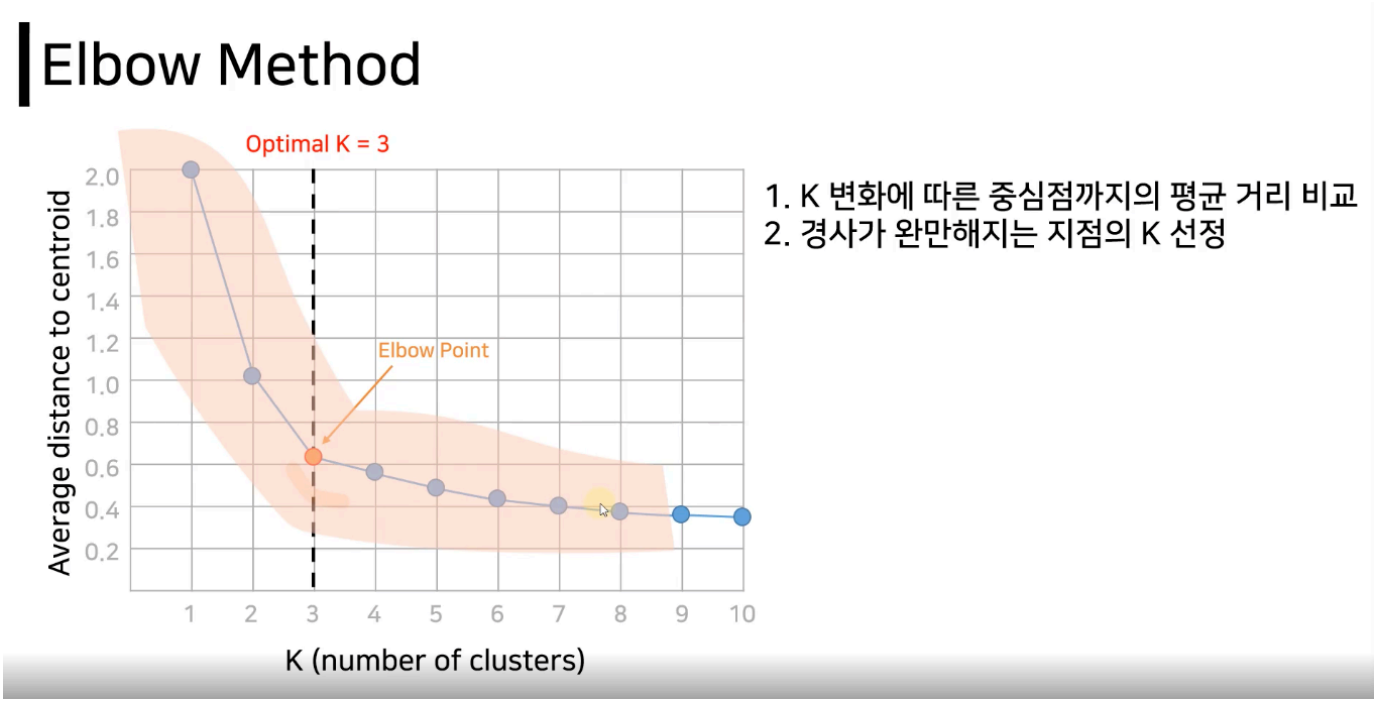


4.중심점이 K개가 될 때까지 반복

5.4번까지 완료되고 난 후, 앞에서의 K-Means 전통적인 방식으로 진행. 중심점 이동

우리가 가용할 Cyclone 패키지에서는 K-Means를 사용할 때 디폴트로 옵션을 K-Means++방식으로 사용하도록 되어 있기 때문에 코드에서 특별히 추가로 해줘야 하는 부분은 없습니다.

그러면 우리가 최적의 K 값을 지정해 줘야 하는데 정확한 K값을 지정하기에 힘들다. 그래서 이때 우리는 **Elbow Method**(엘보우 방법)을 이용해서 최적의 K를 판단하는데 도움을 받는다.



1. K 변화에 따른 중심점까지의 평균 거리 비교
2. 경사가 완만해지는 지점의 K 선정 (Optimal K = 3)

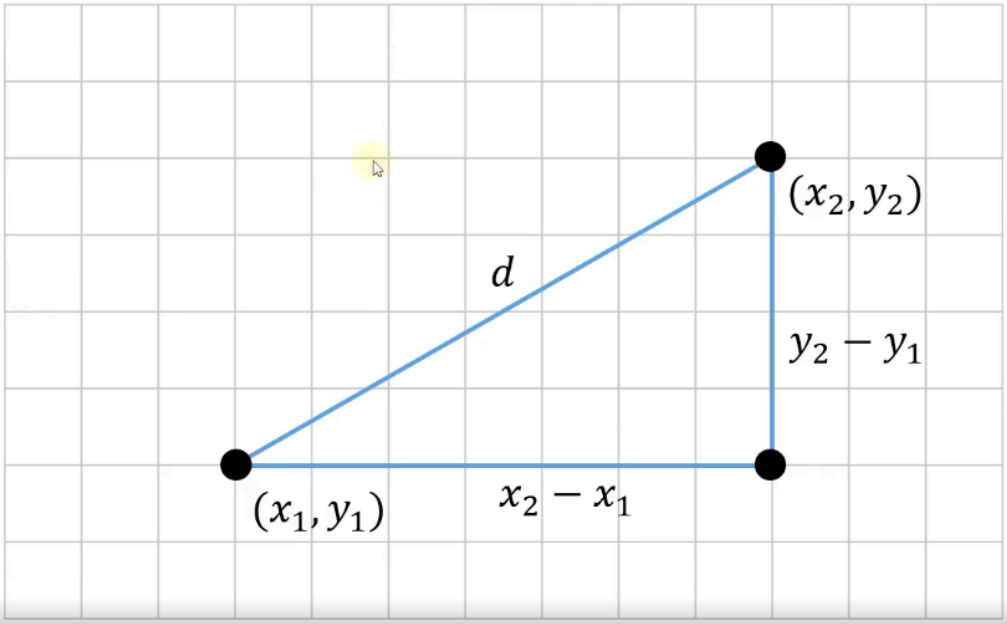
- 너무 많지 않은 cluster의 개수이면서, 평균 거리의 값은 어느 정도 작은 상태

이때의 모양이 이렇게 팔꿈치를 닮았다고 해서 이름이 엘보 메소드입니다.

데이터 유사도를 비교하기 위한, 몇 가지 방법 첫번째가 k-mEANS에서 이용했던 거리제기 방식인 유클리드 거리 (Euclidean Distance)입니다. - 두지점의 거리

Euclidean Distance

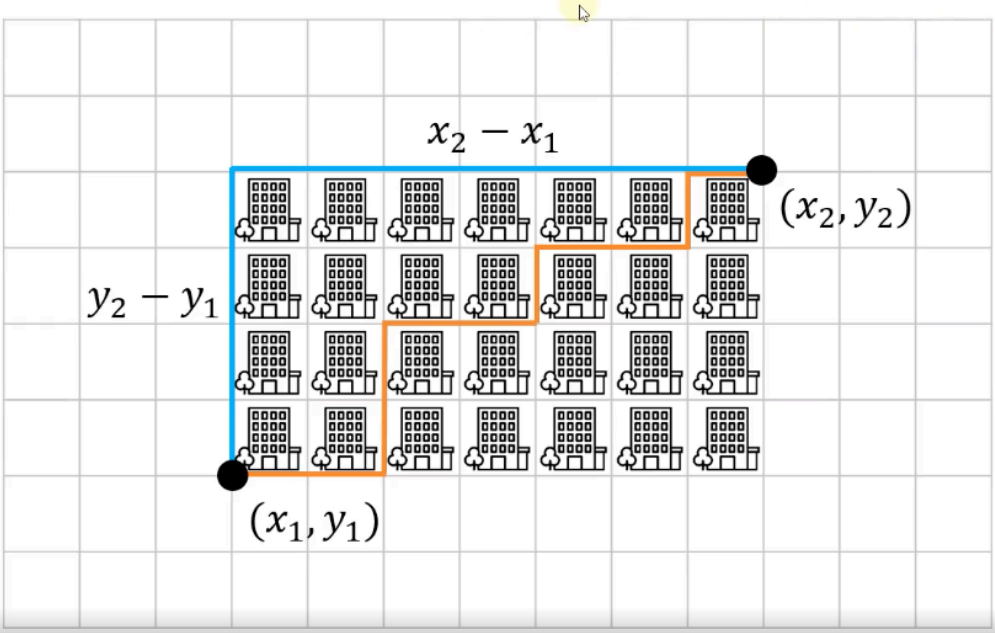
$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$



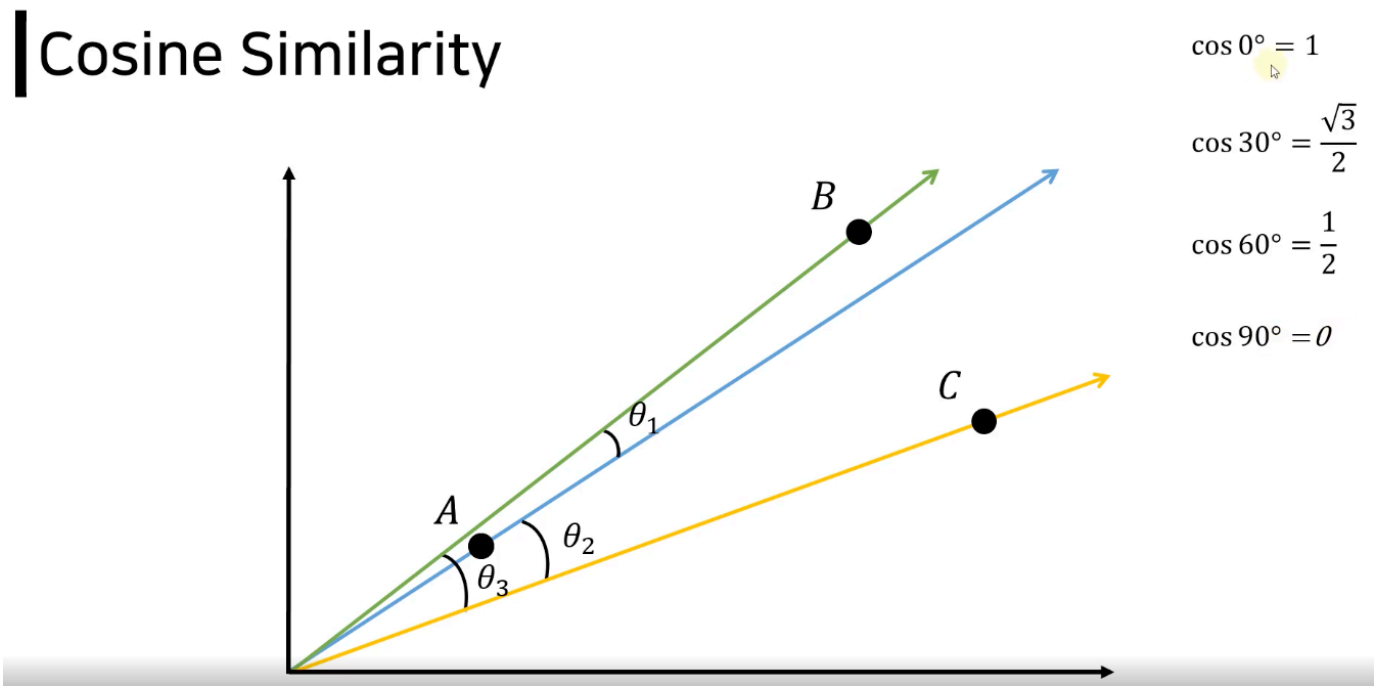
또 다른 방식은 맨해튼 거리 (Manhattan Distance)가 있습니다. - 건물 사이를 걸어가는 거리, 주황색과 파란색은 거리는 항상 같다.

Manhattan Distance

$$d = |x_2 - x_1| + |y_2 - y_1|$$



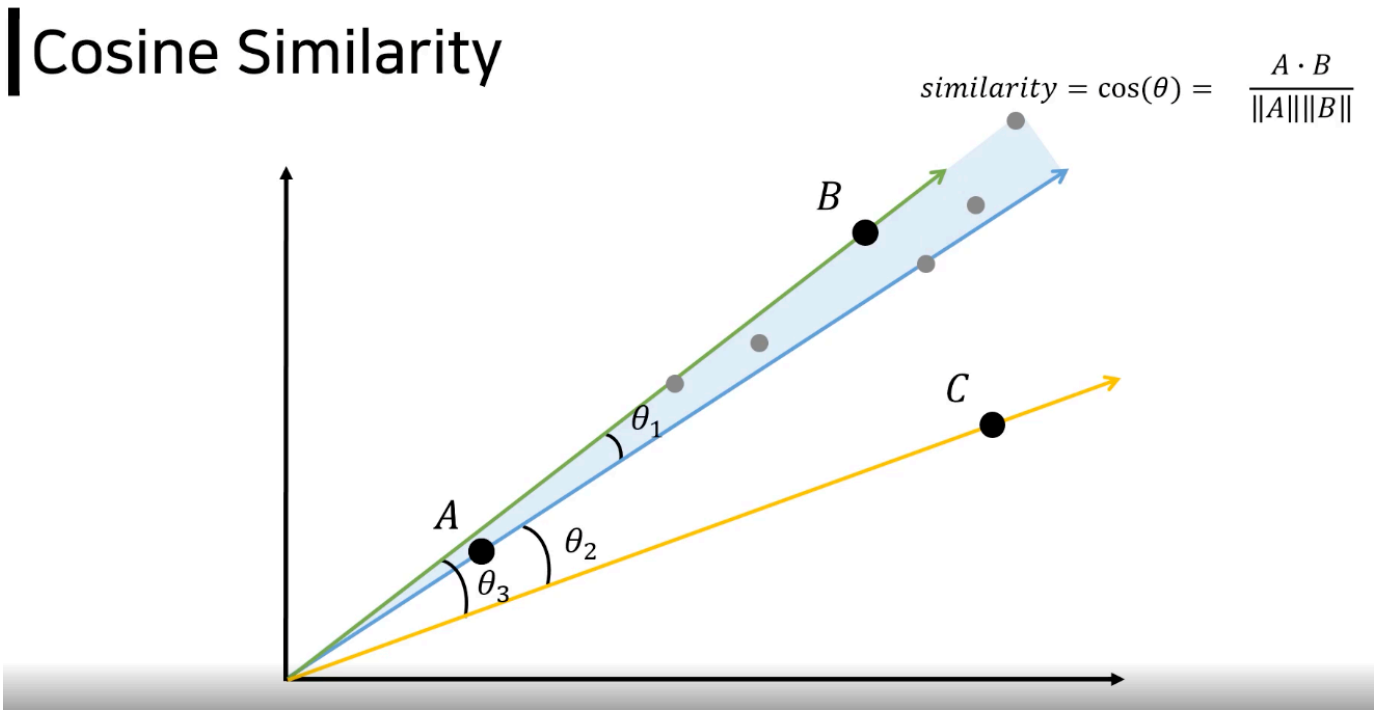
그리고 Cosine Similarity(코사인 유사도) 가 있습니다. 그림에 나오는 코사인의 값을 보면 0동 일때 1이고 90도일 때 0값이 나오는 것으로 보면 각이 작을 수로 값이 커지는 것을 볼수 있다.



코사인은 각도가 작을수록 실제 값이 커진다.

- 따라서, 코사인 유사도에서도 각도가 작을수록 유사도가 더 높다.

아래 그림처럼 색칠한 부분의 면적안에 있는 데이터들은 A와 B 사이의 각도들 보다 더 작기 때문에 이 데이터들은 모두 유ㅜ사도가 상당히 높다고 판단 할 수 있다.



- 영역 안의 있는 값들은 유사도가 상당히 높은 값들이다.

```
import os # 경고 대응
os.environ['OMP_NUM_THREADS'] = '1' # 스레드 갯수를 한개로 정해준다는 것 넘파이 보다 먼저 실해해이
```

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

```
from google.colab import drive # 구글드라이버에 있는 파일 사용하기
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/c
```

```
dataset = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/활용편7-머신러닝/ScikitLearn/KMeans[
dataset[:5]
```

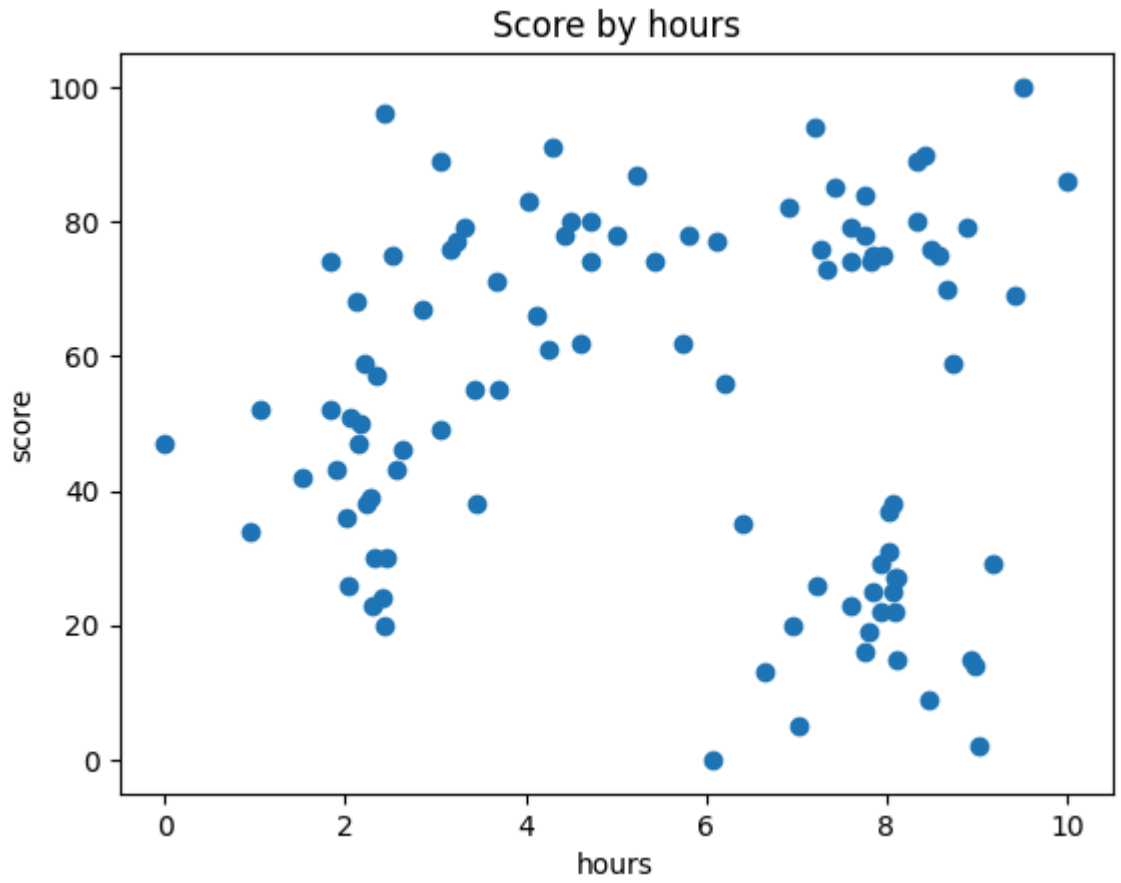
	hour	score
0	7.33	73
1	3.71	55
2	3.43	55
3	3.06	89
4	3.33	79

```
X = dataset.iloc[:, :].values
# X = dataset.values # 이렇게 해도 됨.
# X = dataset.to_numpy() # 공식 홈페이지 권장
X[:5]
```

```
array([[ 7.33, 73.  ],
       [ 3.71, 55.  ],
       [ 3.43, 55.  ],
       [ 3.06, 89.  ],
       [ 3.33, 79.  ]])
```

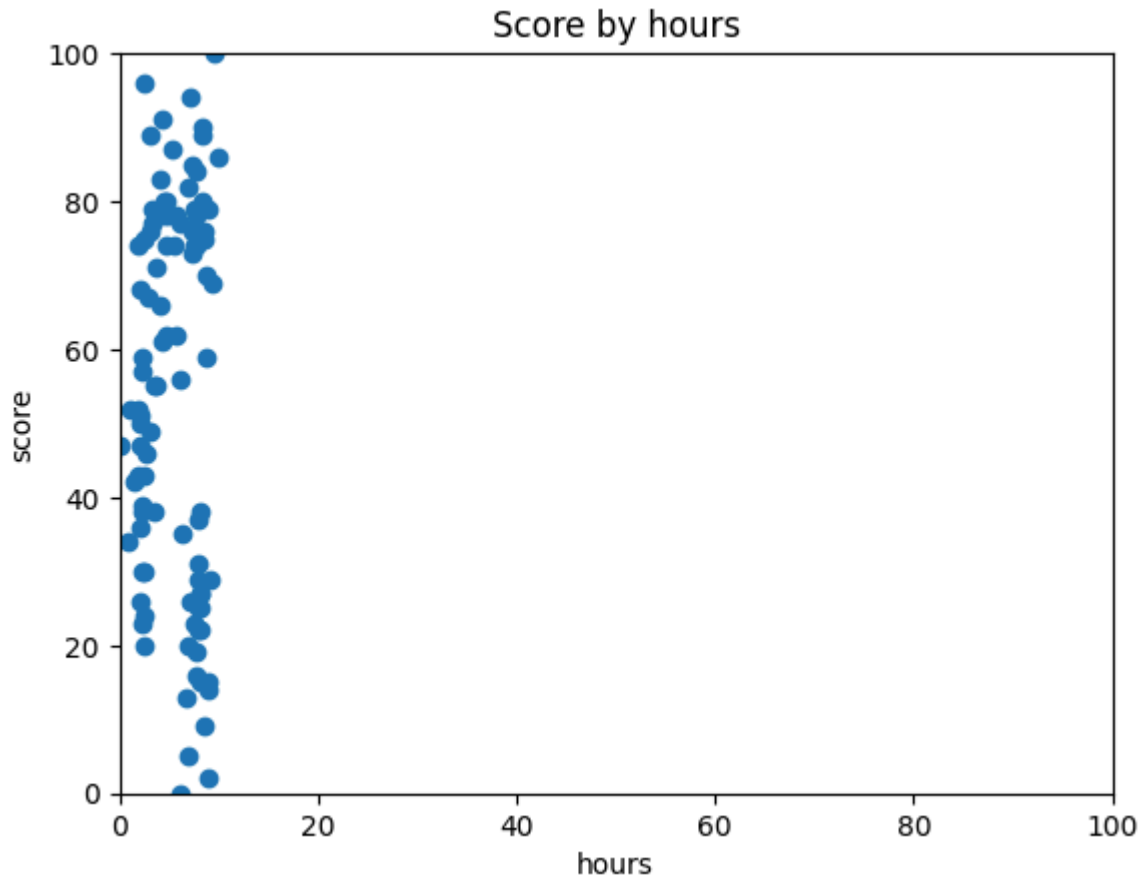
데이터 시각화 (전체 데이터 분포 확인)

```
plt.scatter(X[:, 0], X[:, 1]) # x축 : hour, y축 : score
plt.title('Score by hours')
plt.xlabel('hours') # 공부시간
plt.ylabel('score') # 점수
plt.show()
```



데이터 시각화 (축 범위 통일)

```
plt.scatter(X[:, 0], X[:, 1]) # x축 : hour, y축 : score
plt.title('Score by hours')
plt.xlabel('hours')
plt.xlim(0, 100) # X 축
plt.ylabel('score')
plt.ylim(0, 100) # Y 축. 두 축이 다르면 실제 구하는 거리와 보이는 거리가 다르기 때문에 두축을 같은
plt.show()
```



피처 스케일링 (Feature Scaling)

Feature Scaling은 데이터의 피처(feature)들이 서로 다른 범위(scale)를 가질 때 이를 동일한 스케일로 맞추는 작업입니다. 서로 다른 스케일을 가진 데이터들은 모델의 성능을 떨어뜨리기 때문에 중요한 기법입니다.

데이터 스케일링(Data Scaling)이란 서로 다른 변수의 값 범위를 일정한 수준으로 맞추는 작업을 의미합니다. 값을 조정하는 과정이기 때문에 수치형 변수에만 적용해야 합니다.

사이킷런에서는 스케일링을 수행하기 위한 다양한 스케일러를 제공하는데요. 이때 모든 스케일러는 공통적으로 다음과 같은 메서드를 이용합니다.

- fit(): 데이터 변환을 위한 기존 정보 설정 (ex: 데이터 세트의 최댓값/최솟값)
- transform(): fit()을 통해 설정된 정보를 이용해 실제로 데이터를 변환

그리고 fit_transform()은 위 두 가지 메서드를 한 번에 적용하는 기능을 수행합니다.

스케일링의 대표적인 방법인 표준화(Standardization)와 정규화(Normalization)가 있다.

표준화(Standardization)는 변수 각각의 평균을 0, 분산을 1로 만들어주는 스케일링 기법입니다. 표준화가 적용된 변수는 가우시안 정규분포를 가진 값으로 변환됩니다.

표준화를 위해, 사이킷런에서는 StandardScaler를 제공합니다. 이는 표준화를 쉽게 지원하기 위한 클래스로, 개별 변수를 평균이 0이고 분산이 1인 값으로 변환해줍니다.

- 표준화를 위해 사이킷런의 StandardScaler 클래스를 사용합니다.
- fit_transform()을 사용해서 학습과 스케일링을 한 번에 적용합니다.

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)
X[:5] # 초기값보다 작아지는 것을 볼수 있다.
```

```
array([[ 0.68729921,  0.73538376],
       [-0.66687438,  0.04198891],
       [-0.77161709,  0.04198891],
       [-0.9100271 ,  1.35173473],
       [-0.8090252 ,  0.96651537]])
```

데이터 시각화 (스케일링된 데이터)

```
plt.figure(figsize=(5, 5)) # 정확한 사이이즈를 위해서 가로 세로 크기를 5로 지정한다. 그럼 정사각형
plt.scatter(X[:, 0], X[:, 1])
plt.title('Score by hours')
plt.xlabel('hours')
plt.ylabel('score')
```