

딥러닝 스터디

# 5주차 비지도학습.

20기 시각화 한은빈  
20기 시각화 홍나연



# 비지도학습



## ▶ 정의

지도 학습과는 달리 정답(label, target)이 없는 데이터를 비슷한 특징끼리 군집화하여 새로운 데이터에 대한 결과를 예측하는 방법

#정답 없음

#비슷한 특징

#군집화

## ▶ 언제 사용하는지?

데이터 자체가 부족하거나 훈련 데이터를 수집하기에는 비용이 너무 높은 등의 이유로 출력에 대해 알 수 없거나 활용할 수 없을 때 주로 사용

#데이터 부족

#많은 비용

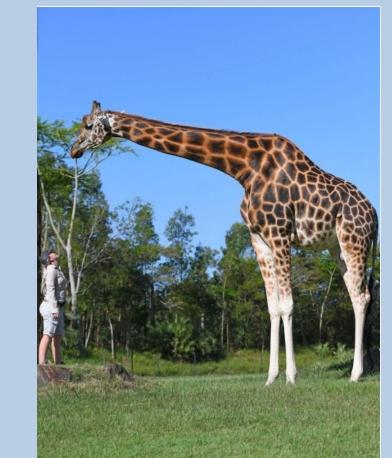
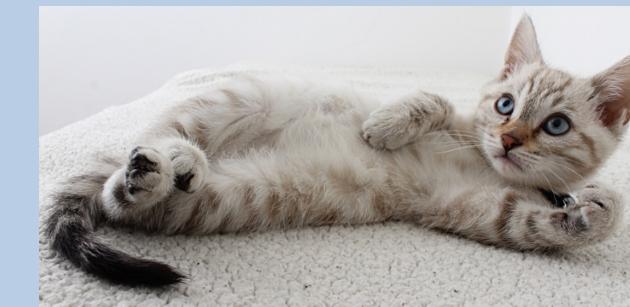
#출력 알 수 없음

# 비지도학습



## ▶ 예시

다음과 같이 8개의 동물 사진이 있다. 우리는 각 사진이 어떤 동물인지 알지만, 모른다는 가정하에 비슷한 모습을 지닌 동물끼리 그룹으로 만들어보려고 한다.



# 비지도학습



## ▶ 예시

그룹1 : 다리 4개가 있으며 귀가 뾰족하게 생긴 형태

그룹2 : 두발로 서있는 형태

그룹3 : 목, 다리가 긴 형태

그룹 1



그룹 2



그룹 3



이와 같이 비지도 학습은 예측 등이 아닌  
데이터가 어떻게 구성되어 있는지 밝히는데 주로 사용하고,  
일종의 그룹핑 알고리즘으로 볼 수 있다.

# 비지도학습



## - 비지도학습의 2가지 종류

### ➤ Clustering (군집 = 군집화 = 클러스터)

각 데이터의 유사성(거리)을 측정한 후,  
유사성이 높은(=거리가 짧은) 데이터끼리 집단으로 분류하는 것

# 데이터 그룹화

# K-Means

# 사용자 관심사에 따라  
그룹화하여 마케팅에 활용

### ➤ dimensionality reduction (차원 축소)

차원을 나타내는 특성을 줄여서 데이터를 줄이는 방식

# 데이터 간소화

# PCA

# 데이터 압축  
# 중요한 속성 도출

# K-means<sup>•</sup> Clustering (K-평균 군집화)



## ▶ K-means Clustering 이란?

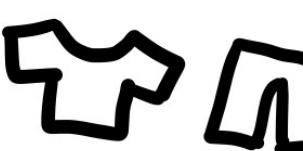
데이터를 입력받아 소수의 그룹으로 묶는 알고리즘으로,  
레이블이 없는 데이터를 입력받아 각 데이터에 레이블을 할당 후 군집화를 수행하는 것

## ▶ 왜 사용할까?

주어진 데이터에 대한 군집화

## ▶ 언제 사용하면 좋을까?

주어진 데이터셋을 이용하여 몇 개의 클러스터를 구성할지  
사전에 알 수 있을 때 사용하면 유용

‘옷 쇼핑몰’ 어플 유저 

→ 3개의 군집 예상

A군집 -  
just 아이쇼핑  
구매X. 장바구니 O

B군집  
쇼핑러버  
매달 꾸준히 구매

C군집  
총종 구매  
몇 달에 한번씩 구매

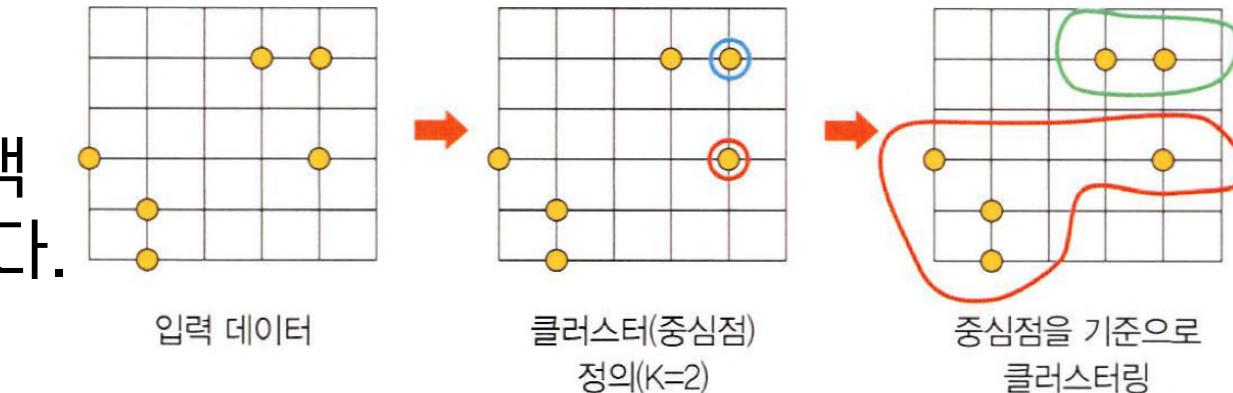
# K-means Clustering (K-평균 군집화)



## ▶ K-means Clustering 의 학습 과정

### 1. 중심점 선택

: 랜덤하게 초기 중심점(centroid)을 선택  
\*  $K = 2$ 로 초기화 = 2개의 군집으로 나눈다.

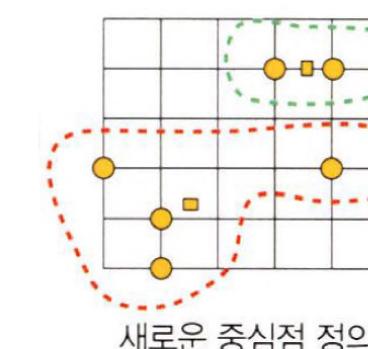


### 2. 클러스터 할당

:  $K$ 개의 중심점과 각각의 개별 데이터 간의 거리를 측정한 후,  
가장 가까운 중심점을 기준으로 데이터를 할당  
\* 클러스터링 : 데이터를 하나 혹은 둘 이상의 덩어리로 묶는 과정  
\* 클러스터 : 덩어리 자체를 의미

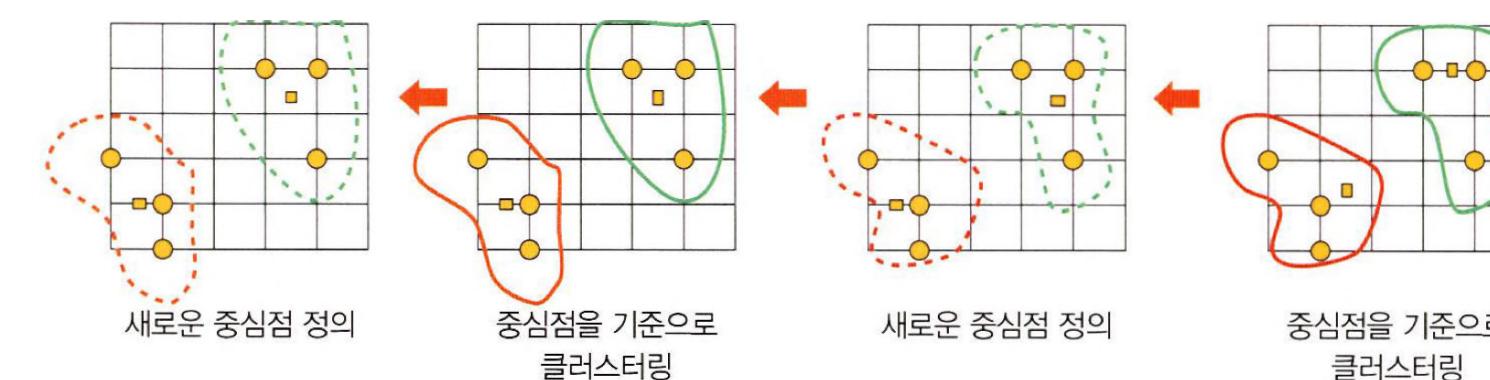
### 3. 새로운 중심점 선택

: 클러스터마다 새로운 중심점을 계산



### 4. 범위 확인

: 선택된 중심점에 더 이상의 변화가 없다면 진행을 끝낸다.  
만약 계속 변화가 있다면 2, 3 과정 반복



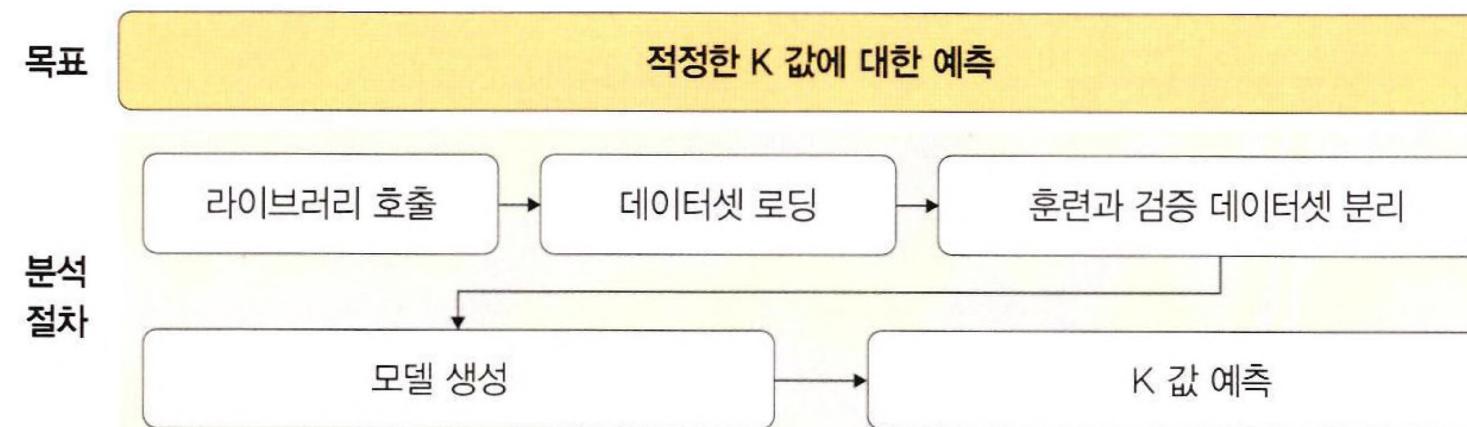
# K-means Clustering (K-평균 군집화)



## ▶ K-means Clustering 예제

해당 알고리즘의 성능은 K값에 따라 달라진다. -> 적절한 K 값 찾는 것이 중요 !!

### - 진행 방향



### 1. 필요한 라이브러리 호출

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
```

### 2. 예제 파일(상품의 연 지출 데이터 : sales data.csv) 호출

```
data = pd.read_csv('../chap03/data/sales data.csv')
data.head()
```

# K-means<sup>•</sup> Clustering (K-평균 군집화)

- Channel: 고객 채널(호텔/레스토랑/카페) 또는 소매 채널(명목형 데이터)
- Region: 고객 지역(명목형 데이터)
- Fresh: 신선한 제품에 대한 연간 지출(연속형 데이터)
- Milk: 유제품에 대한 연간 지출(연속형 데이터)
- Grocery: 식료품에 대한 연간 지출(연속형 데이터)
- Frozen: 냉동 제품에 대한 연간 지출(연속형 데이터)
- Detergents\_Paper: 세제 및 종이 제품에 대한 연간 지출(연속형 데이터)
- Delicassen: 조제 식품에 대한 연간 지출(연속형 데이터)

## 3. 연속형/명목형 데이터 분류

categorical\_features = ['Channel', 'Region'] ..... 명목형 데이터

continuous\_features = ['Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents\_Paper', 'Delicassen'] ..... 연속형 데이터

for col in categorical\_features:

```
dummies = pd.get_dummies(data[col], prefix=col)
data = pd.concat([data, dummies], axis=1)
data.drop(col, axis=1, inplace=True)
```

----- 명목형 데이터는 판다스의 get\_dummies() 메서드를 사용하여 숫자(0과 1)로 변환

data.head()

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	2	3	22615	5410	7198	3915	1777	5185



	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen	Channel_1	Channel_2	Region_1	Region_2	Region_3
0	12669	9656	7561	214	2674	1338	0	1	0	0	1
1	7057	9810	9568	1762	3293	1776	0	1	0	0	1
2	6353	8808	7684	2405	3516	7844	0	1	0	0	1
3	13265	1196	4221	6404	507	1788	1	0	0	0	1
4	22615	5410	7198	3915	1777	5185	0	1	0	0	1

-> 연속형, 명목형 데이터로 분류

# K-means Clustering (K-평균 군집화)



## 4. 데이터 전처리 (scaling)

데이터 범위가 다르기 때문에 범위에 따라 중요도가 달라지는 것을 방지 !!  
-> 일정한 범위를 유지하도록 사이킷런의 MinMaxScaler() 메서드 사용

```
mms = MinMaxScaler()  
mms.fit(data)  
data_transformed = mms.transform(data)
```

## 5. 적당한 K 값 추출

```
Sum_of_squared_distances = [] ..... ①  
K = range(1, 15) ..... K에 1부터 14까지 적용해 봅니다.  
for k in K:  
    km = KMeans(n_clusters=k) ..... 1~14의 K 값 적용  
    km = km.fit(data_transformed) ..... KMeans 모델 훈련  
    Sum_of_squared_distances.append(km.inertia_)
```

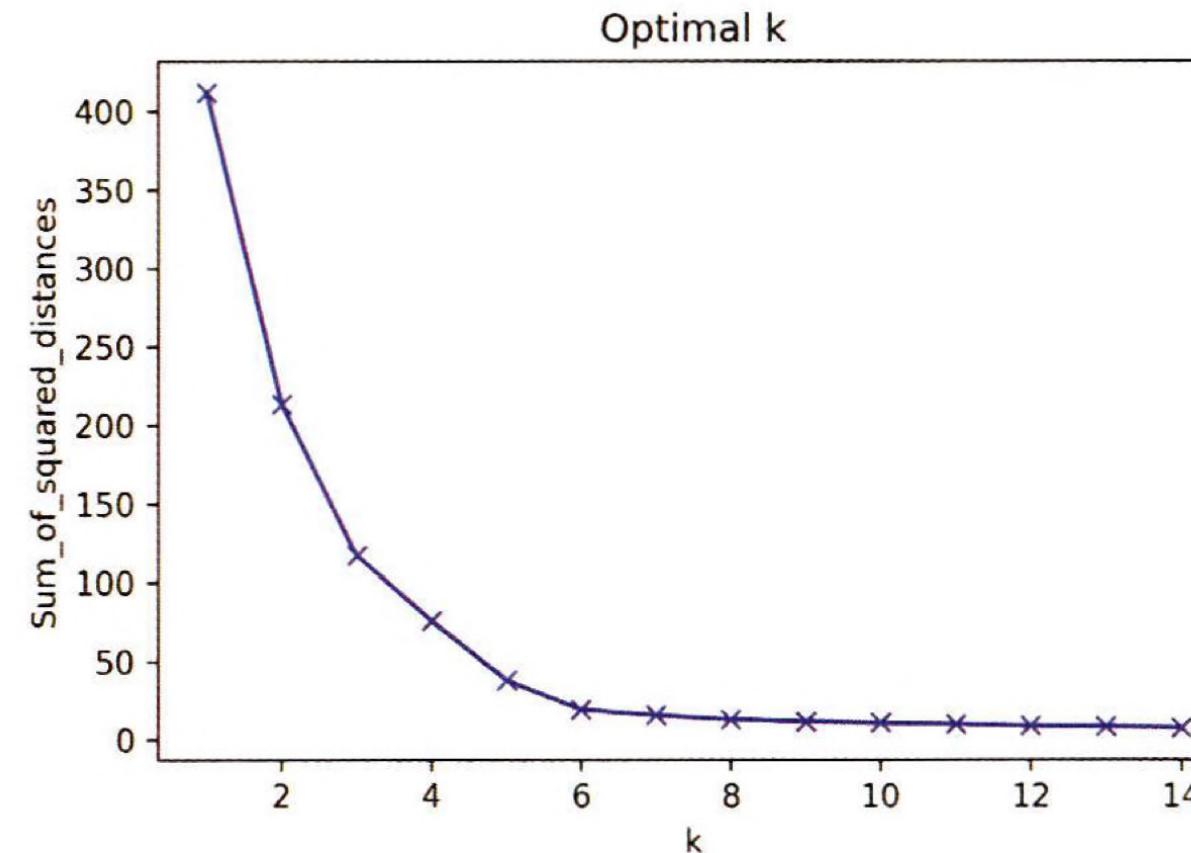
```
plt.plot(K, Sum_of_squared_distances, 'bx-')  
plt.xlabel('k')  
plt.ylabel('Sum_of_squared_distances')  
plt.title('Optimal k')  
plt.show()
```

\* Sum\_of\_squared\_distances.append( ~~ )  
: inertia 는 각 데이터로부터 자신이 속한  
군집의 중심까지의 거리를 의미  
-> SSD(거리 제곱의 합)

# K-means<sup>•</sup> Clustering (K-평균 군집화)



## Elbow Methods(엘보우 메소드)



## SSD(거리 제곱의 합)의 수식

$$SSD = \sum_{x,y} (I_1(x, y) - I_2(x, y))^2$$

그래프 해석 : K = 5 이후의 지점부터는 유의미한 변화가 없다. (= 0에 가까워지고 있다.)  
즉, K = 5가 적절하다고 볼 수 있다.  
x축 = 클러스터 개수 / y축 = SSD(거리 제곱의 합)

## SSD(거리 제곱의 합, Sum of Squared Distances)

: x, y 두 데이터의 차를 구해 제곱한 값을 모두 더한 후 유사성을 측정하는 데 사용  
즉, 가장 가까운 클러스터 중심까지 거리를 제곱한 값의 합을 구할 때 사용  
-> K 값이 증가하면 SSD는 0이 되는 경향이 있으며, K의 최댓값 n(= 샘플 수)으로 설정하면 각 샘플이 자체 클러스터를 형성하므로 SSD = 0 과 같아지기 때문

# 밀도기반 군집분석 (DBSCAN)



# 밀도기반 군집분석 (DBSCAN)



01

왜 사용할까?

주어진 데이터에 대한 군집화

02

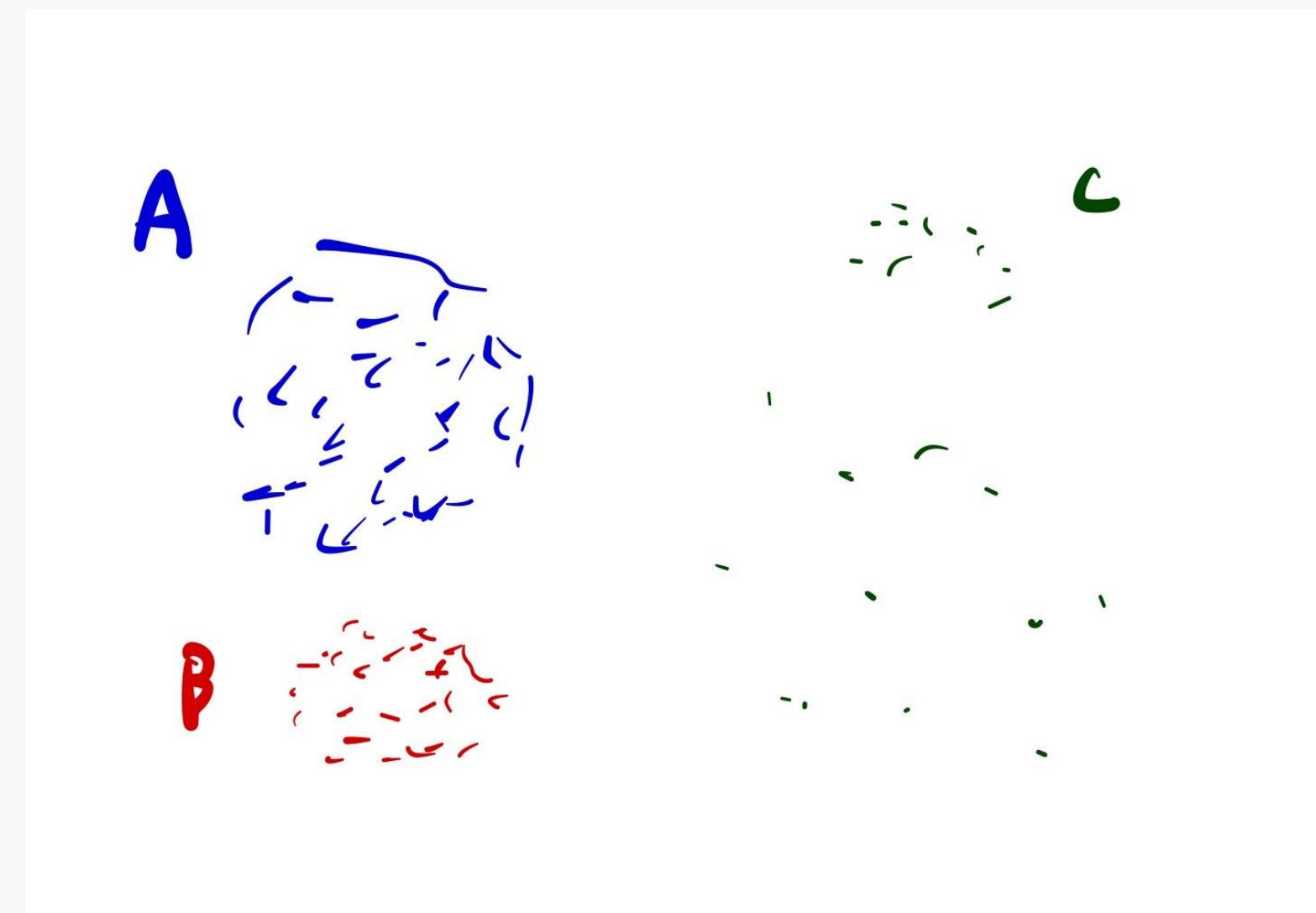
언제 사용하면 좋을까?

K-평균 군집화와는 다르게 사전에 클러스터의 숫자를 알지 못할 때  
주어진 데이터에 이상치가 많이 포함되었을 때

# 밀도기반 군집분석 (DBSCAN)



밀도를 정확하게 파악해서  
밀도가 높은 군집 두개와 밀도가 낮은 군집 하나를 구별 가능

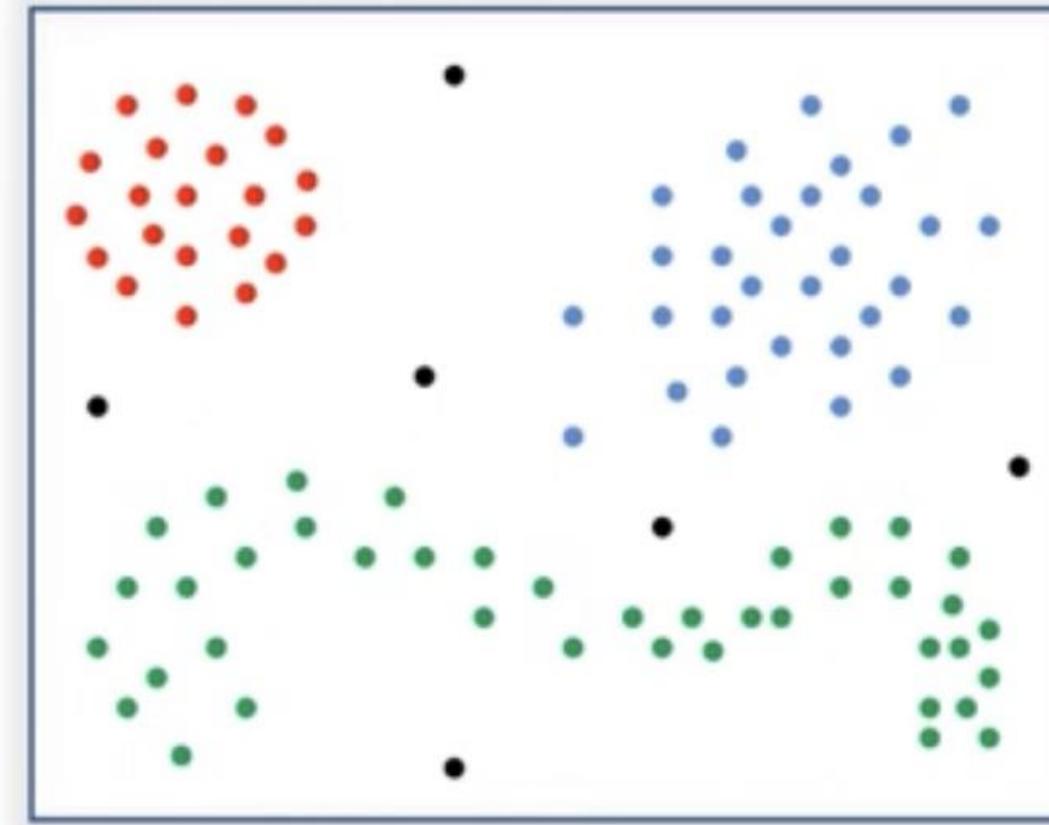


- 참고자료: 고려대학교 산업경영공학부 DSBA 연구실  
(<https://www.youtube.com/watch?v=PuVH38UpgNU>)

# 밀도기반 군집분석 (DBSCAN)



밀도 기반 군집 분석(DBSCAN)의 장점



01

구형 뿐만 아니라 어떤 모형이든 클러스터링 할 수 있음

02

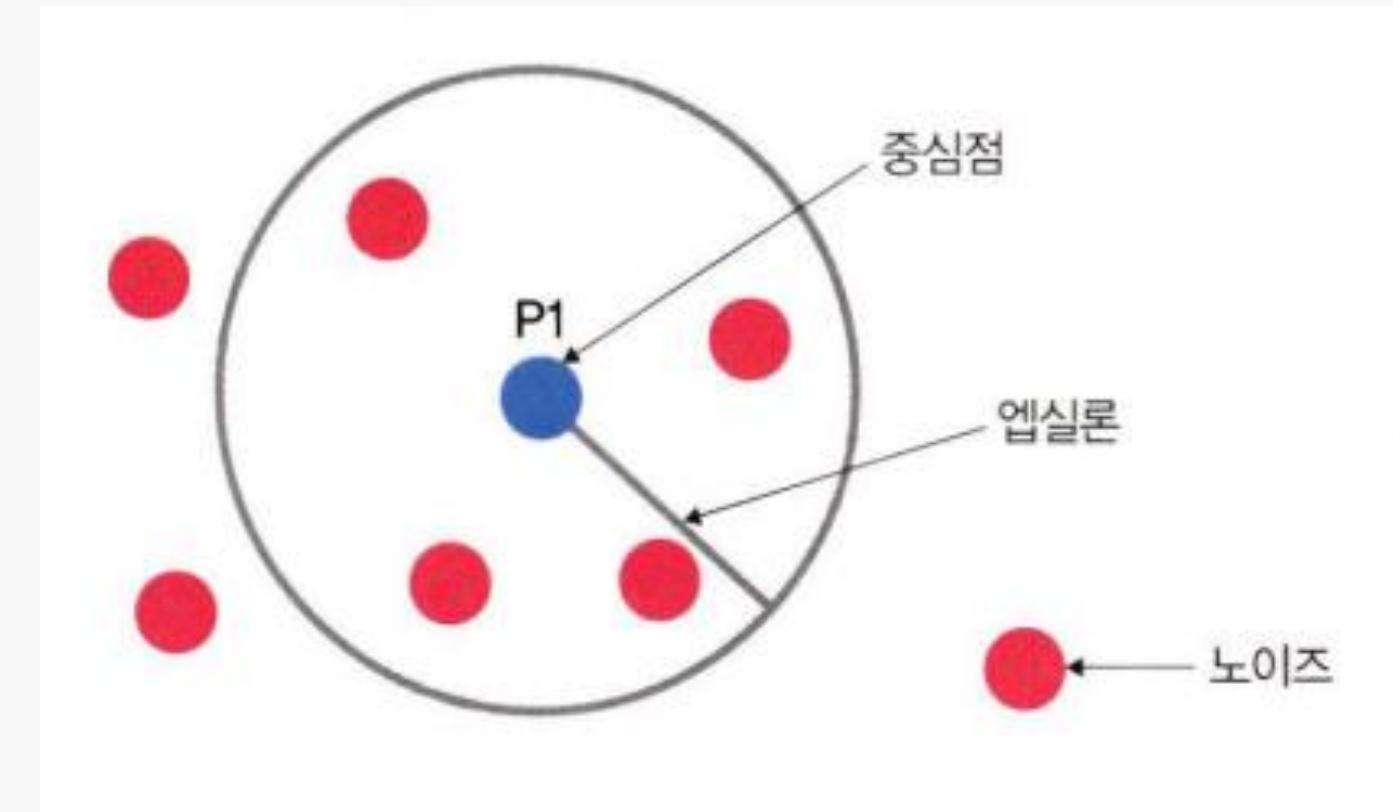
어떤 클러스터에도 할당되지 않는 개체를 존재하도록 할 수 있음 (noise)

- 참고자료: 고려대학교 산업경영공학부 DSBA 연구실  
(<https://www.youtube.com/watch?v=PuVH38UpgNU>)

# 밀도기반 군집분석 (DBSCAN)



밀도 기반 군집 분석(DBSCAN)의 파라미터(parameter)



01

엡실론 (epsilon)  
두 점 사이의 거리로 임계치 (범주) 역할

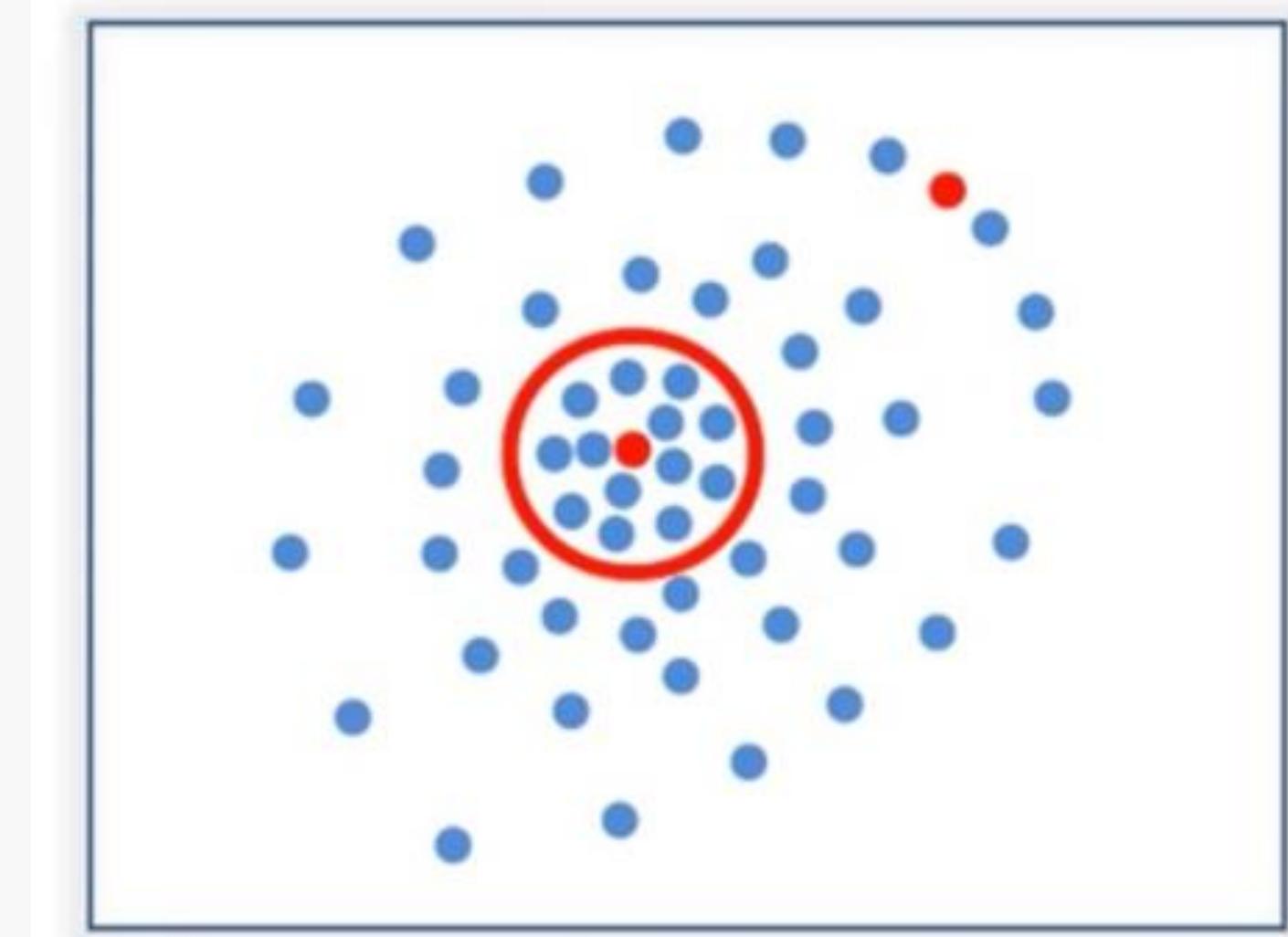
02

M(minPts)  
중심점을 만드는 구성 요건으로 엡실론 내 데이터 개수

- 참고자료: 고려대학교 산업경영공학부 DSBA 연구실  
(<https://www.youtube.com/watch?v=PuVH38UpgNU>)

## 1단계 엡실론 내 점 개수 확인 및 중심점 결정

# 밀도기반 군집분석 (DBSCAN)



군집이 되려면 클러스트 내에 임의 점을 찔러도  
반경 엡실론 (epsilon) 내에는  
우리가 정한 최소 개수((M)minPts)가 있어야 한다!

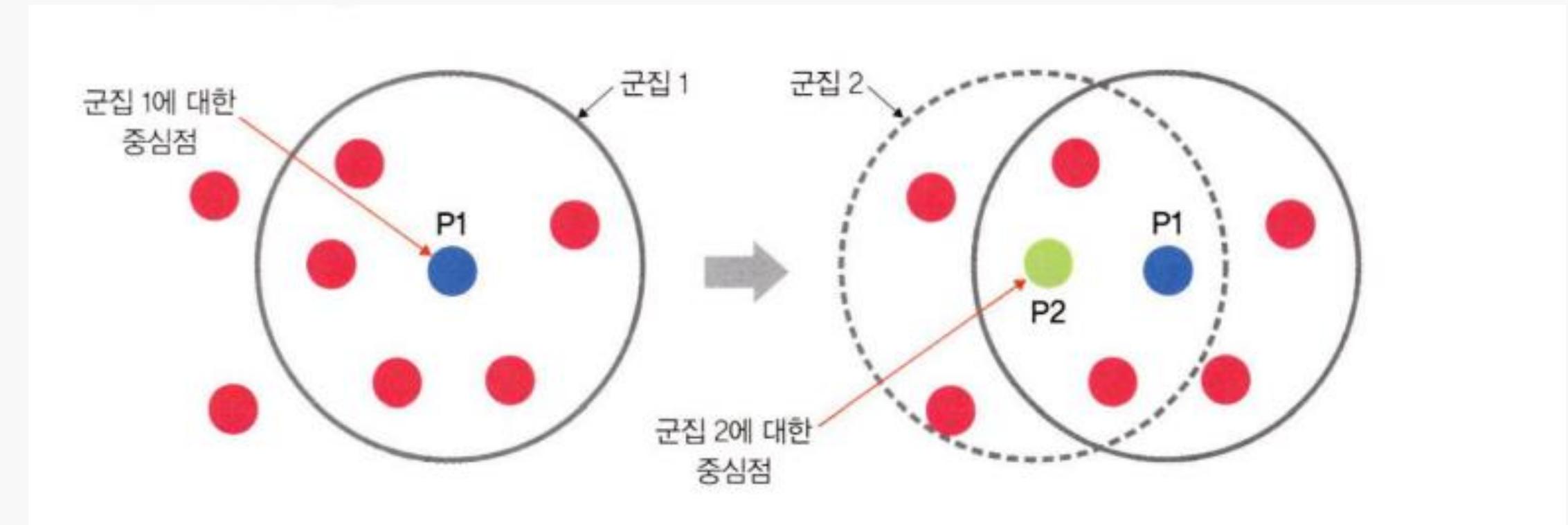
\* 최소 개수((M)minPts)를 셀 때는 core points도 포함

- 참고자료: 고려대학교 산업경영공학부 DSBA 연구실  
(<https://www.youtube.com/watch?v=PuVH38UpgNU>)

# 밀도기반 군집분석 (DBSCAN)



## 2단계 군집 확장



- 1)  $p \in N_\epsilon(q)$  (*reachability*)
- 2)  $|N_\epsilon(q)| \geq MinPts$  (*core point condition*)

- 1) P와 q는 이어져있다  
2) 반경 입실론 내에는 우리가 정한 최소 개수((M)minPts)가 있어야 한다

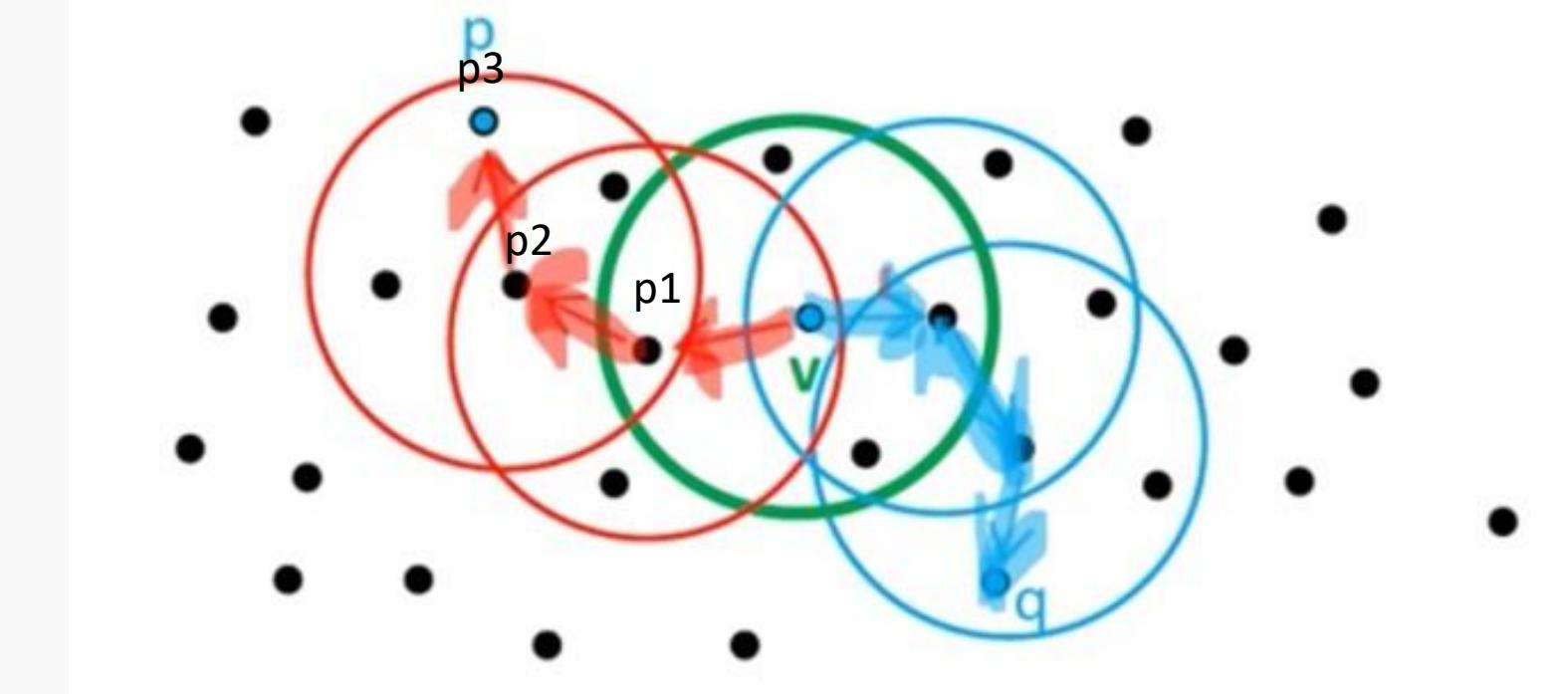


Direct  
destiny reachable

# 밀도기반 군집분석 (DBSCAN)



3단계 1~2단계 반복



border points로부터 direct destiny reachable를  
찾아내면서 군집을 확장함

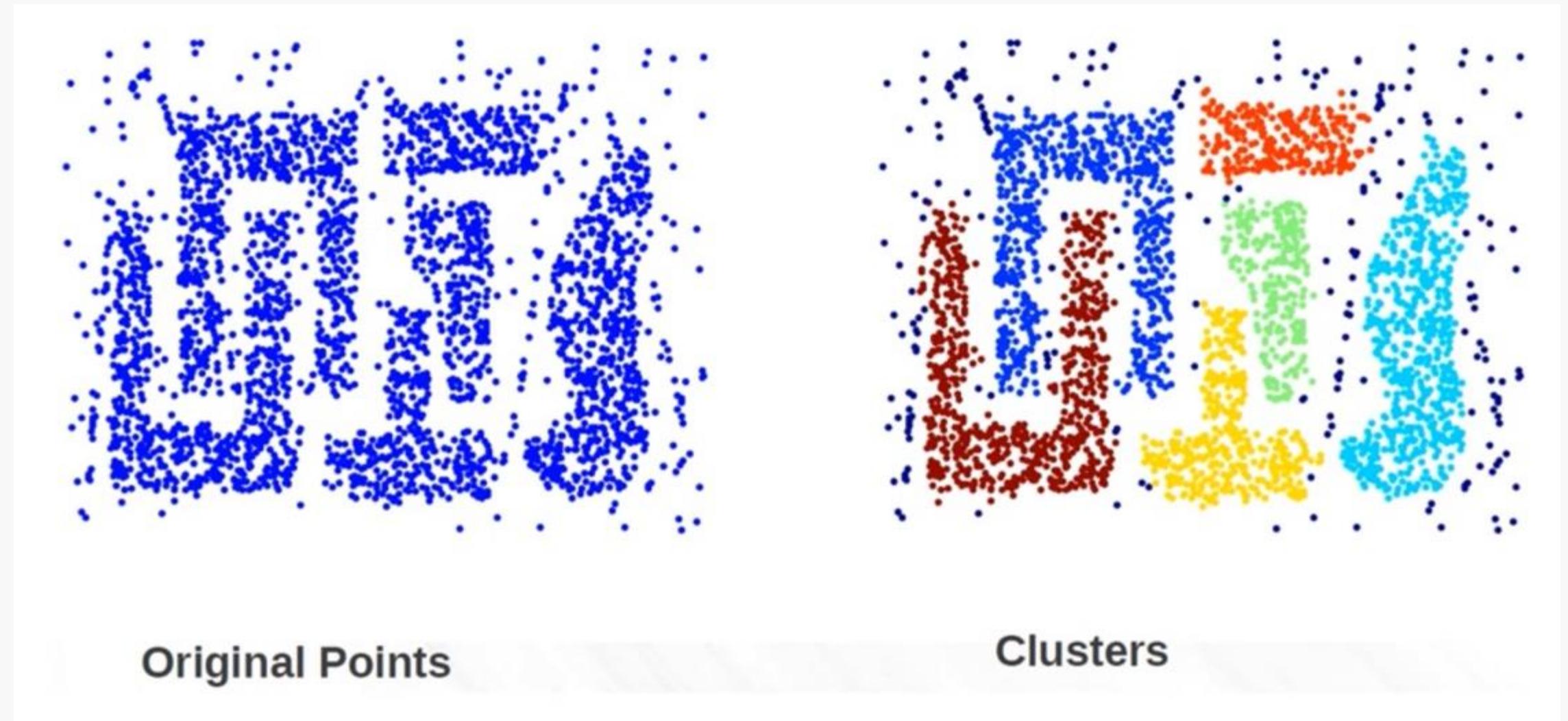
- 1) V는  $p_1$ 의 direct destiny reachable  $\rightarrow$  V는  $p_1$ 으로 부터 direct destiny reachable
- 2)  $p_1$ 은  $p_2$ 의 direct destiny reachable  $\rightarrow$  V는  $p_2$ 로 부터 destiny reachable
- 3)  $p_2$ 는  $p_3$ 의 direct destiny reachable  $\rightarrow$  V는  $p_3$ 로 부터 destiny reachable

Core points 의 destiny reachable를 찾아내면서  
군집을 확장함

# 밀도기반 군집분석 (DBSCAN)



## 4단계 노이즈 정의



- 참고자료: 고려대학교 산업경영공학부 DSBA 연구실  
(<https://www.youtube.com/watch?v=PuVH38UpgNU>)

- 노이즈 = 주어진 데이터셋과 무관하거나  
무작위성 데이터로 전처리 과정에서 제거해야 할 부분

# 주성분 분석 (PCA)



# 주성분 분석 (PCA)



01

왜 사용할까?

주어진 데이터의 간소화

02

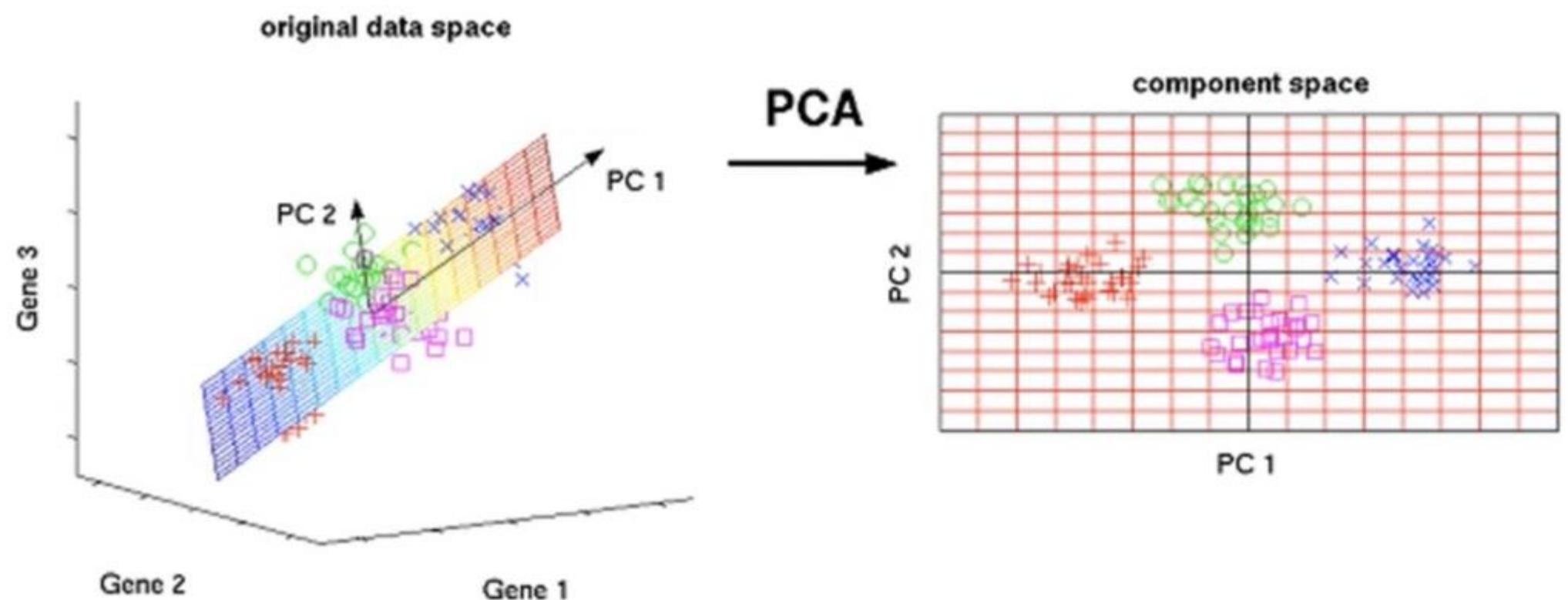
언제 사용하면 좋을까?

현재 데이터의 특성(변수)이 너무 많을 경우에는 데이터를 하나의  
플롯(plot)에 시각화해서 살펴보는 것이 어려움  
이때 특성 P개를 두세 개 정도로 압축해서 데이터를 시각화하여 살펴보고  
싶을 때 유용함

# 주성분 분석 (PCA)



고차원 데이터를 저차원(차원 축소)



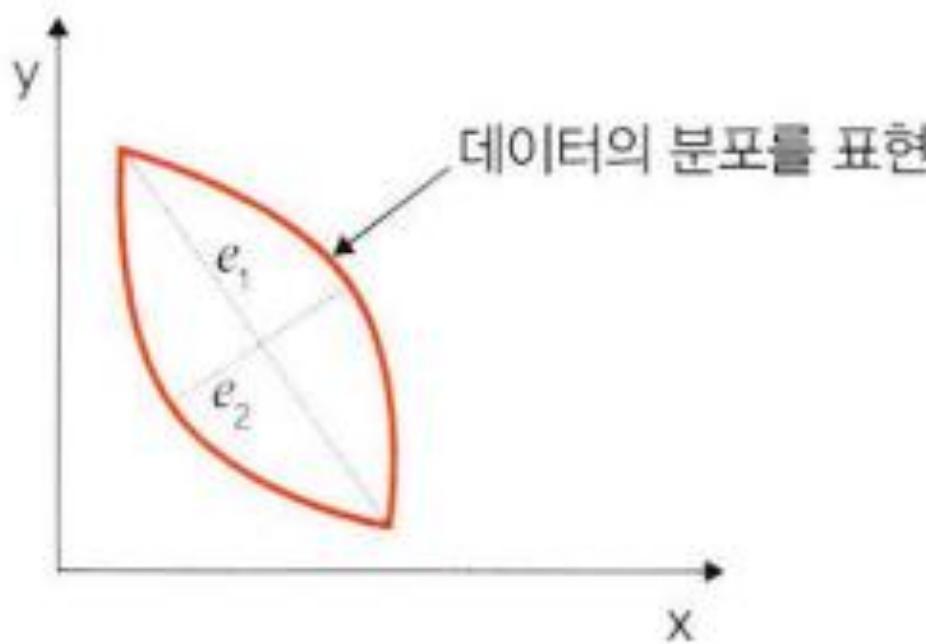
- 참고자료 다크프로그래머  
<https://darkpgmr.tistory.com/110>

# 주성분 분석 (PCA)



1. 데이터들의 분포 특성을 잘 설명하는 벡터를 두 개 선택

데이터의 분산을 보존하는 변수를 찾기



e1의 방향과 크기, e2의 방향과 크기  
-> 데이터 분포 형태

2. 벡터 두 개를 위한 적정한 가중치를 찾을 때까지 학습을 진행

- 참고자료 다크프로그래머  
<https://darkpgmr.tistory.com/110>

# 밀도기반군집분석 (DBSCAN) 주성분 분석 (PCA) 코드로 살펴보기



# 코드로 살펴보기



## 분석 절차



### 1. 라이브러리 호출

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

from sklearn.cluster import DBSCAN ..... 밀도 기반 군집 분석
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize
from sklearn.decomposition import PCA ..... 데이터 차원 축소
```

- 참고자료 다크프로그래머  
<https://darkpgmr.tistory.com/110>

# 코드로 살펴보기



## 2. 데이터셋 로딩

```
X = pd.read_csv('..../chap03/data/credit card.csv')  
X = X.drop('CUST_ID', axis=1) ..... 불러온 데이터에서 'CUST_ID' 열(칼럼)을 삭제  
X.fillna(method='ffill', inplace=True) ..... ①  
print(X.head()) ..... 데이터셋 형태 확인
```

## 3. 데이터 전처리

```
scaler = StandardScaler()  
X_scaled = scaler.fit_transform(X) ..... 평균이 0, 표준편차가 1이 되도록 데이터 크기를 조정  
  
X_normalized = normalize(X_scaled) ..... 데이터가 가우스 분포를 따르도록 정규화  
X_normalized = pd.DataFrame(X_normalized) ..... 넘파일 배열을 데이터프레임(dataframe)으로 변환  
  
pca = PCA(n_components=2) ..... 2차원으로 차원 축소 선언  
X_principal = pca.fit_transform(X_normalized) ..... 차원 축소 적용  
X_principal = pd.DataFrame(X_principal)  
X_principal.columns = ['P1', 'P2']  
print(X_principal.head())
```

- 참고자료 다크프로그래머  
<https://darkpgmr.tistory.com/110>

## 4. 모델 생성 및 훈련, 클러스터링에 대한 시각화

# 코드로 살펴보기



- 참고자료 다크프로그래머  
<https://darkpgmr.tistory.com/110>

```
db_default = DBSCAN(eps=0.0375, min_samples=3).fit(X_principal) ..... 모델 생성 및 훈련  
labels = db_default.labels_ ..... 각 데이터 포인트에 할당된 모든 클러스터 레이블의 넘파일 배열을 labels에 저장
```

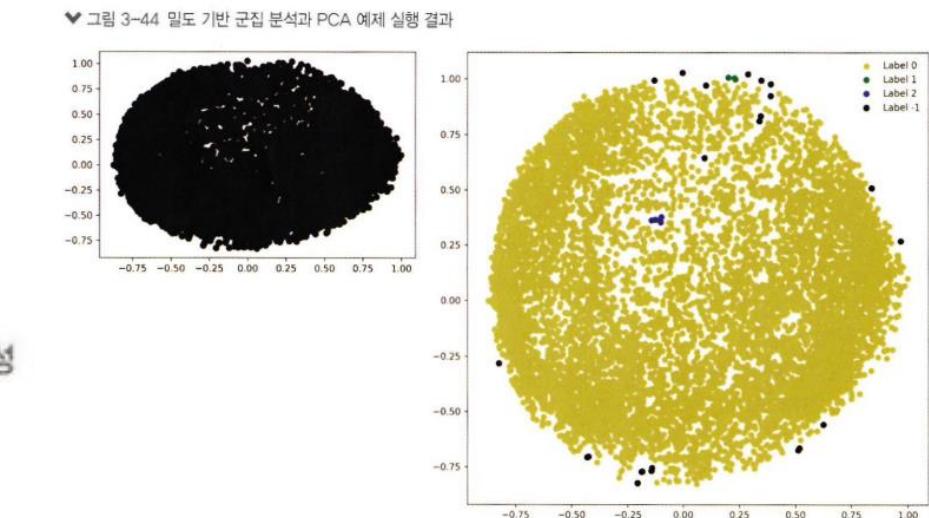
```
colours = {} ..... 출력 그래프의 색상을 위한 레이블 생성  
colours[0] = 'y'  
colours[1] = 'g'  
colours[2] = 'b'  
colours[-1] = 'k'
```

```
cvec = [colours[label] for label in labels] ..... 각 데이터 포인트에 대한 색상 벡터 생성
```

```
r = plt.scatter(X_principal['P1'], X_principal['P2'], color='y');  
g = plt.scatter(X_principal['P1'], X_principal['P2'], color='g');  
b = plt.scatter(X_principal['P1'], X_principal['P2'], color='b');  
k = plt.scatter(X_principal['P1'], X_principal['P2'], color='k'); ..... 플롯(plot)의  
범례(legend) 구성
```

```
plt.figure(figsize=(9,9))  
plt.scatter(X_principal['P1'], X_principal['P2'], c=cvec) ..... 정의된 색상 벡터에 따라 X축에  
P1, Y축에 P2 플로팅(plotting)
```

```
plt.legend((r, g, b, k), ('Label 0', 'Label 1', 'Label 2', 'Label -1')) ..... 범례 구축  
plt.show()
```



# 코드로 살펴보기



- 참고자료 다크프로그래머  
<https://darkpgmr.tistory.com/110>

## 4. 모델 튜닝 K = 3 -> K = 50

```
db = DBSCAN(eps=0.0375, min_samples=50).fit(X_principal)
labels1 = db.labels_

colours1 = {}
colours1[0] = 'r'
colours1[1] = 'g'
colours1[2] = 'b'
colours1[3] = 'c'
colours1[4] = 'y'
colours1[5] = 'm'
colours1[-1] = 'k'

cvec = [colours1[label] for label in labels1]
colors1 = ['r', 'g', 'b', 'c', 'y', 'm', 'k']

r = plt.scatter(
    X_principal['P1'], X_principal['P2'], marker='o', color=colors1[0])
g = plt.scatter(
    X_principal['P1'], X_principal['P2'], marker='o', color=colors1[1])
b = plt.scatter(
    X_principal['P1'], X_principal['P2'], marker='o', color=colors1[2])
c = plt.scatter(
    X_principal['P1'], X_principal['P2'], marker='o', color=colors1[3])
y = plt.scatter(
    X_principal['P1'], X_principal['P2'], marker='o', color=colors1[4])
m = plt.scatter(
    X_principal['P1'], X_principal['P2'], marker='o', color=colors1[5])
k = plt.scatter(
    X_principal['P1'], X_principal['P2'], marker='o', color=colors1[6])

plt.figure(figsize=(9,9))
plt.scatter(X_principal['P1'], X_principal['P2'], c=cvec)
plt.legend((r, g, b, c, y, m, k),
           ('Label 0', 'Label 1', 'Label 2', 'Label 3', 'Label 4', 'Label 5', 'Label -1'),
           scatterpoints=1,
           loc='upper left',
           ncol=3,
           fontsize=8)
plt.show()
```

▼ 그림 3-45 밀도 기반 군집 분석과 PCA 예제 튜닝 결과

