Clustering

20.11.27

Seunghan Lee

Contents

1. Intro to Machine Learning
2. Intro to Clustering
3. Distance
4. K-Means Clustering
5. Choosing optimal number of clusters
6. Other Clustering methods
7. Clustering using Scikit-Learn (Python)

Contents

1. Intro to Machine Learning
2. Intro to Clustering
3. Distance
4. K-Means Clustering
5. Choosing optimal number of clusters
6. Other Clustering methods
7. K-Means Clustering using Scikit-Learn (Python)

1. Introduction To Machine Learning

Fields of ML

- Supervised Learning (지도 학습)
  - predict “Y” given “X”

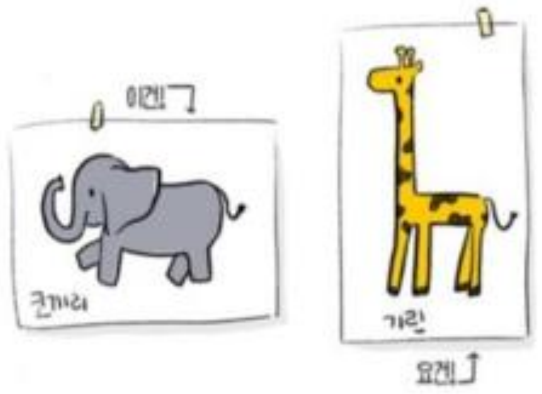
- Unsupervised Learning (비지도 학습)
  - only “X”

- Reinforcement Learning (강화 학습)
  - choose “action” that maximizes the “reward”

머신 러닝

지도 학습 (Supervised Learning)

문제와 정답을 모두 알려주고
공부시키는 방법



? , 분류

비지도 학습 (Unsupervised Learning)

답을 가르쳐주지 않고
공부시키는 방법

연관 규칙, ?

강화 학습 (Reinforcement Learning)

보상을 통해
상은 최대화, 벌은 최소화하는
방향으로 행위를 강화하는 학습

보상

Contents

1. Intro to Machine Learning
2. Intro to Clustering
3. Distance
4. K-Means Clustering
5. Choosing optimal number of clusters
6. Other Clustering methods
7. K-Means Clustering using Scikit-Learn (Python)

2. Intro to Clustering

Purpose of Clustering?

Gather data into groups!

- Maximize “inter-cluster variance”

  (different group -> different characteristics)

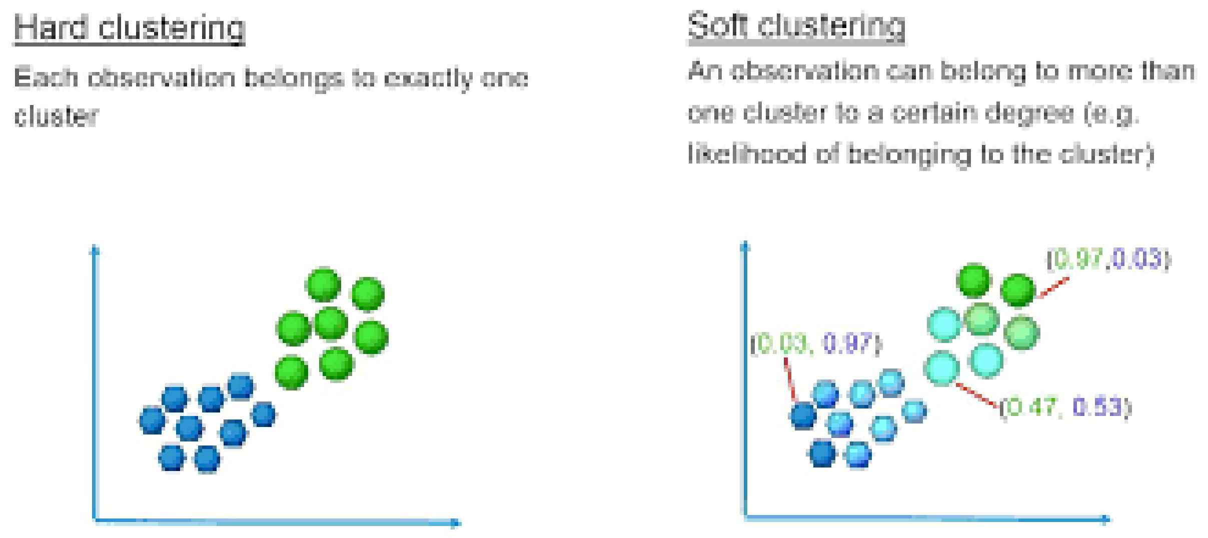
- Minimize “inner-cluster variance”

  (same group -> similar characteristics)

2. Intro to Clustering

- Hard Clustering
  - Data 1 : class A
  - Data 2 : class B

- Soft Clustering
  - Data 1 : class A 90%, B 10%
  - Data 2 : class A 25%, B 75%



Hard clustering

Each observation belongs to exactly one cluster

Soft clustering

An observation can belong to more than one cluster to a certain degree (e.g. likelihood of belonging to the cluster)

2. Intro to Clustering

- Classification (분류) ? Clustering (군집화) ?

2. Intro to Clustering

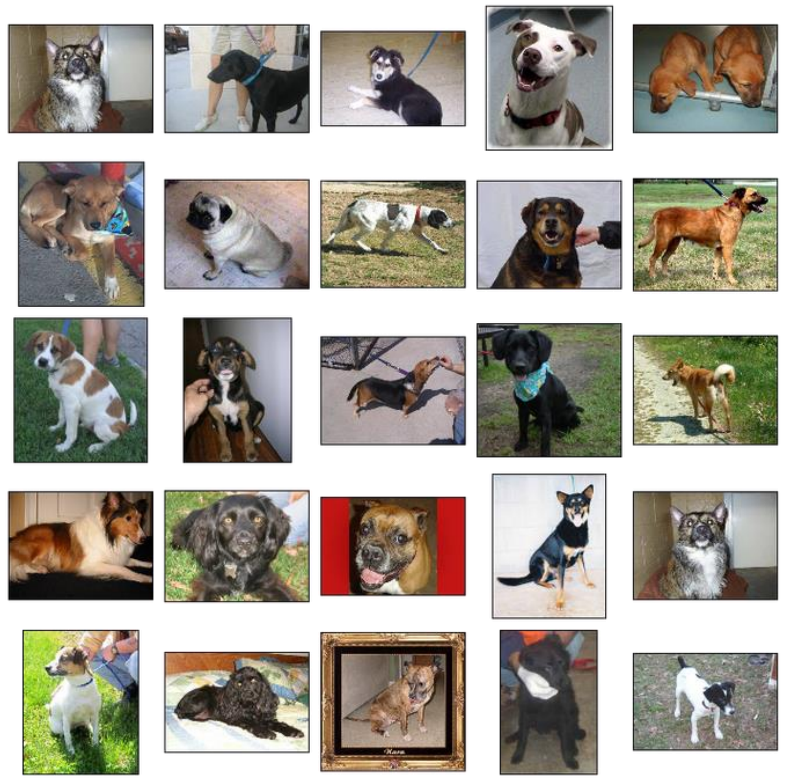
- Classification (분류) ? Clustering (군집화) ?

  Does the problem you want to solve has an “Answer” ?

  - If YES -> Classification
  - If No -> Clustering

2. Intro to Clustering

- Example)



2. Intro to Clustering

- Example)

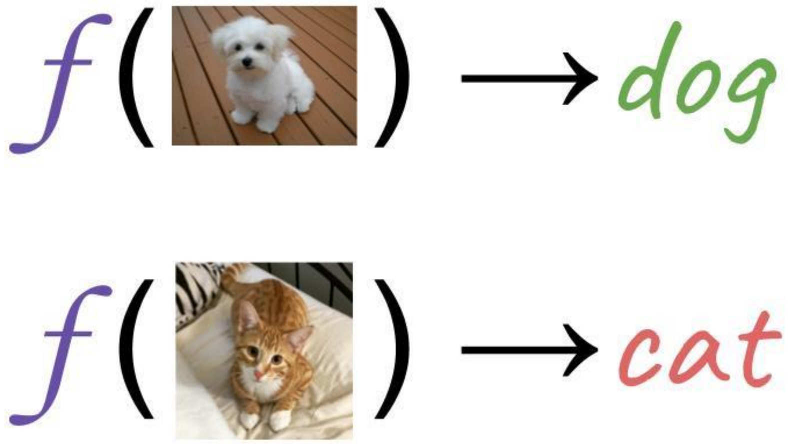


사진	X1	…	…	X999	X1000	Y
사진1	1.3			2.1	0.9	Dog
사진2	2.1			3.3	2.2	Cat
사진3	0.9			1.0	3.2	Cat
사진4	3.2			0.2	1.5	Dog
사진5	2.3			0.8	1.0	Cat
사진6	4.1			2.4	3.4	Dog
사진7	0.9			3.2	2.2	Cat

It's a "Classification" task, since your problem have an answer!

To train the model, you have to feed both "X" and "Y"

2. Intro to Clustering

- Example)

CEO : I want to group my customers into several groups!

2. Intro to Clustering

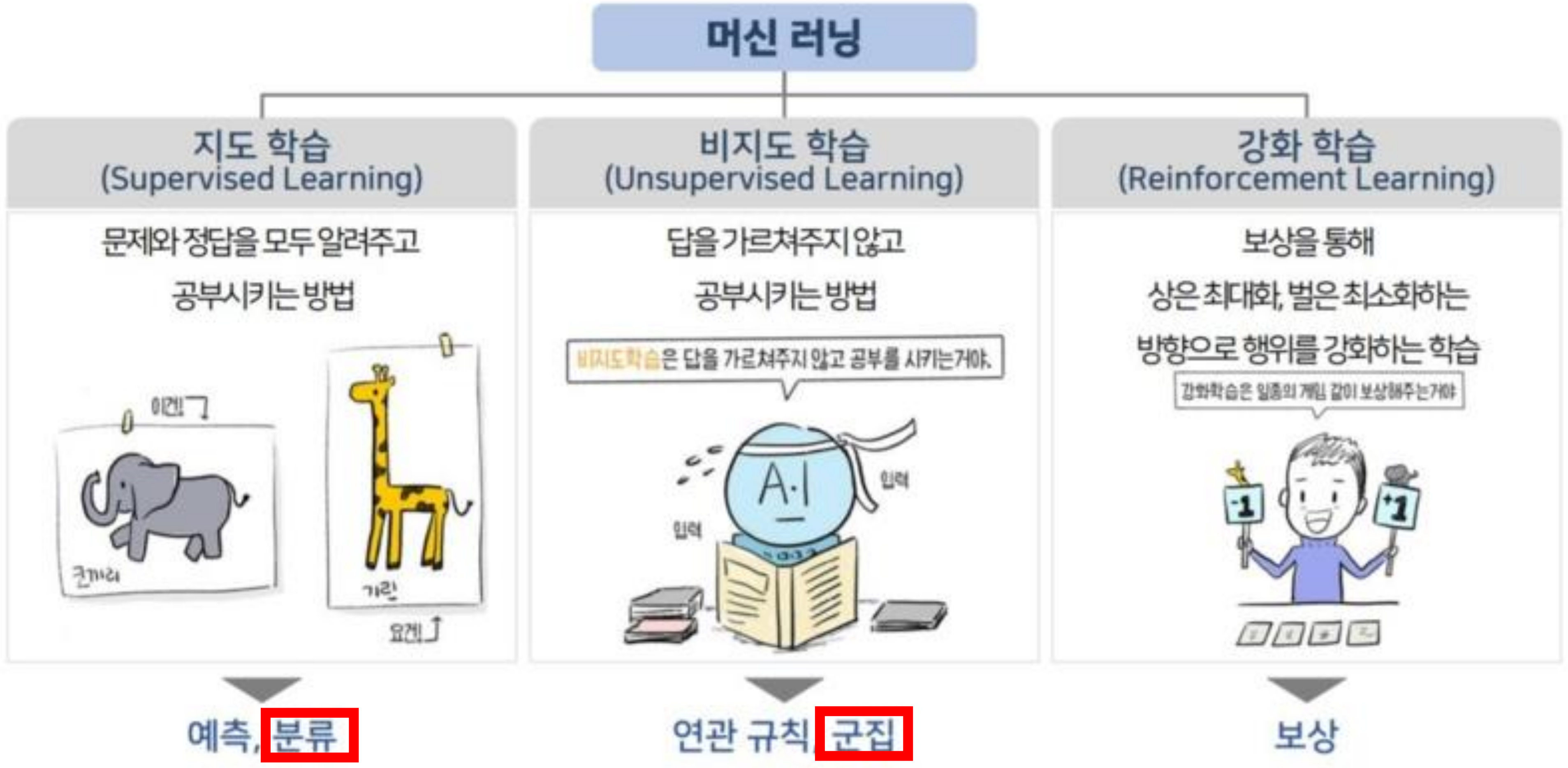
- Example)

고객	X1	…	…	X999	X1000	~~Y~~
고객1	1.3			2.1	0.9	
고객2	2.1			3.3	2.2	
고객3	0.9			1.0	3.2	
고객4	3.2			0.2	1.5	
고객5	2.3			0.8	1.0	
고객6	4.1			2.4	3.4	
고객7	0.9			3.2	2.2	

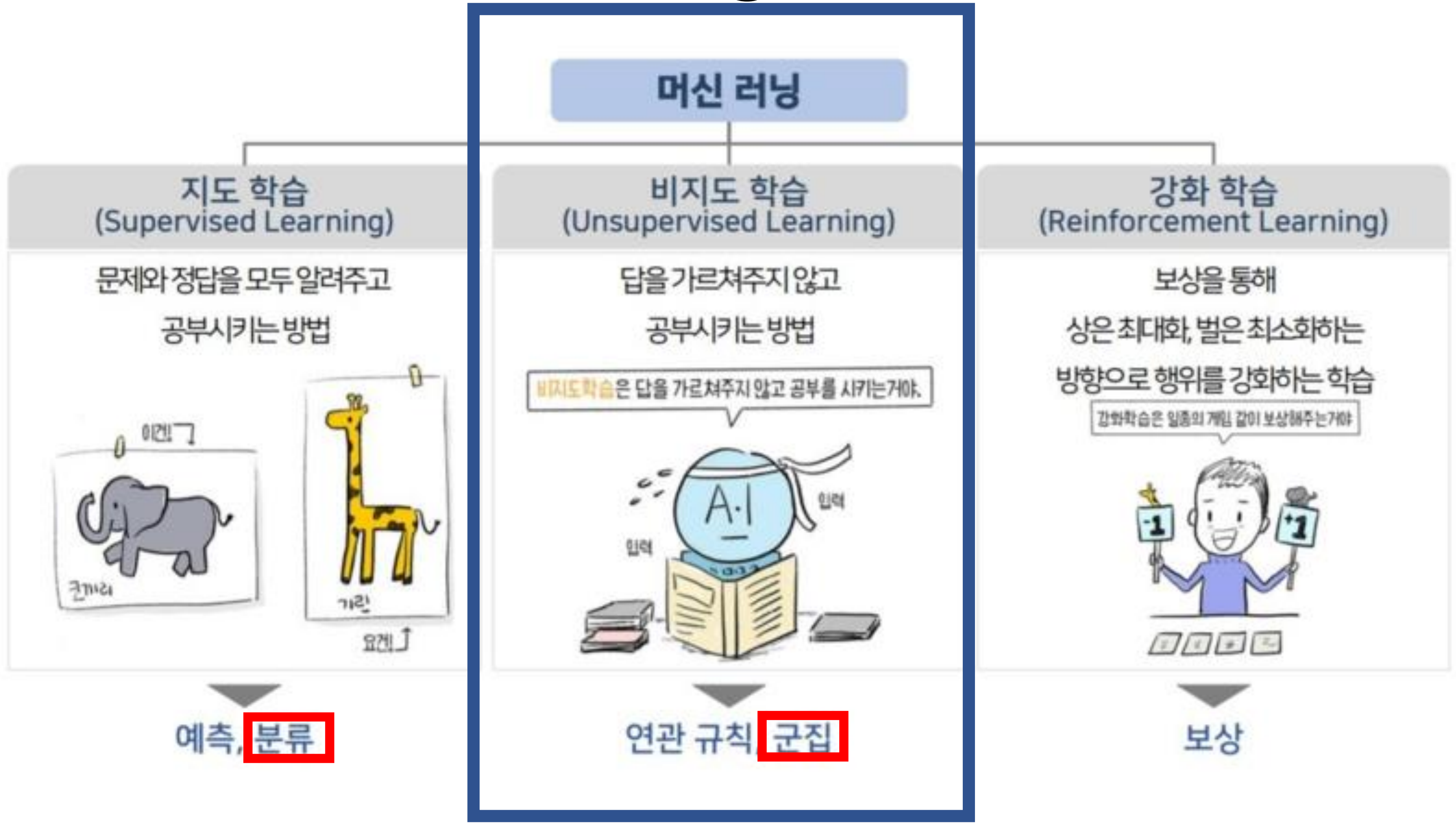
It's a "Clustering" task, since your problem doesn't have an answer!

To train the model, you only feed "X"

2. Intro to Clustering


머신 러닝
지도 학습 (Supervised Learning)
문제와 정답을 모두 알려주고 공부시키는 방법
이건!
코끼리
기린
요건!
예측, 분류
비지도 학습 (Unsupervised Learning)
답을 가르쳐주지 않고 공부시키는 방법
비지도학습은 답을 가르쳐주지 않고 공부를 시키는거야.
A·I
입력
입력
연관 규칙, 군집
강화 학습 (Reinforcement Learning)
보상을 통해 상은 최대화, 벌은 최소화하는 방향으로 행위를 강화하는 학습
강화학습은 일종의 게임 같이 보상해주는거야
-1
+1
보상

2. Intro to Clustering



머신 러닝

지도 학습 (Supervised Learning)

문제와 정답을 모두 알려주고 공부시키는 방법

비지도 학습 (Unsupervised Learning)

답을 가르쳐주지 않고 공부시키는 방법

강화 학습 (Reinforcement Learning)

보상을 통해 상은 최대화, 벌은 최소화하는 방향으로 행위를 강화하는 학습

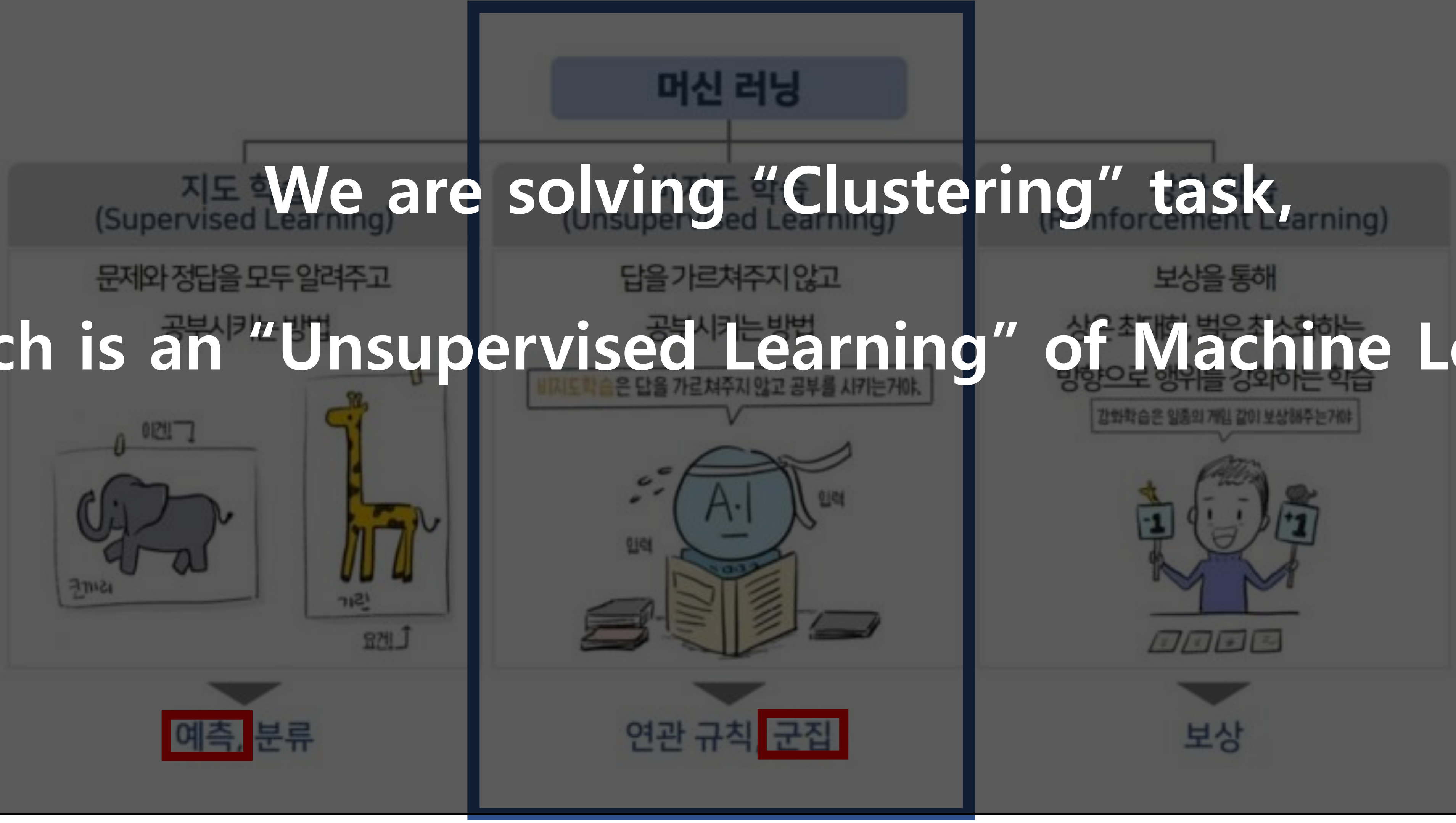
예측, 분류

연관 규칙, 군집

보상

이건!
코끼리
기린
요건!
비지도학습은 답을 가르쳐주지 않고 공부를 시키는거야.
A·I
입력
입력
강화학습은 일종의 게임 같이 보상해주는거야
-1
+1

2. Intro to Clustering



We are solving "Clustering" task,

which is an "Unsupervised Learning" of Machine Learning

Contents

1. Intro to Machine Learning
2. Intro to Clustering
3. Distance
4. K-Means Clustering
5. Choosing optimal number of clusters
6. Other Clustering methods
7. K-Means Clustering using Scikit-Learn (Python)

3. Distance

- Clustering = Grouping data with “Similarity”
- Similarity?

 - have to measure “dissimilarity” (=distance)

3. Distance

- Clustering = Grouping data with “Similarity”
- Similarity?

 - have to measure “dissimilarity” (=distance)

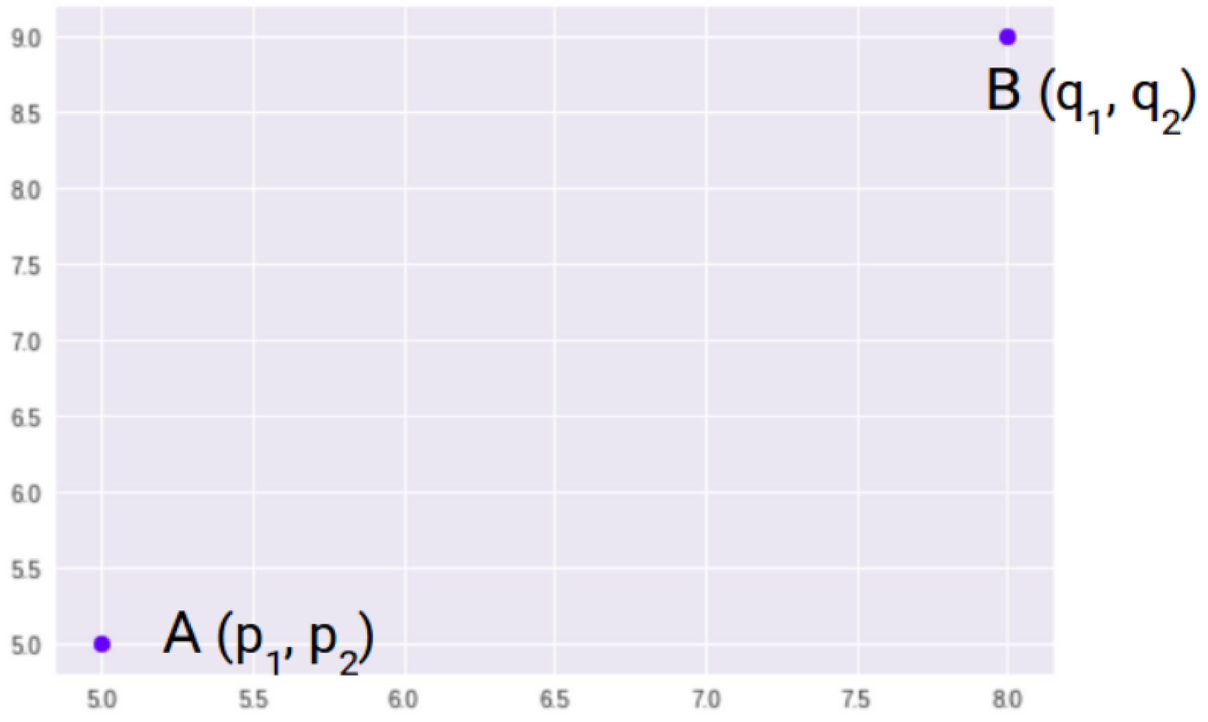
How do we define Similarity?

That is, **how do we define “distance” ?**

3. Distance

Widely used "distance" metric

1. Euclidean Distance
2. Manhattan Distance
3. Minkowski Distance
4. Hamming Distance



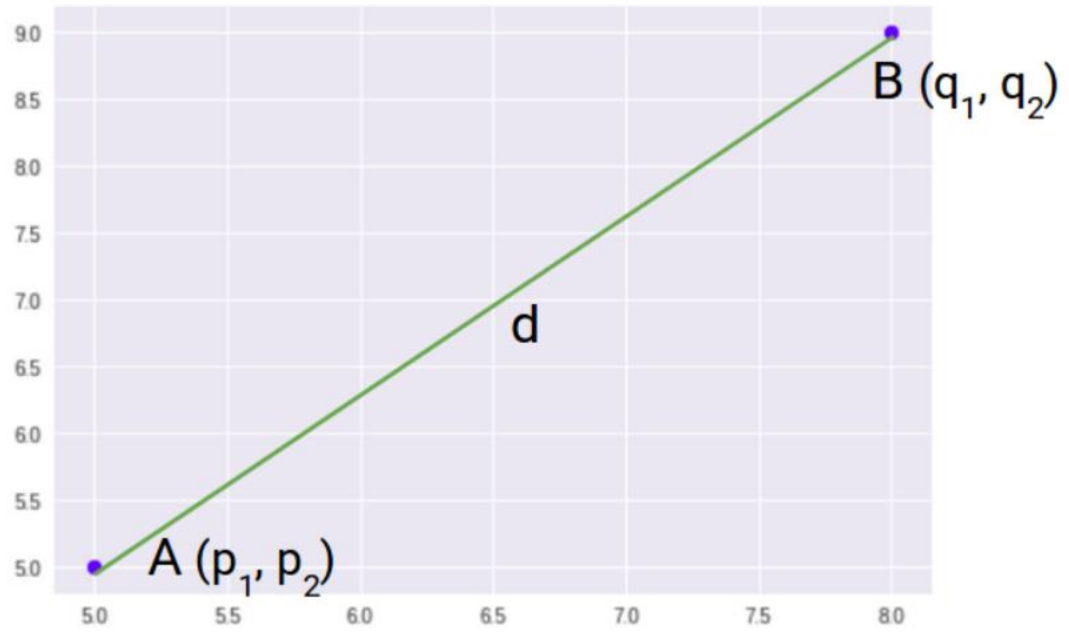
https://www.analyticsvidhya.com/blog/2020/02/4-types-of-distance-metrics-in-machine-learning/

3. Distance

1. Euclidean Distance

> *Euclidean Distance represents the shortest distance between two points.*

So, the Euclidean Distance between these two points A and B will be:



$$D_e = \left(\sum_{i=1}^{n} (p_i - q_i)^2 \right)^{1/2}$$

- n = number of dimensions
- pi, qi = data points

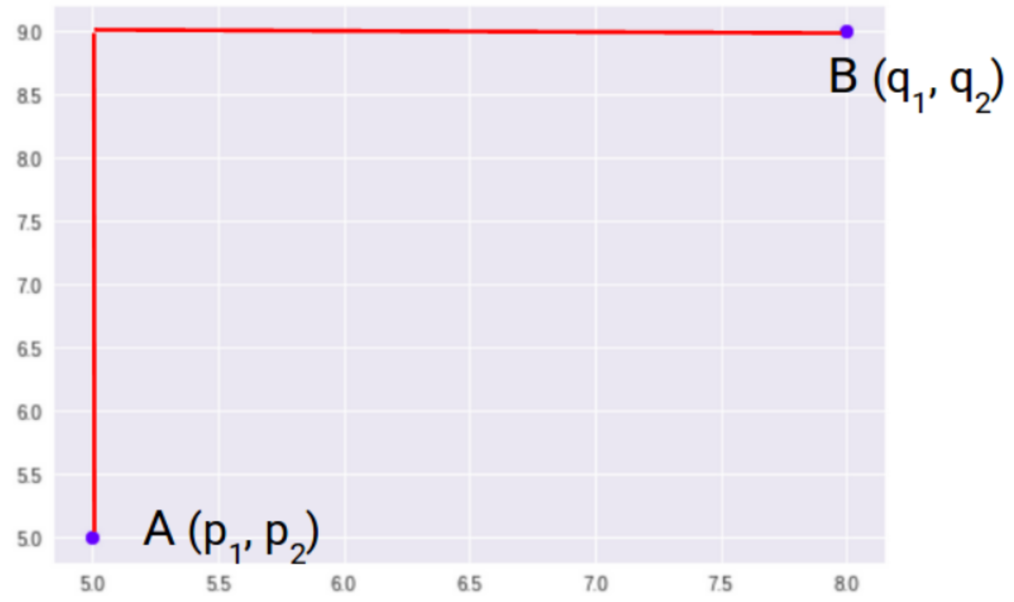
https://www.analyticsvidhya.com/blog/2020/02/4-types-of-distance-metrics-in-machine-learning/

3. Distance

2. Manhattan Distance

> *Manhattan Distance is the sum of absolute differences between points across all the dimensions.*

We can represent Manhattan Distance as:

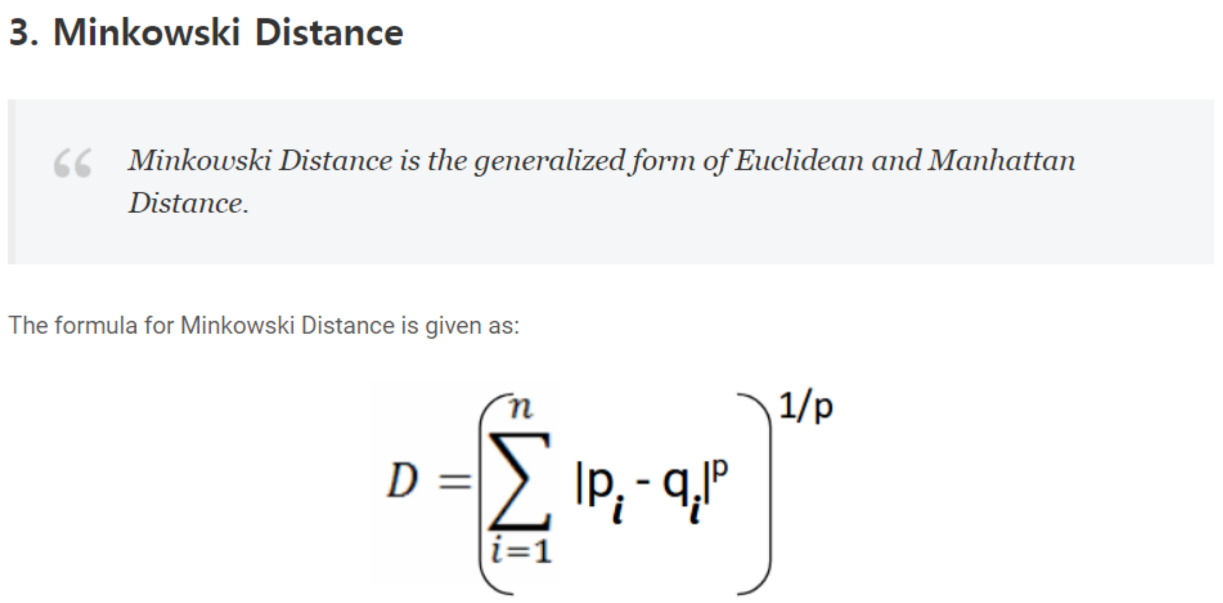


$$D_m = \sum_{i=1}^{n} |p_i - q_i|$$

- n = number of dimensions
- pi, qi = data points

https://www.analyticsvidhya.com/blog/2020/02/4-types-of-distance-metrics-in-machine-learning/

3. Distance

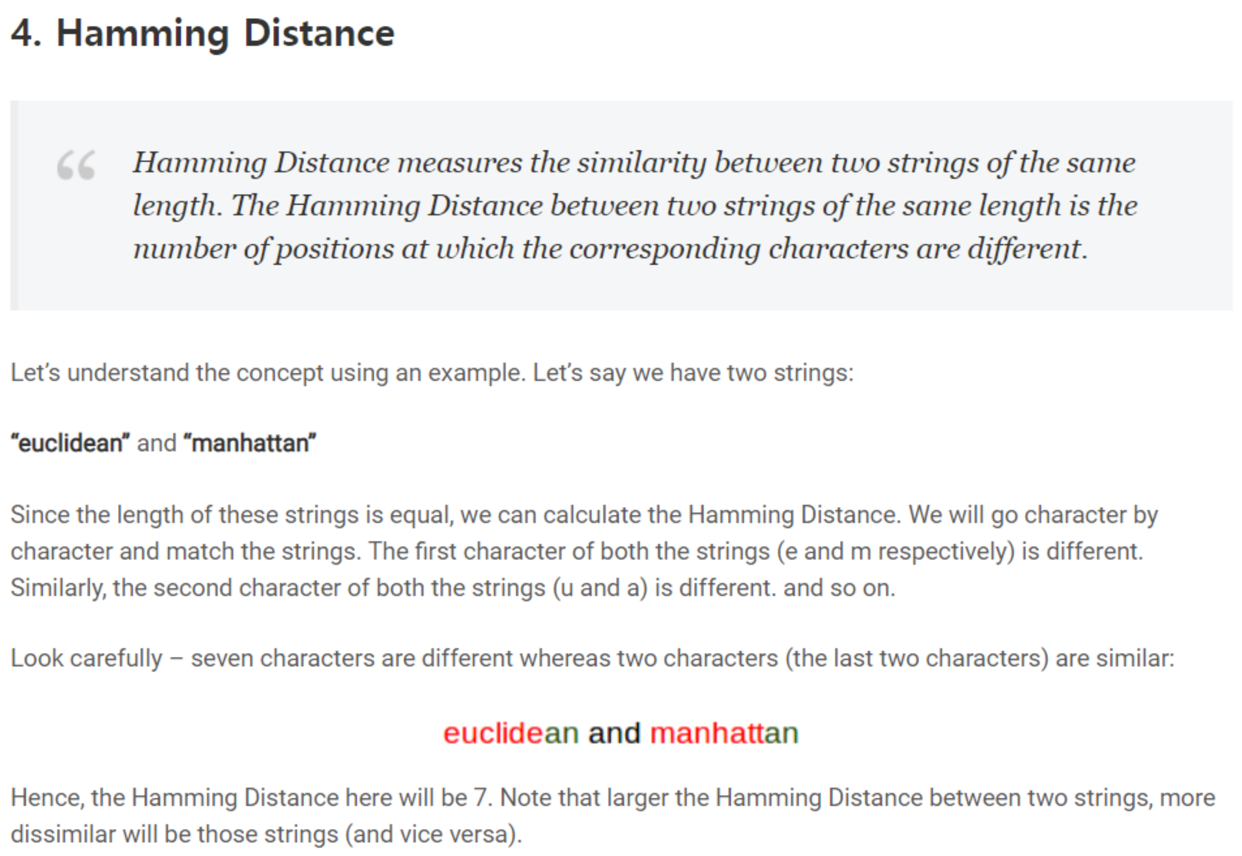


3. Minkowski Distance

> *Minkowski Distance is the generalized form of Euclidean and Manhattan Distance.*

The formula for Minkowski Distance is given as:

$$D = \left(\sum_{i=1}^{n} |p_i - q_i|^p \right)^{1/p}$$



4. Hamming Distance

> *Hamming Distance measures the similarity between two strings of the same length. The Hamming Distance between two strings of the same length is the number of positions at which the corresponding characters are different.*

Let's understand the concept using an example. Let's say we have two strings:

"euclidean" and **"manhattan"**

Since the length of these strings is equal, we can calculate the Hamming Distance. We will go character by character and match the strings. The first character of both the strings (e and m respectively) is different. Similarly, the second character of both the strings (u and a) is different. and so on.

Look carefully – seven characters are different whereas two characters (the last two characters) are similar:

euclidean and manhattan

Hence, the Hamming Distance here will be 7. Note that larger the Hamming Distance between two strings, more dissimilar will be those strings (and vice versa).

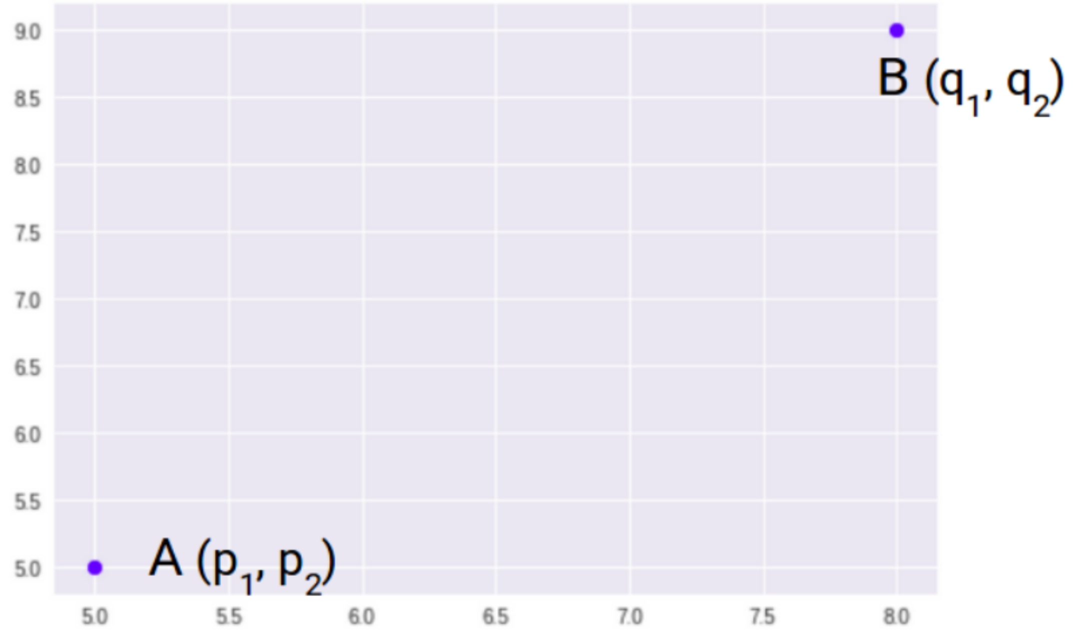
https://www.analyticsvidhya.com/blog/2020/02/4-types-of-distance-metrics-in-machine-learning/

3. Distance

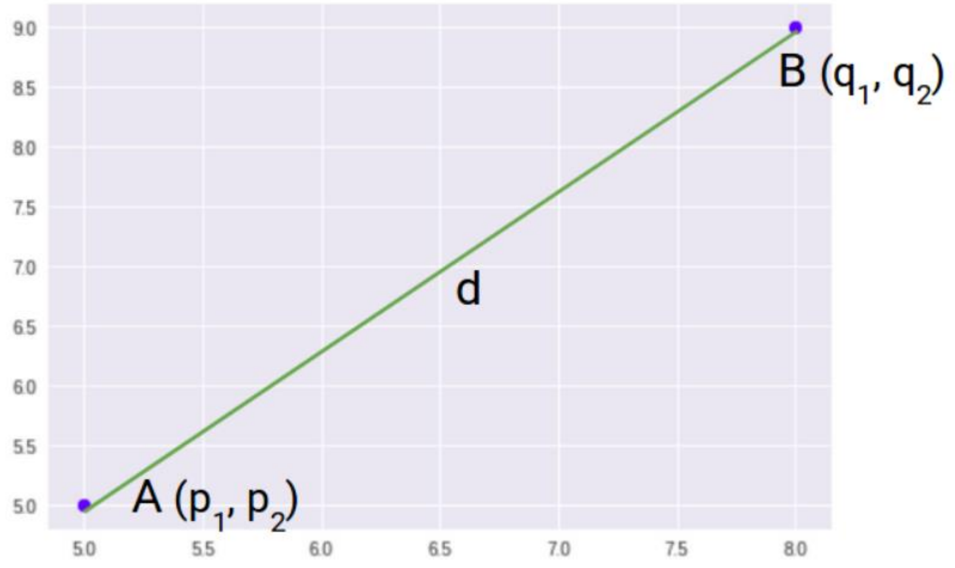
1. Euclidean Distance

> *Euclidean Distance represents the shortest distance between two points.*

Most machine learning algorithms including K-Means use this distance metric to measure the similarity between observations. Let's say we have two points as shown below:



So, the Euclidean Distance between these two points A and B will be:



$$D_e = \left(\sum_{i=1}^{n} (p_i - q_i)^2 \right)^{1/2}$$

- n = number of dimensions
- pi, qi = data points

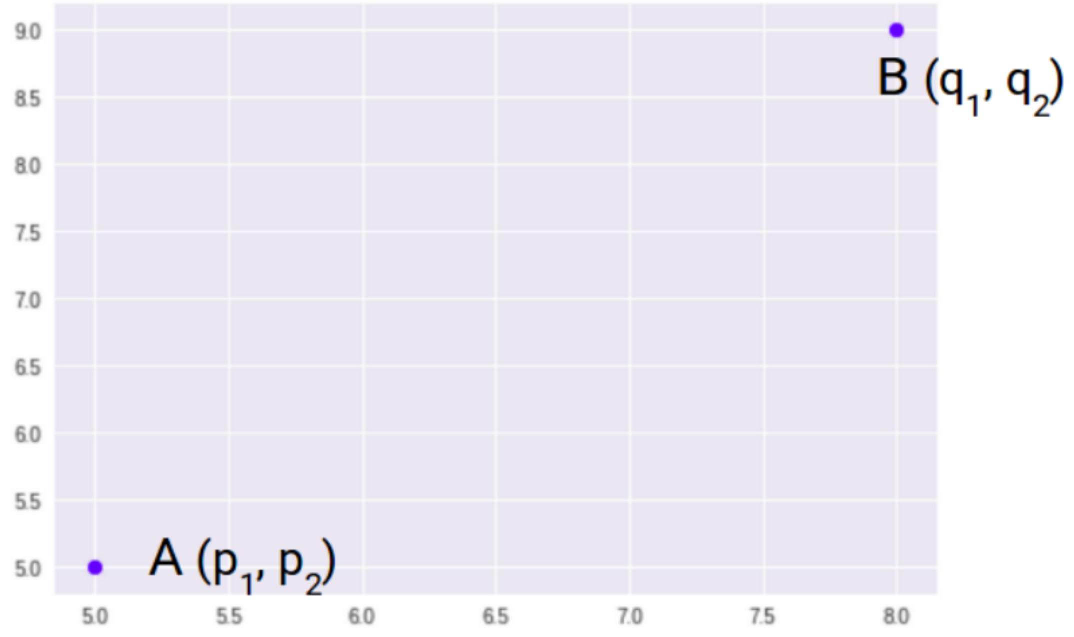
https://www.analyticsvidhya.com/blog/2020/02/4-types-of-distance-metrics-in-machine-learning/

3. Distance

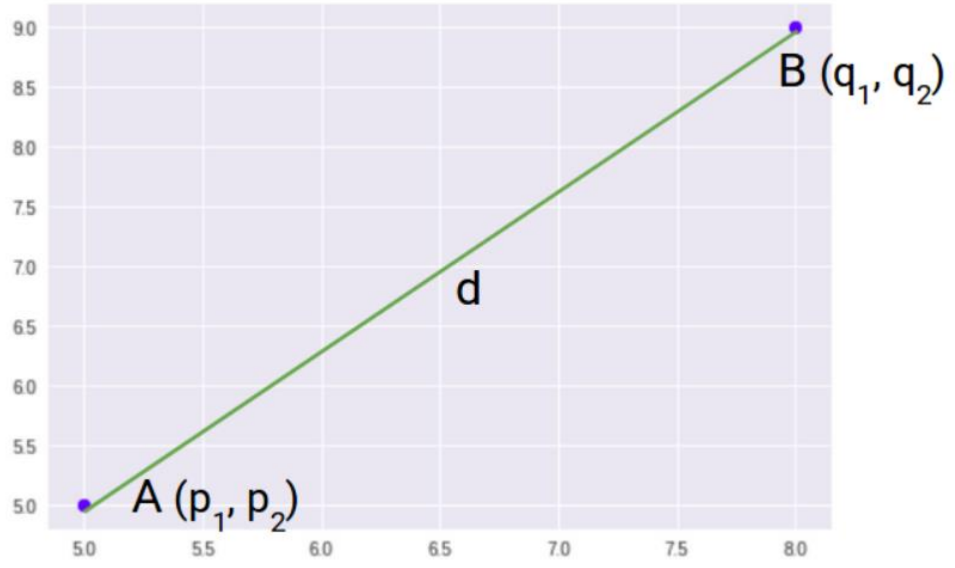
1. Euclidean Distance

> *Euclidean Distance represents the shortest distance between two points.*

Most machine learning algorithms including K-Means use this distance metric to measure the similarity between observations. Let's say we have two points as shown below:



So, the Euclidean Distance between these two points A and B will be:



$$D_e = \left(\sum_{i=1}^{n} (p_i - q_i)^2 \right)^{1/2}$$

- n = number of dimensions
- pi, qi = data points

Contents

1. Intro to Machine Learning
2. Intro to Clustering
3. Distance
4. K-Means Clustering
5. Choosing optimal number of clusters
6. Other Clustering methods
7. K-Means Clustering using Scikit-Learn (Python)

4. K-Means Clustering

- The most basic method of Clustering (most widely-known)
- Only for “Numerical” data
- Use “Euclidean” distance
- “K” = number of clusters

4. K-Means Clustering

Introduction of K-means Algorithm

Key 1) partition our data into K groups

Key 2) Each group has a ‘centroid’

(= center of each group)

Key 3) Finding ‘centroid’ & assigning other data into centroid

(by reducing SSE(Sum of Square Error))

4. K-Means Clustering

Introduction of K-means Algorithm

Key 1) partition our data into K groups

Key 2) Each group has a ‘centroid’

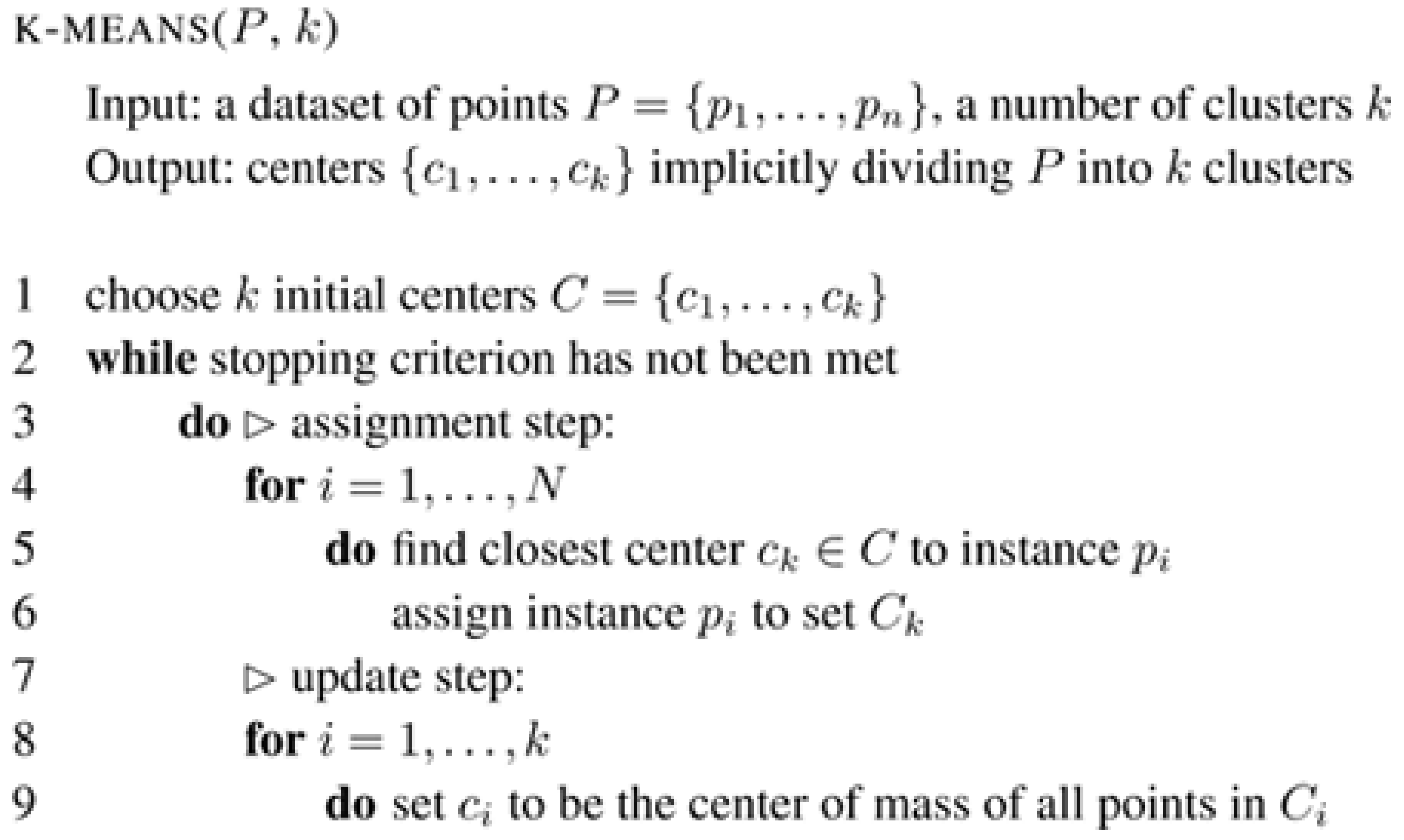
(= center of each group)

Key 3) Finding ‘centroid’ & assigning other data into centroid

(by reducing SSE(Sum of Square Error))

How does it work?

4. K-Means Clustering



```
K-MEANS($P$, $k$)
    Input: a dataset of points $P = \{p_1, \dots, p_n\}$, a number of clusters $k$
    Output: centers $\{c_1, \dots, c_k\}$ implicitly dividing $P$ into $k$ clusters

    choose $k$ initial centers $C = \{c_1, \dots, c_k\}$
    while stopping criterion has not been met
        do ▷ assignment step:
           for $i = 1, \dots, N$
               do find closest center $c_k \in C$ to instance $p_i$
                  assign instance $p_i$ to set $C_k$
           ▷ update step:
           for $i = 1, \dots, k$
               do set $c_i$ to be the center of mass of all points in $C_i$
```

4. K-Means Clustering

1) Choose “K” (number of clusters)

4. K-Means Clustering

1) Choose “K” (number of clusters)

2) Initialize “centroid” (randomly)

   - ex) N = 1000, k=7 -> initialize 7 centroids

4. K-Means Clustering

1) Choose “K” (number of clusters)

2) Initialize “centroid” (randomly)

   - ex) N = 1000, k=7 -> initialize 7 centroids

3) [Assignment] Assign non-centroid data into the closest centroid

   - ex) assign 993 data into its closest centroid (among the 7 centroids)

     (closest = minimum distance)

4. K-Means Clustering

1) Choose “K” (number of clusters)

2) Initialize “centroid” (randomly)

   - ex) N = 1000, k=7 -> initialize 7 centroids

3) [Assignment] Assign non-centroid data into the closest centroid

   - ex) assign 993 data into its closest centroid (among the 7 centroids)

     (closest = minimum distance)

4) [Update] Update the centroid of each cluster (= change the centroid)

   (change the centroid to the “center of mass of all points” in each cluster)

4. K-Means Clustering

1) Choose “K” (number of clusters)

2) Initialize “centroid” (randomly)

   - ex) N = 1000, k=7 -> initialize 7 centroids

3) [Assignment] Assign non-centroid data into the closest centroid

   - ex) assign 993 data into its closest centroid (among the 7 centroids)

     (closest = minimum distance)

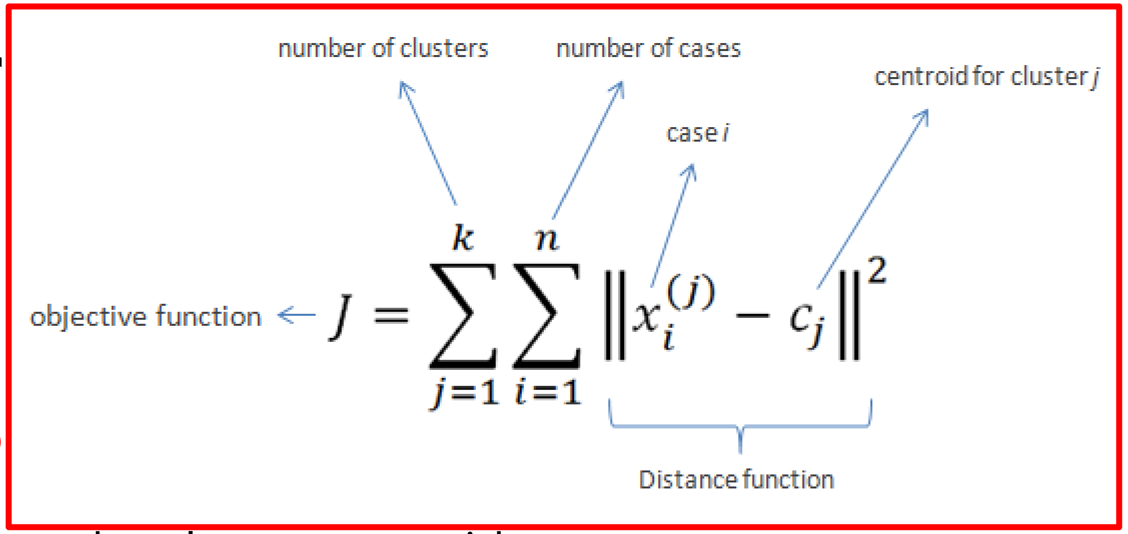
4) [Update] Update the centroid of each cluster (= change the centroid)

   (change the centroid to the “center of mass of all points” in each cluster)

5) Repeat 3), 4) until stopping criterion reaches

4. K-Means Cluster

SSE(Sum of Square Error)



$$J = \sum_{j=1}^{k}\sum_{i=1}^{n}\underbrace{\left\|x_i^{(j)} - c_j\right\|^2}_{\text{Distance function}}$$

1) Choose “K” (number of clusters)

2) Initialize “centroid” (randomly)

   - ex) N = 1000, k=7 -> initialize 7 centro

3) [Assignment] Assign non-centroid data into the closest centroid

   - ex) assign 993 data into its closest centroid (among the 7 centroids)

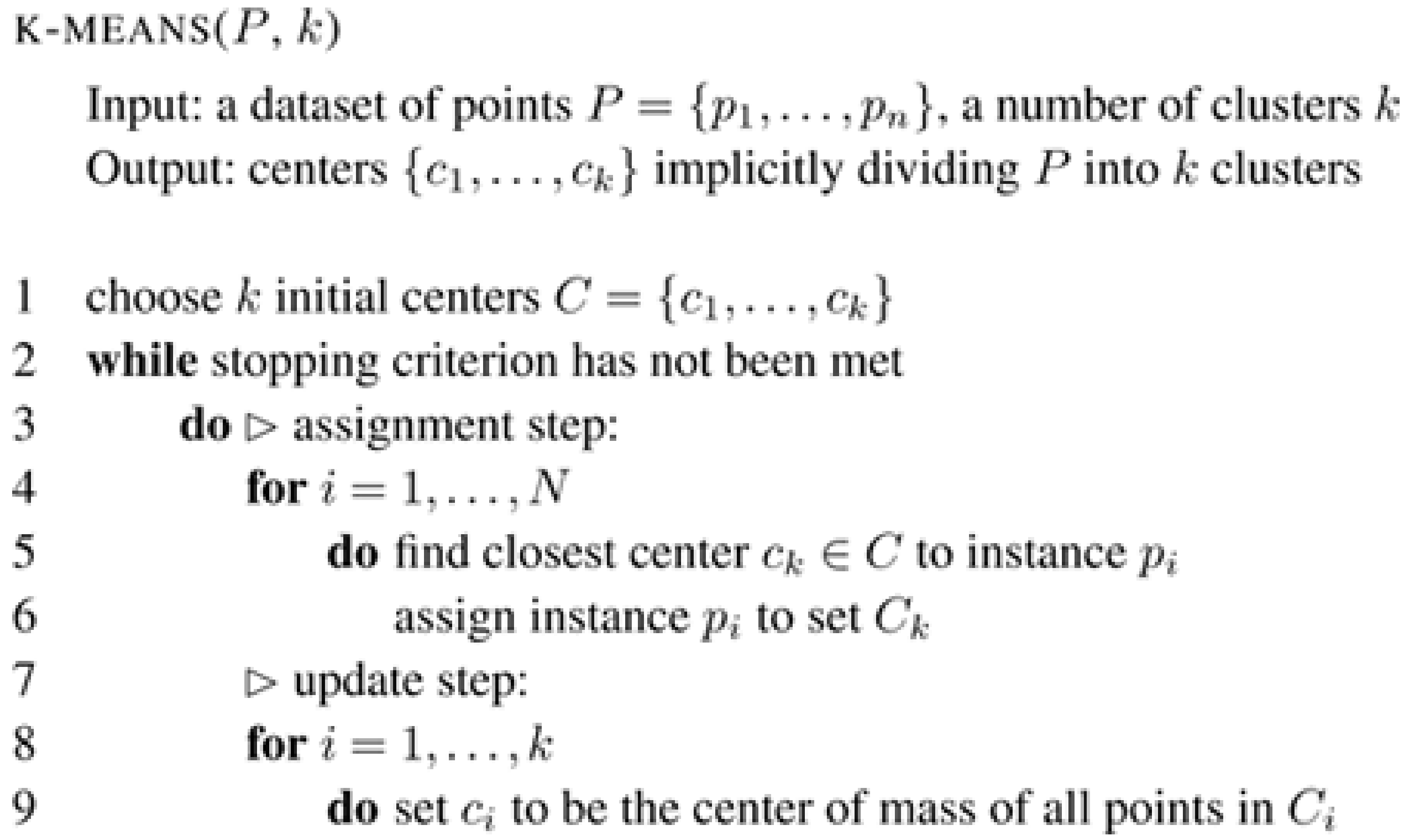
     (closest = minimum distance)

4) [Update] Update the centroid of each cluster (= change the centroid)

   (change the centroid to the “center of mass of all points” in each cluster)

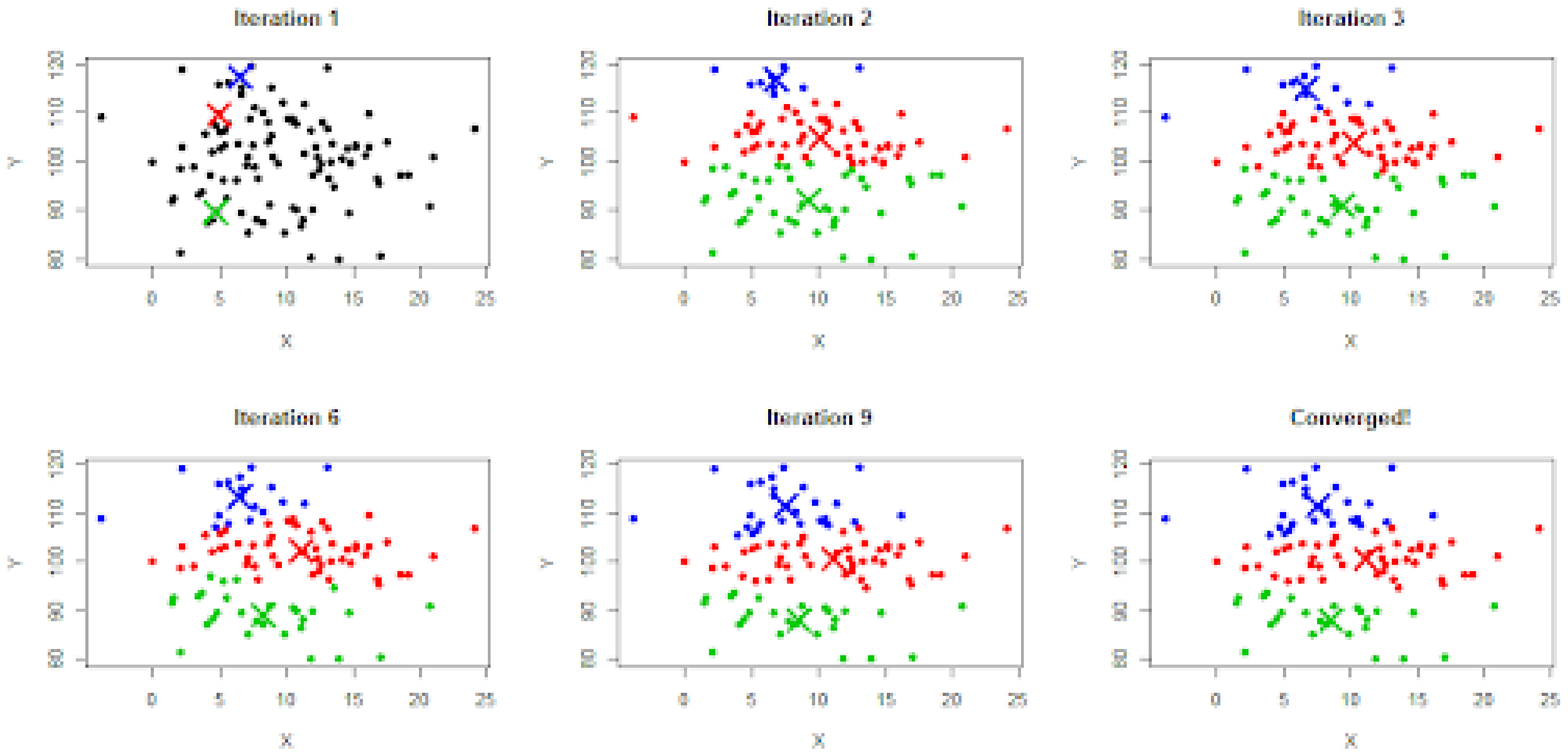
5) Repeat 3), 4) until stopping criterion reaches

4. K-Means Clustering



```
K-MEANS($P$, $k$)
    Input: a dataset of points $P = \{p_1, \ldots, p_n\}$, a number of clusters $k$
    Output: centers $\{c_1, \ldots, c_k\}$ implicitly dividing $P$ into $k$ clusters

1   choose $k$ initial centers $C = \{c_1, \ldots, c_k\}$
2   while stopping criterion has not been met
3       do ▷ assignment step:
4           for $i = 1, \ldots, N$
5               do find closest center $c_k \in C$ to instance $p_i$
6                   assign instance $p_i$ to set $C_k$
7           ▷ update step:
8           for $i = 1, \ldots, k$
9               do set $c_i$ to be the center of mass of all points in $C_i$
```

4. K-Means Clustering



4. K-Means Clustering

[Pros]

- Scalability – can use at large data
- Low Computing Cost
- Easy to understand

4. K-Means Clustering

[Pros]

- Scalability – can use at large data
- Low Computing Cost
- Easy to understand

[Cons]

- Have to choose number of clusters -> Elbow Method
- dependent on initial values -> by running several times with different initial value
- Vulnerable to outliers (use “mean” & “SSE”)

Contents

1. Intro to Machine Learning
2. Intro to Clustering
3. Distance
4. K-Means Clustering
5. Choosing optimal number of clusters
6. Other Clustering methods
7. K-Means Clustering using Scikit-Learn (Python)

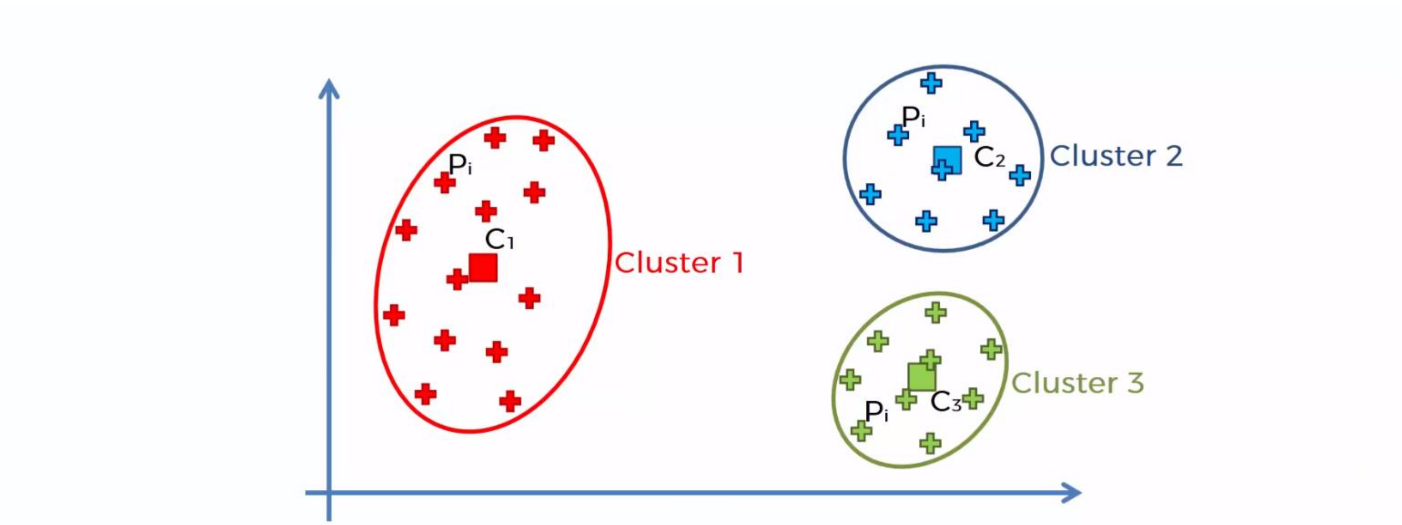
5. Choosing optimal number of clusters

What does it mean by “Clustering is done well!” ?

5. Choosing optimal number of clusters

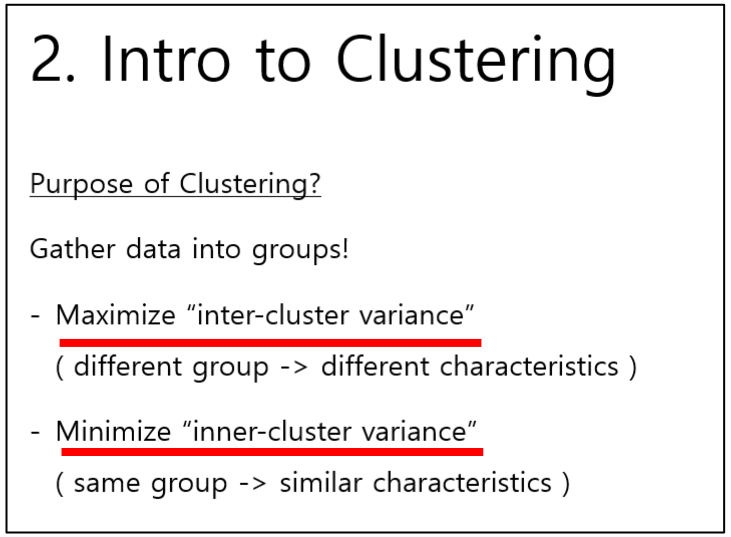
What does it mean by "Clustering is done well!" ?

- "Close" within a cluster!
- "Far" between different clusters!



$$\text{WCSS} = \sum_{P_i \text{ in Cluster 1}} \text{distance}(P_i, C_1)^2 + \sum_{P_i \text{ in Cluster 2}} \text{distance}(P_i, C_2)^2 + \sum_{P_i \text{ in Cluster 3}} \text{distance}(P_i, C_3)^2$$

# Slide 5



2. Intro to Clustering

Purpose of Clustering?

Gather data into groups!

- Maximize "inter-cluster variance"
  (different group -> different characteristics)
- Minimize "inner-cluster variance"
  (same group -> similar characteristics)

5. Choosing optimal number of clusters

- **Within Cluster Sums of Squares :** $$\text{WSS} = \sum_{i=1}^{N_C} \sum_{x \in C_i} d(\mathbf{x}, \bar{\mathbf{x}}_{\mathrm{C_i}})^2$$
- **Between Cluster Sums of Squares:** $$\text{BSS} = \sum_{i=1}^{N_C} |C_i| \cdot d(\bar{\mathbf{x}}_{\mathrm{C_i}}, \bar{\mathbf{x}})^2$$

$\boldsymbol{C_i}$ = Cluster, N_c = # clusters, $\overline{\boldsymbol{x}}_{\boldsymbol{c_i}}$= Cluster centroid, $\overline{\boldsymbol{x}}$ = Sample Mean

5. Choosing optimal number of clusters

Widely used criterion for choosing optimal "K"

- **Within Cluster Sums of Squares :** $$\mathrm{WSS} = \sum_{i=1}^{N_C} \sum_{x \in C_i} d(\mathbf{x}, \bar{\mathbf{x}}_{C_i})^2$$

- **Between Cluster Sums of Squares:** $$\mathrm{BSS} = \sum_{i=1}^{N_C} |C_i| \cdot d(\bar{\mathbf{x}}_{C_i}, \bar{\mathbf{x}})^2$$

Size of C_i = Cluster, N_c = # clusters, $\overline{x}_{c_i}$ = Cluster centroid, $\overline{x}$ = Sample Mean

We have to minimize(?) WSS!

5. Choosing optimal number of clusters

<u>1. Elbow Curve</u>

As “K” increases, “WSS” decreases!

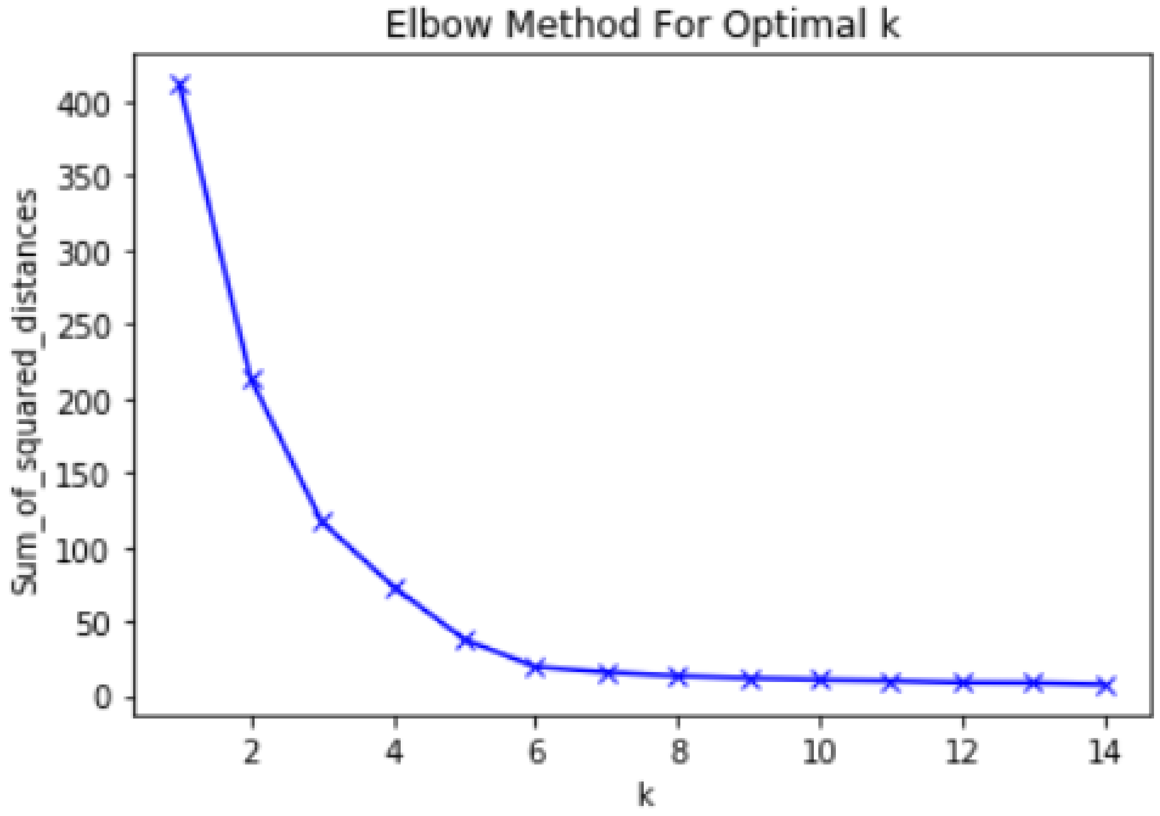
1000 data, 1000 clusters?? No need to do clustering….

5. Choosing optimal number of clusters

1. Elbow Curve

As “K” increases, “WSS” decreases!

1000 data, 1000 clusters?? No need to do clustering….

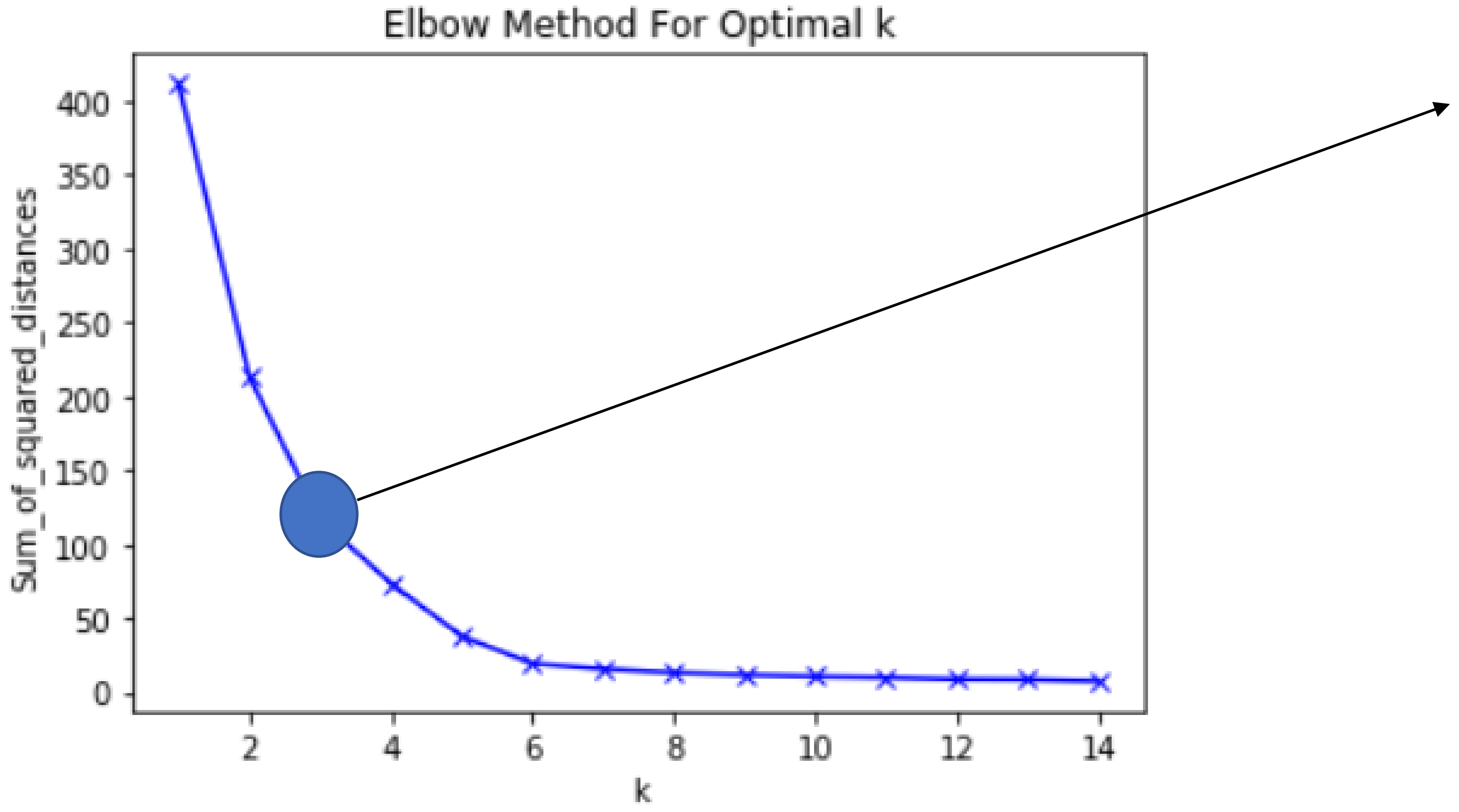


5. Choosing optimal number of clusters

<u>1. Elbow Curve</u>

As "K" increases, "WSS" decreases!

1000 data, 1000 clusters?? No need to do clustering….



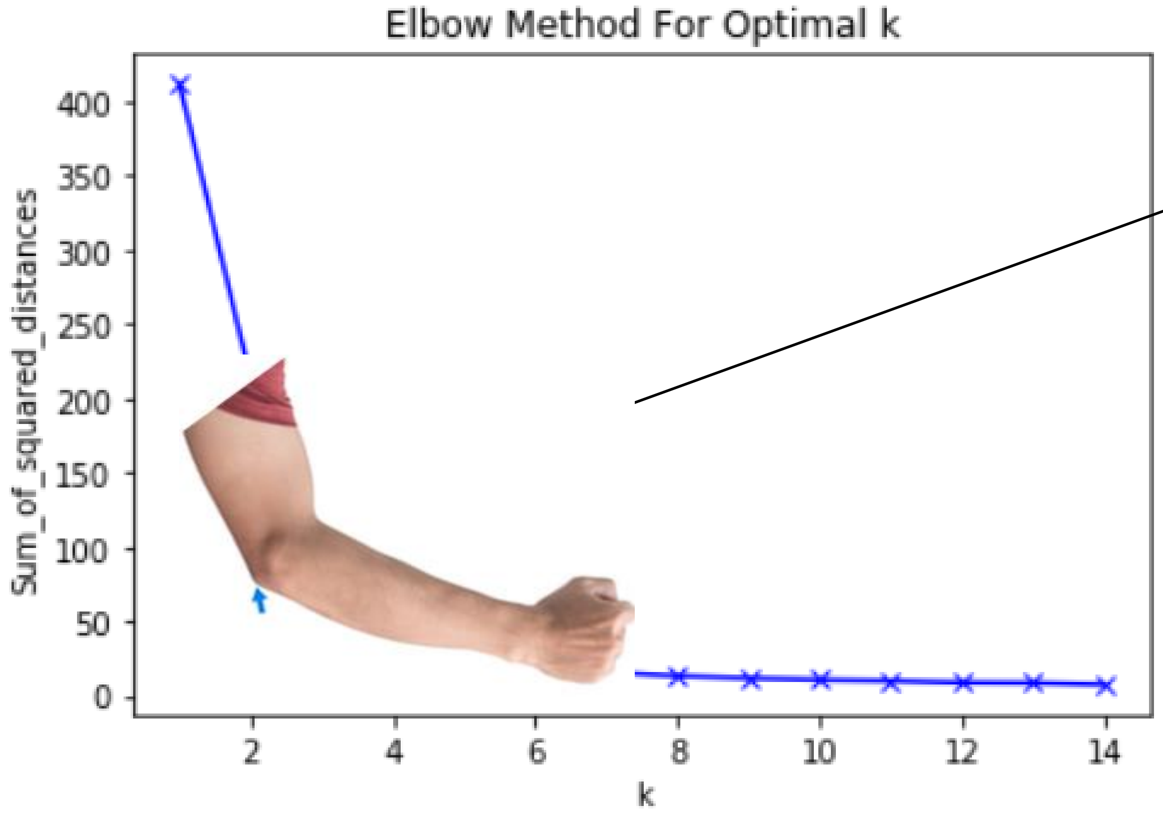
Elbow (still subjective…)

5. Choosing optimal number of clusters

1. Elbow Curve

As “K” increases, “WSS” decreases!

1000 data, 1000 clusters?? No need to do clustering….



Elbow (still subjective…)

5. Choosing optimal number of clusters

2. Silhouette Score

a(i) : average of distance of i th data, with same clusters

b(i) : min(average of distances of i th data, with other clusters)

(= average of distances of i th data, with closest cluster)

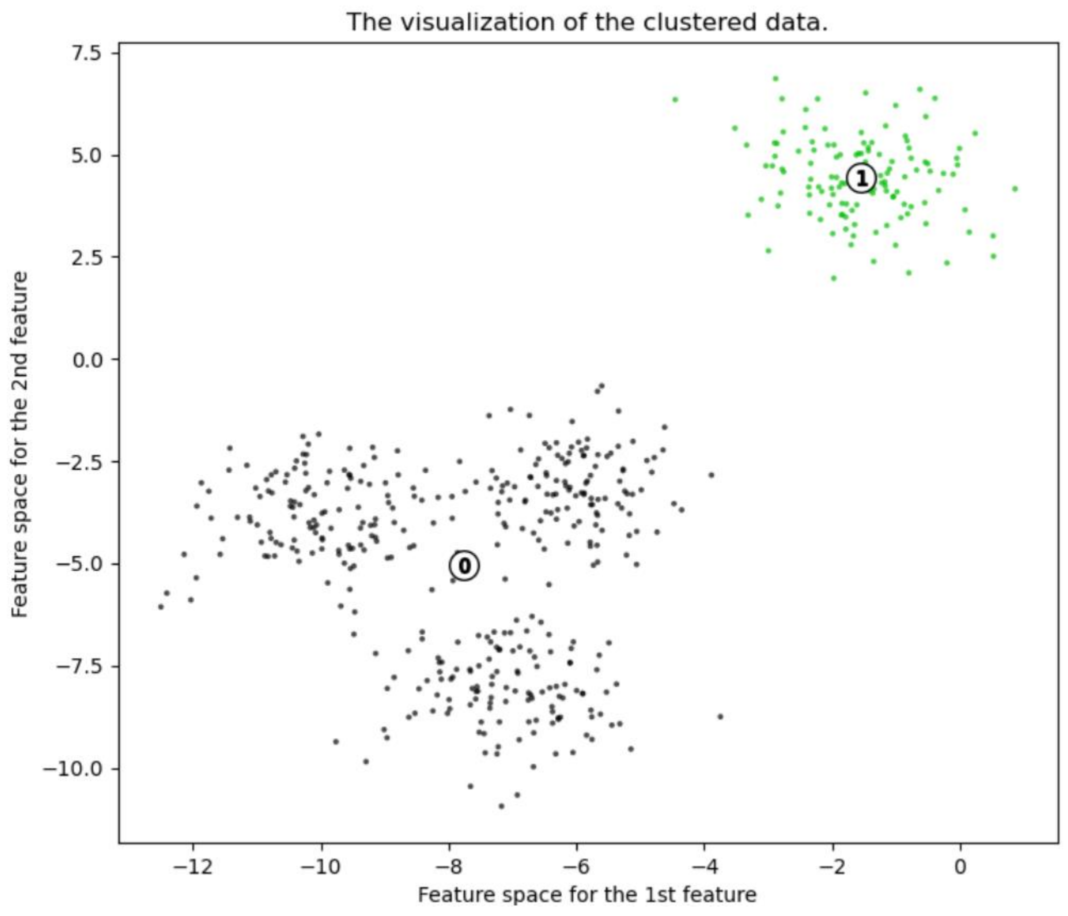
The bigger score, the better clustering

- Best : a(i) = 0 -> s(i) = 1
- Worst : b(i) = 0 -> s(i) = -1

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

5. Choosing optimal number of clusters

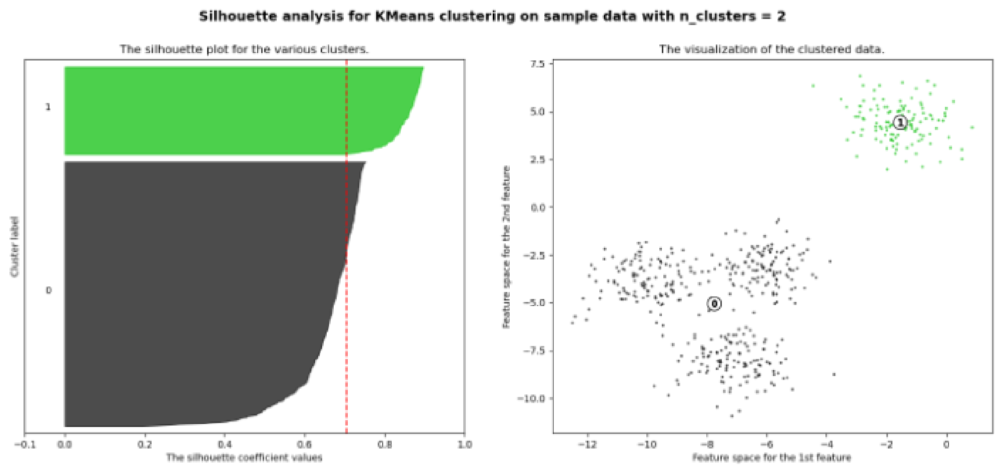
2. Silhouette Score



Which “K” seems to be reasonable?

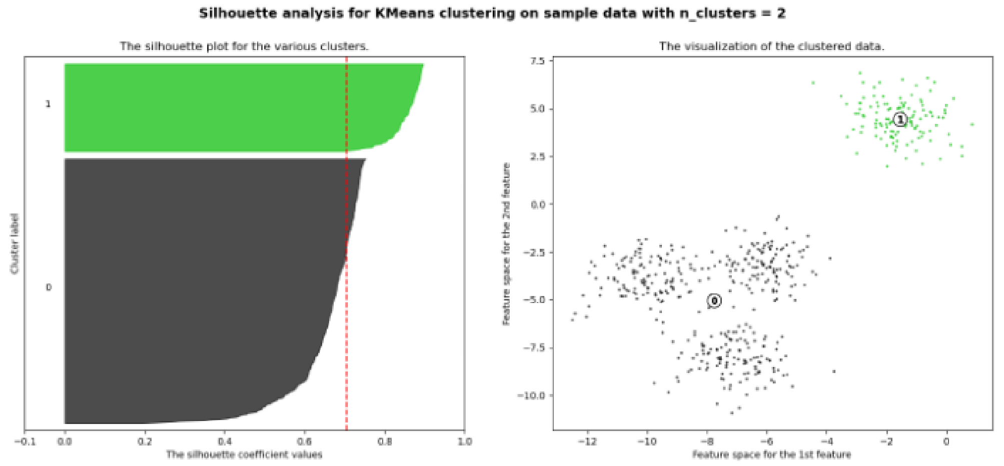
5. Choosing optimal number of clusters

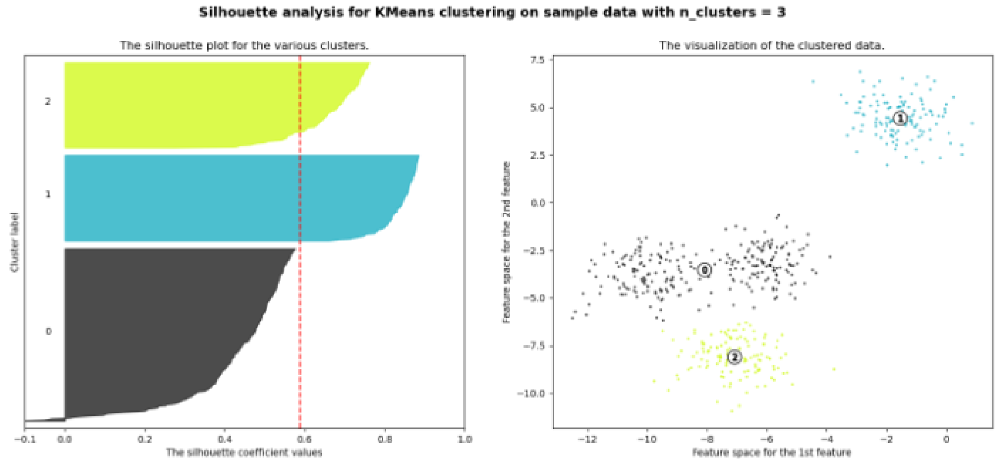
<u>2. Silhouette Score</u>



5. Choosing optimal number of clusters

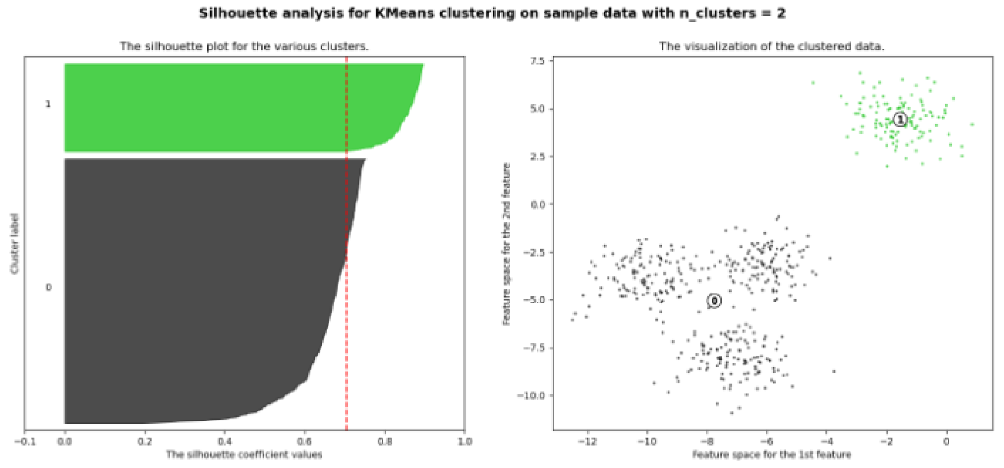
2. Silhouette Score

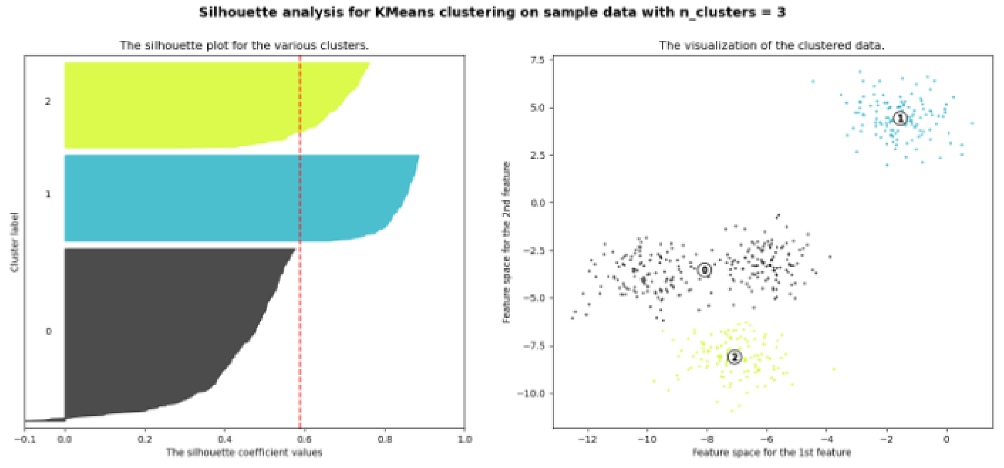


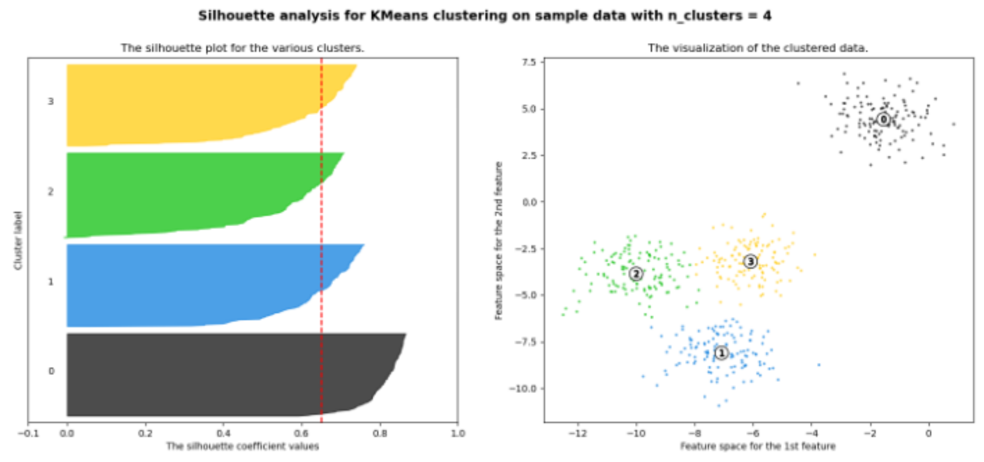


5. Choosing optimal number of clusters

2. Silhouette Score

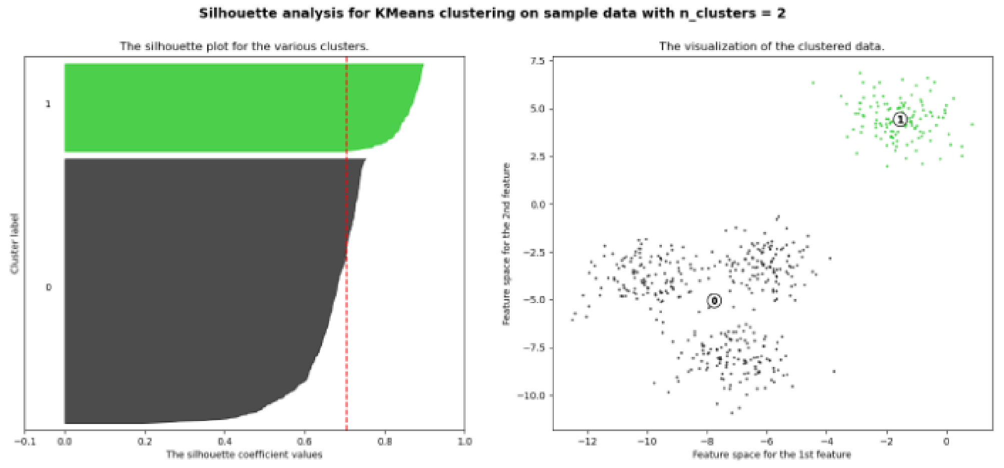


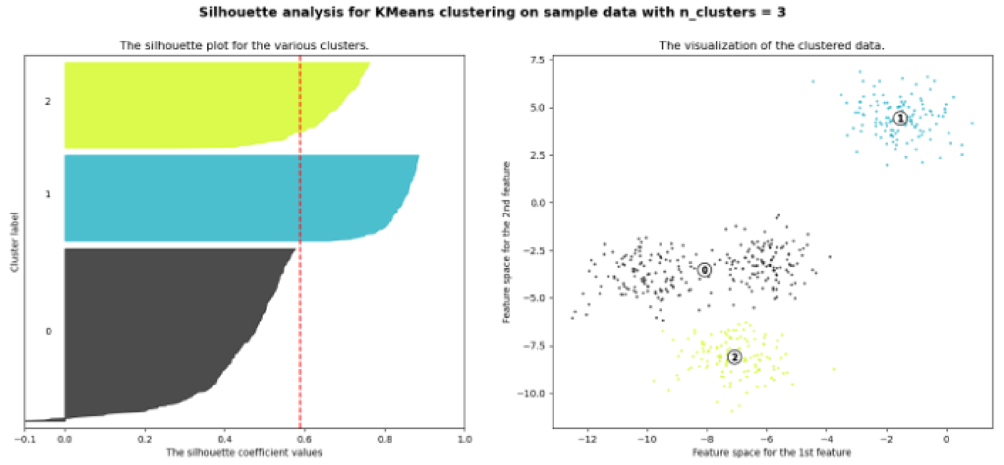


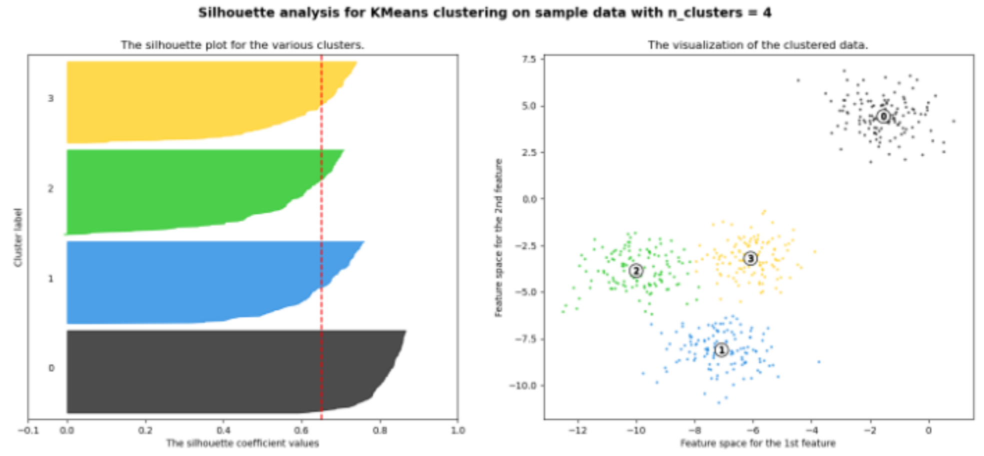


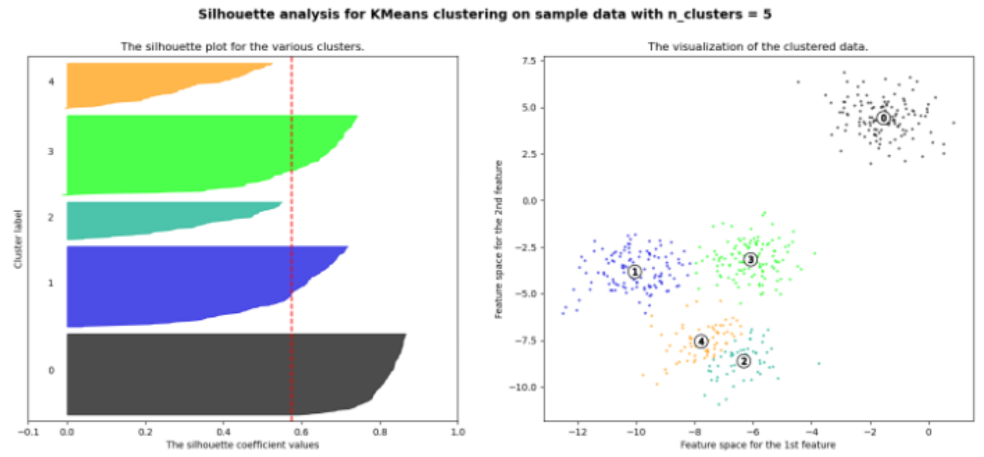
5. Choosing optimal number of clusters

2. Silhouette Score



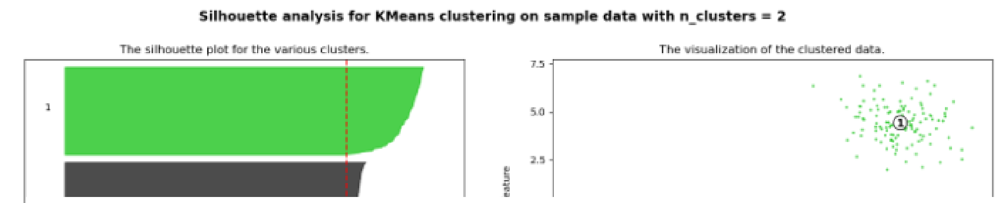


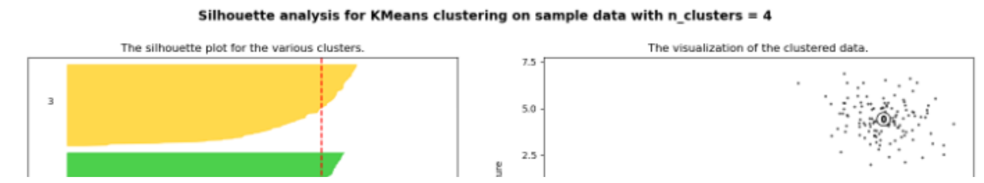




5. Choosing optimal number of clusters

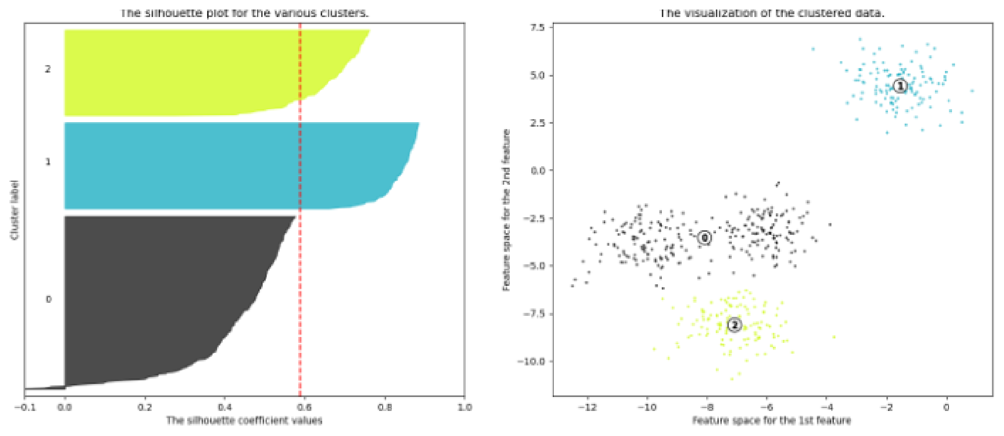
2. Silhouette Score

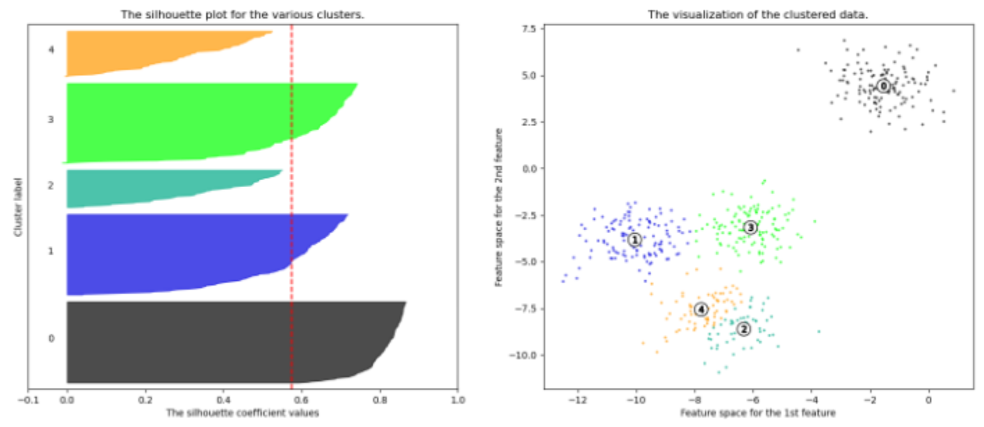




Out:

```
For n_clusters = 2 The average silhouette_score is : 0.7049787496083262
For n_clusters = 3 The average silhouette_score is : 0.5882004012129721
For n_clusters = 4 The average silhouette_score is : 0.6505186632729437
For n_clusters = 5 The average silhouette_score is : 0.5745566973301872
For n_clusters = 6 The average silhouette_score is : 0.4387644975296138
```





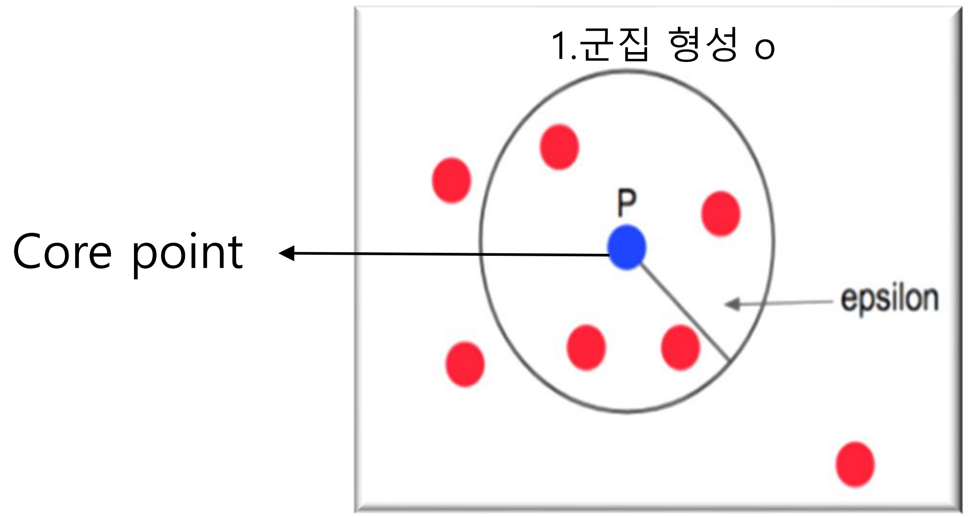
Contents

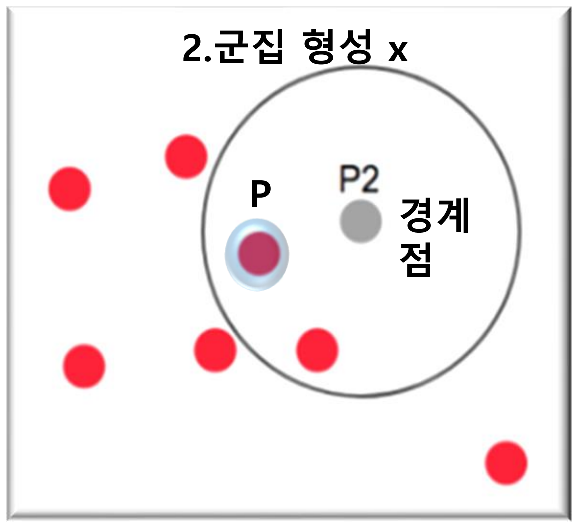
1. Intro to Machine Learning
2. Intro to Clustering
3. Distance
4. K-Means Clustering
5. Choosing optimal number of clusters
6. Other Clustering methods
7. K-Means Clustering using Scikit-Learn (Python)

6. Other Clustering Methods

1. DBSCAN (Density Based Spatial Clustering of Application with Noise)

1) if more than 'n' points inside distance 'epsilon' -> cluster (O)



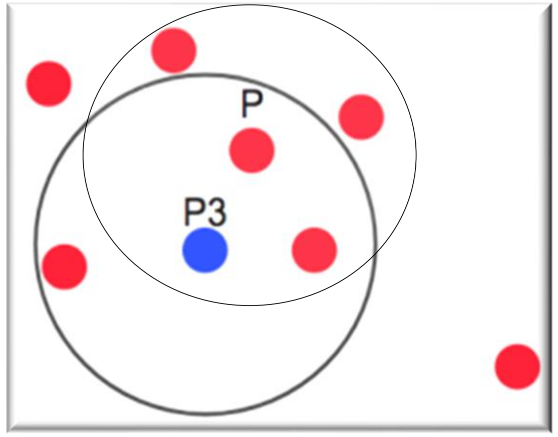


6. Other Clustering Methods

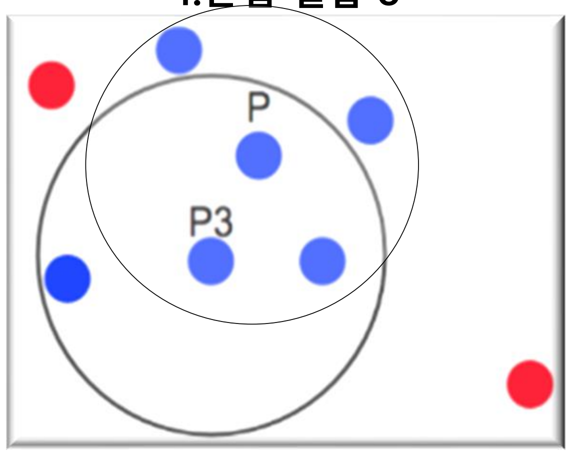
1. DBSCAN (Density Based Spatial Clustering of Application with Noise)

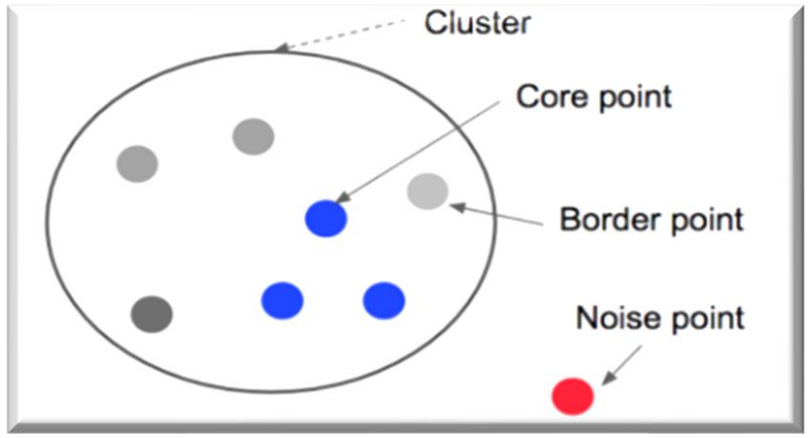
2) If one 'core point' is inside epsilon distance from other 'core point' -> JOIN!

3.군집 형성 o



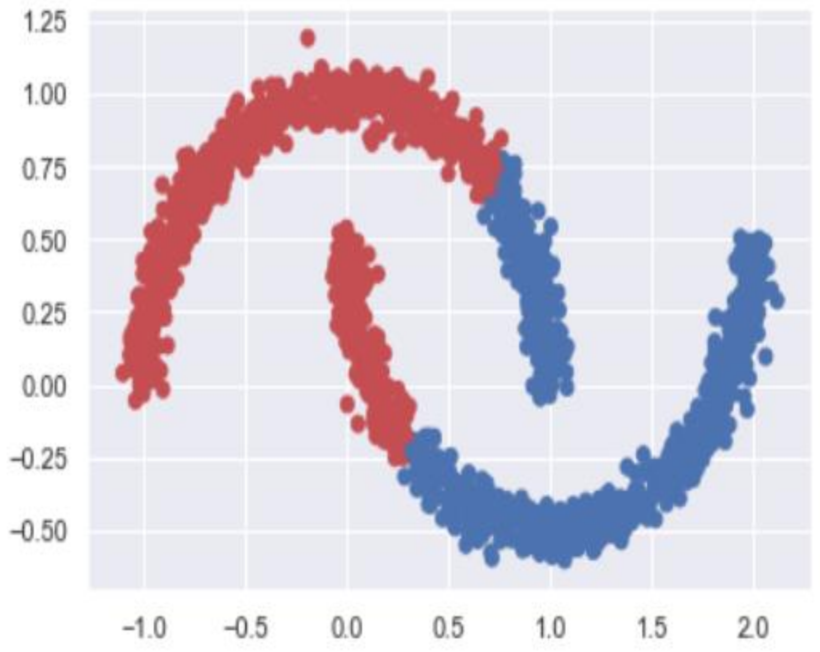
4.군집 결합 o



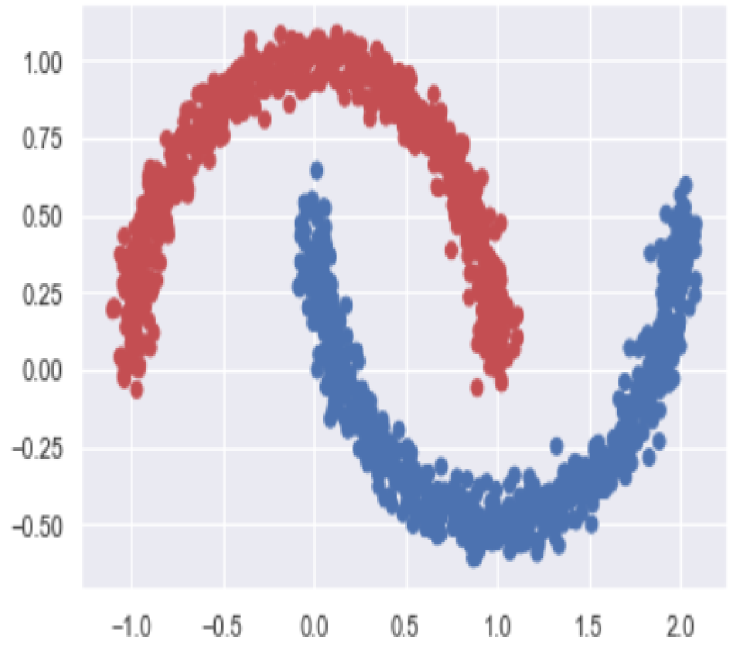


6. Other Clustering Methods

1. DBSCAN (Density Based Spatial Clustering of Application with Noise)



K-Means



DBSCAN

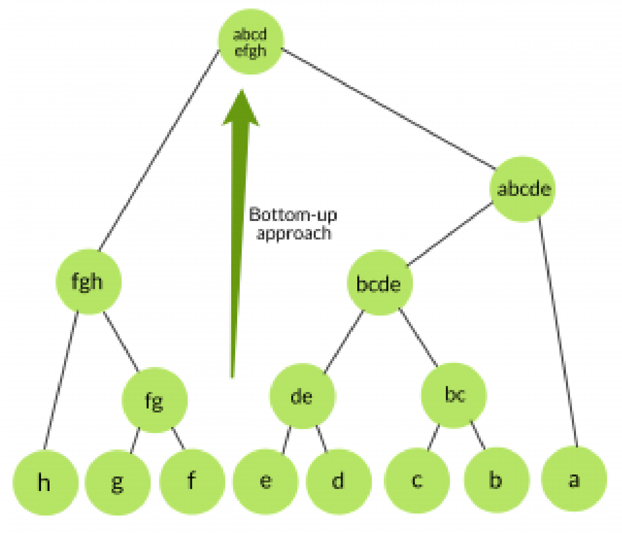
6. Other Clustering Methods

<u>2. Hierarchical Clustering</u>

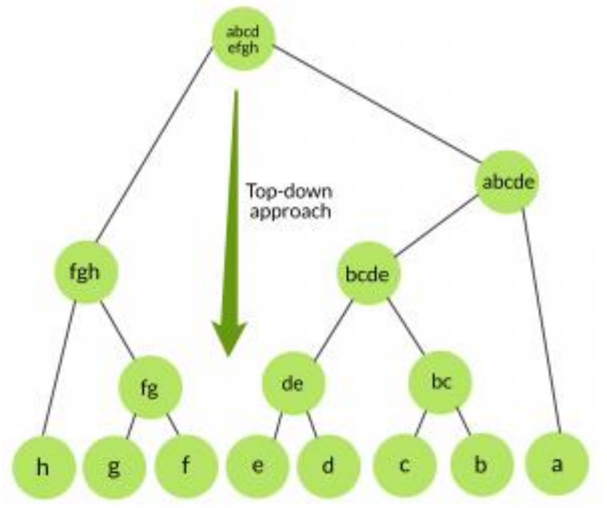
Do not require to pre-specify # of clusters

1) Agglomerative Clustering
   - bottom-up approach

2) Divisive hierarchical clustering
   - top-down approach



Hierarchical agglomerative clustering



Hierarchical divisive clustering

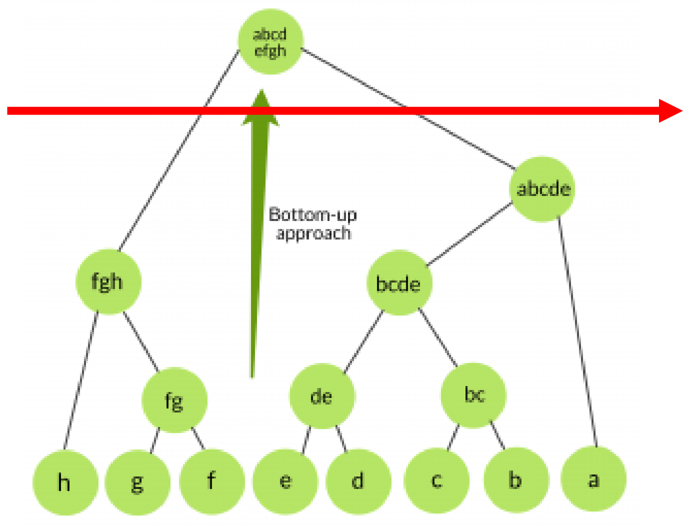
6. Other Clustering Methods

2. Hierarchical Clustering

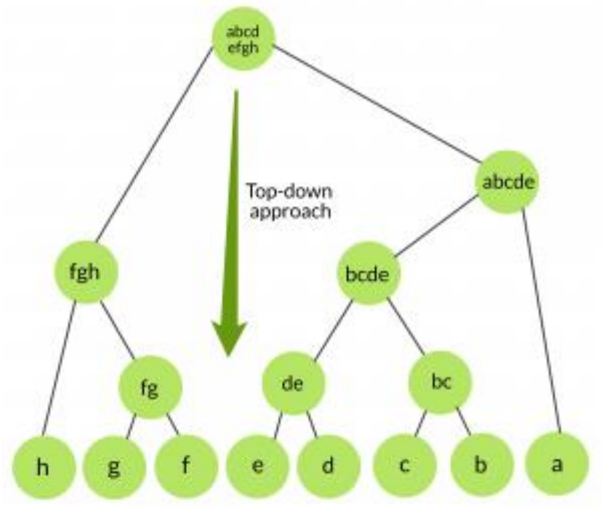
Do not require to pre-specify # of clusters

1) Agglomerative Clustering
   - bottom-up approach

2) Divisive hierarchical clustering
   - top-down approach

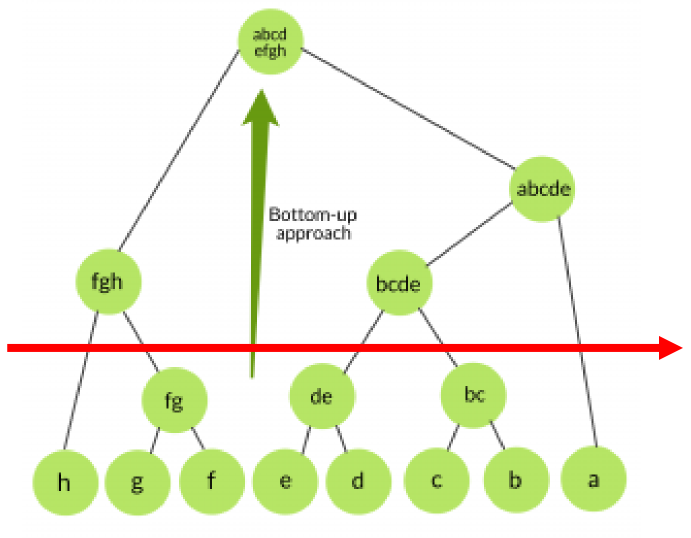


Hierarchical agglomerative clustering

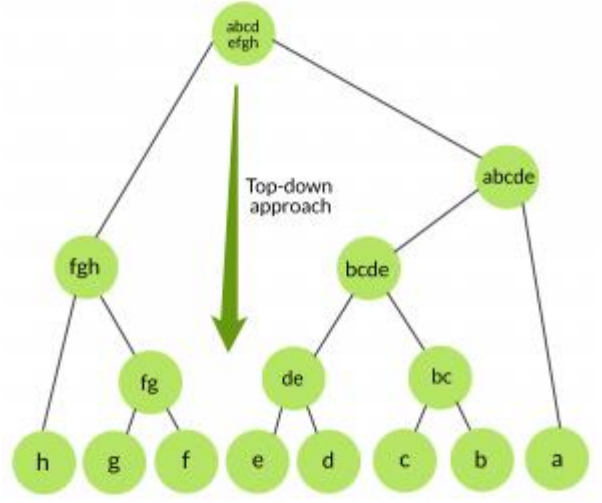


Hierarchical divisive clustering

6. Other Clustering Methods



Hierarchical agglomerative clustering



Hierarchical divisive clustering

<u>2. Hierarchical Clustering</u>

Do not require to pre-specify # of clusters

1) Agglomerative Clustering
   - bottom-up approach

2) Divisive hierarchical clustering
   - top-down approach

6. Other Clustering Methods

<u>3. K-mode</u>

- For “Categorical” Variable

<u>4. K-Prototypes</u>

- For “Numerical + Categorical” Variable

Contents

1. Intro to Machine Learning
2. Intro to Clustering
3. Distance
4. K-Means Clustering
5. Choosing optimal number of clusters
6. Other Clustering methods
7. K-Means Clustering using Scikit-Learn (Python)

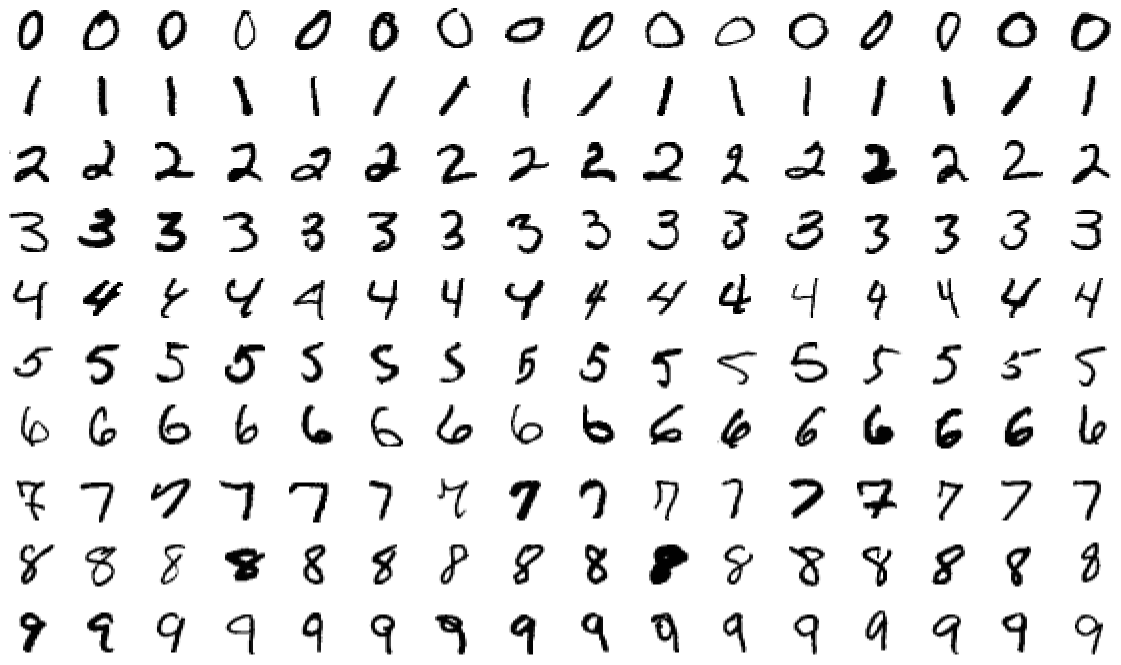
7. K-Means Clustering using Scikit-Learn (Python)

- Step 1) Data & Package 불러오기
- Step 2) Data 스케일(단위) 조정

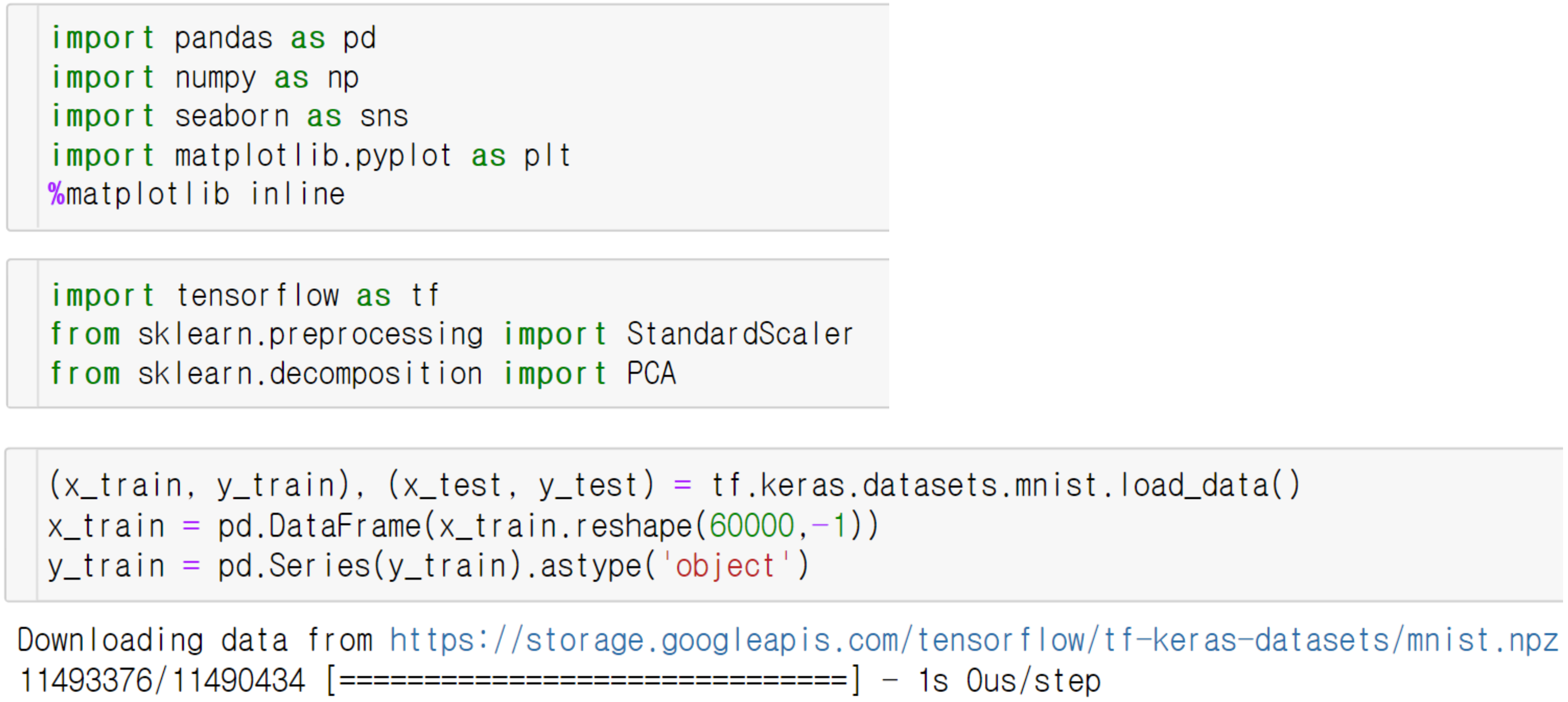
  (ex. MinMaxScaler(0~1), Standard Scaler(mean=0, var=1)
- Step 2.5) 이상치(outlier) 제거
- Step 3) 차원 축소
- Step 4) Clustering

예시 소개

- 사용할 데이터 : MNIST 손글씨 데이터
- 데이터 크기 : (60000, 784) (28x28 pixel 데이터)
- 실제 MNIST데이터는 Y값(글씨가 0~9 중 어느 숫자에 해당하는지)가 있지만,
  이를 사용하지 않고 오직 X(픽셀 정보)만 사용해서 비슷한 데이터를 군집화할 것이다.



Step 1) Data & Package 불러오기



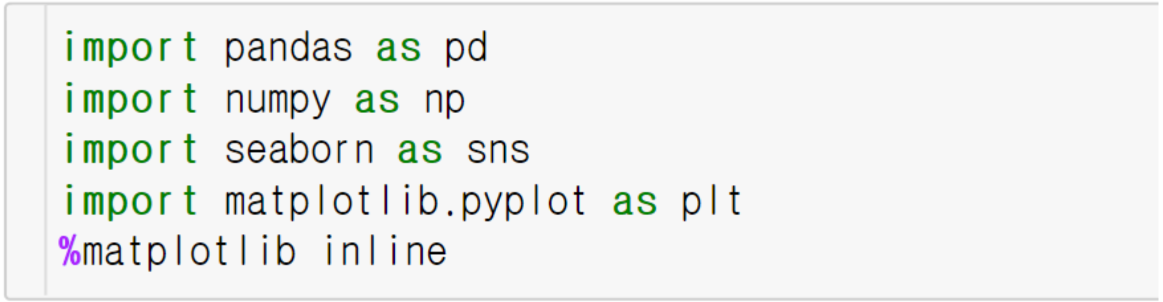
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```python
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

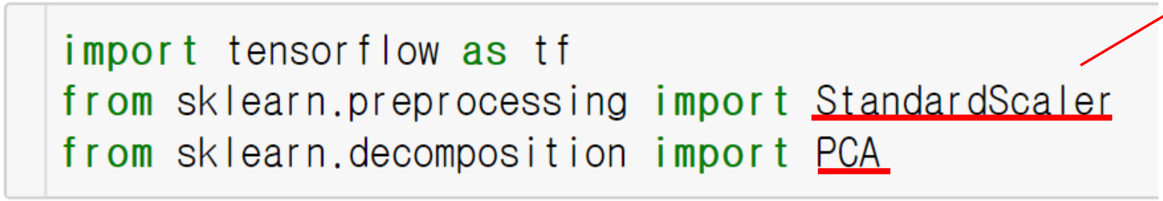
```python
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train = pd.DataFrame(x_train.reshape(60000,-1))
y_train = pd.Series(y_train).astype('object')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 1s 0us/step
```

Step 1) Data & Package 불러오기

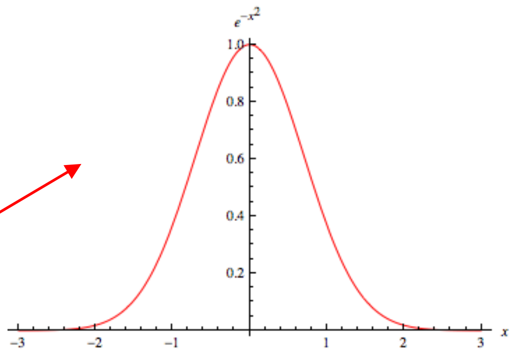


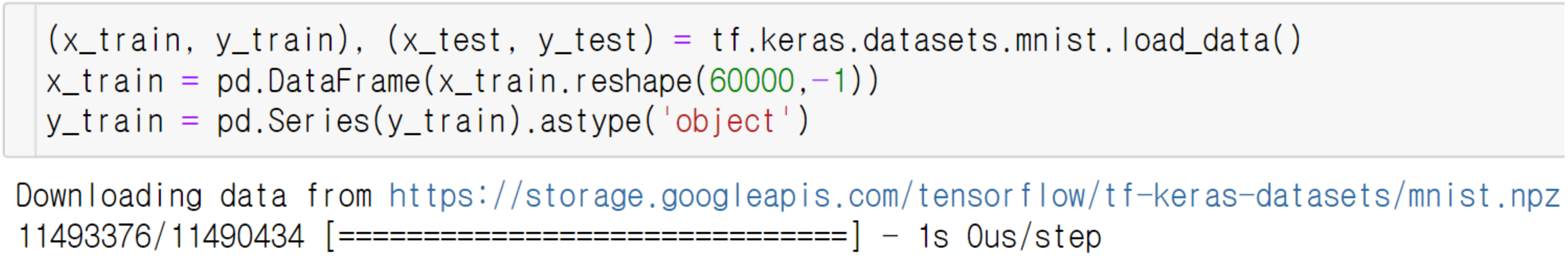
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```



```
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

표준 정규분포화

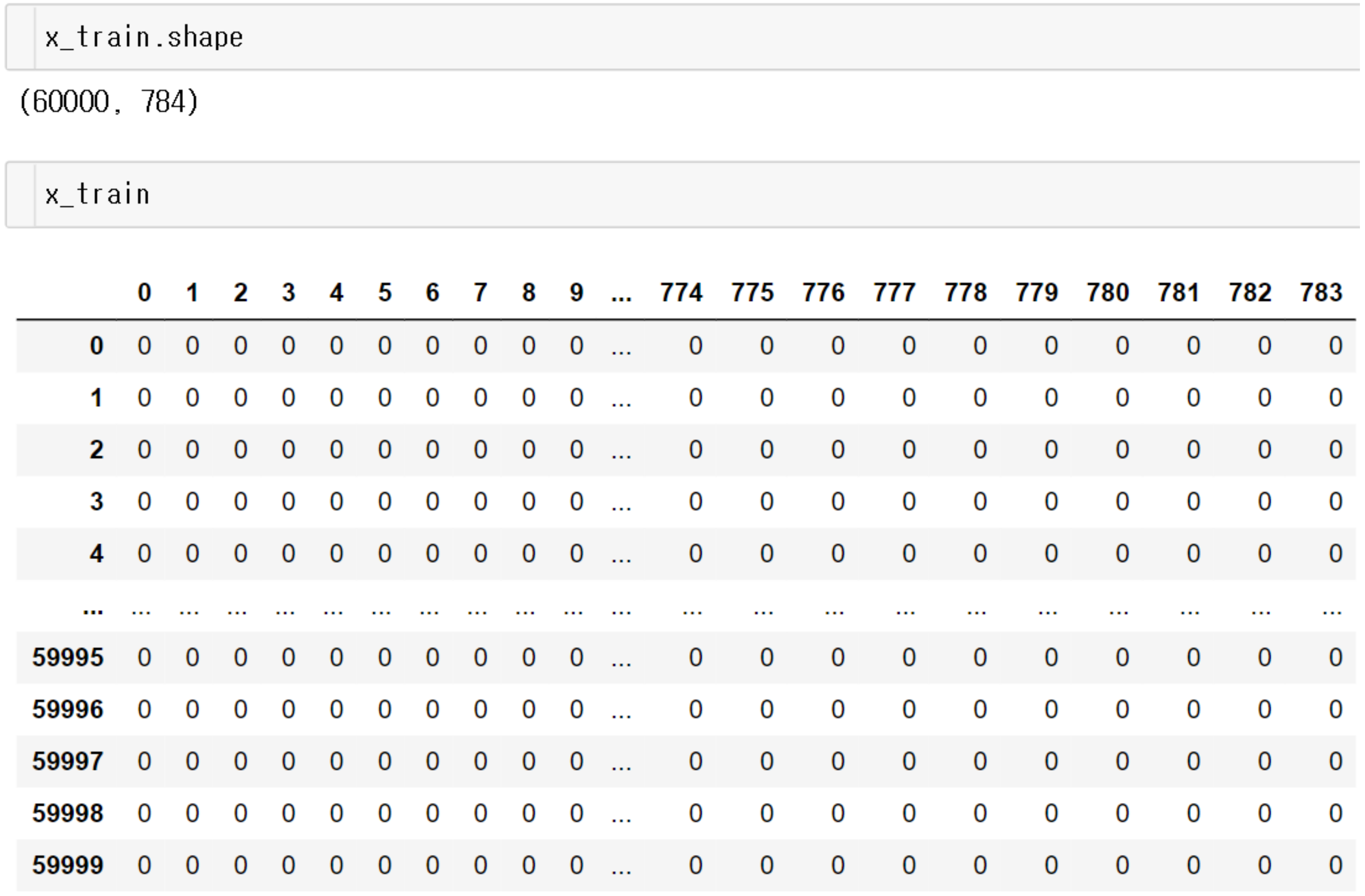




```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train = pd.DataFrame(x_train.reshape(60000,-1))
y_train = pd.Series(y_train).astype('object')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 1s 0us/step
```

Step 1) Data 불러오기

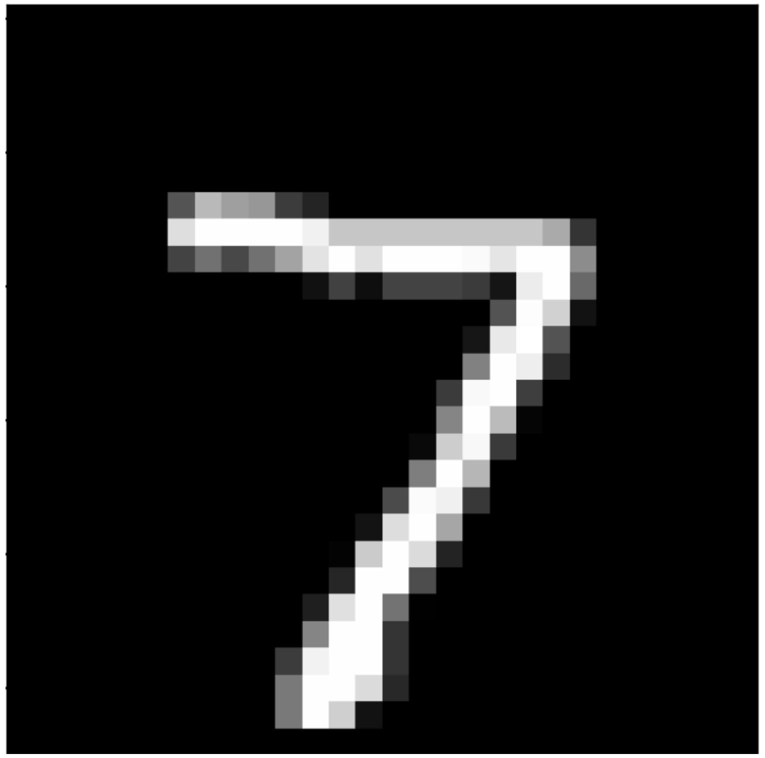


```
x_train.shape
```

```
(60000, 784)
```

```
x_train
```

	0	1	2	3	4	5	6	7	8	9	...	774	775	776	777	778	779	780	781	782	783
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	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
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
59995	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
59996	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
59997	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
59998	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
59999	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0



Step 2) Data 스케일(단위) 조정

PCA를 하기전에, 반드시 거쳐야 하는 과정!

(특히나, 변수들 간의 Scale에 차이가 큰 경우)

사람ID	키	몸무게	시력 평균	…	특징 D
사람 1	170	60	2.0		
사람 2	175	65	1.8		
사람 3	168	58	-2.0		
사람 4	180	70	0.6		
…	…	…	…		
…	…	…	…		
사람 999	177	66	0.3		

곧 바로

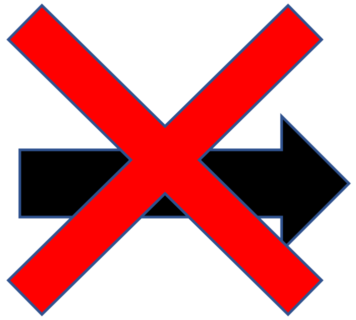
차원 축소 ?

Step 2) Data 스케일(단위) 조정

PCA를 하기전에, 반드시 거쳐야 하는 과정!

(특히나, 변수들 간의 Scale에 차이가 큰 경우)

사람ID	키	몸무게	시력 평균	…	특징 D
사람 1	170	60	2.0		
사람 2	175	65	1.8		
사람 3	168	58	-2.0		
사람 4	180	70	0.6		
…	…	…	…		
…	…	…	…		
사람 999	177	66	0.3		



곧 바로

차원 축소 ?

Step 2) Data 스케일(단위) 조정

대표적인 2가지 스케일 조정 방법

1) Standard Scaler

- 각 변수를 표준 정규분포로 만들어줌 (평균이 0, 분산이 1)

(sklearn.preprocessing.StandardScaler)

2) MinMax Scaler

- 각 변수 내에서, 최대값을 1로, 최소값을 0으로 만들어줌

(sklearn.preprocessing.MinMaxScaler)

Step 2) Data 스케일(단위) 조정

PCA를 하기전에, 반드시 거쳐야 하는 과정!

(특히나, 변수들 간의 Scale에 차이가 큰 경우)

Scaling 이전

사람ID	키	몸무게	시력 평균	...	특징 D
사람 1	170	60	2.0		
사람 2	175	65	1.8		
사람 3	168	58	-2.0		
사람 4	180	70	0.6		
...	...	...	...		
...	...	...	...		
사람 999	177	66	0.3		

Scaling 이후

사람ID	키	몸무게	시력 평균	...	특징 D
사람 1	0.4	-0.3	1.5		
사람 2	1.1	0.5	1.2		
사람 3	-0.3	-1.2	-1.2		
사람 4	1.3	0.8	0.8		
...	...	...	...		
...	...	...	...		
사람 999	1.2	0.7	0.3		

(Standard Scaler를 사용해서)

Step 2) Data 스케일(단위) 조정

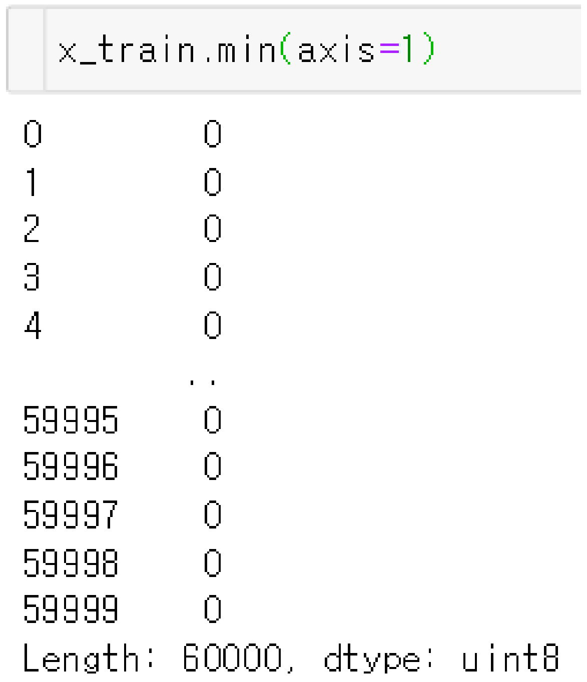
차원 축소를 하기에 앞서서, 거의 필수적인 과정!

[TIP]

- 이 뿐만 아니라 많은 ML 문제에서 , 이와 같이 스케일을 조정하면 학습이 더 나아지는 경우가 많음 (해서 나쁠 것은 없음!)
- DL에서는 안정적인 모델 학습을 위해 필수적!

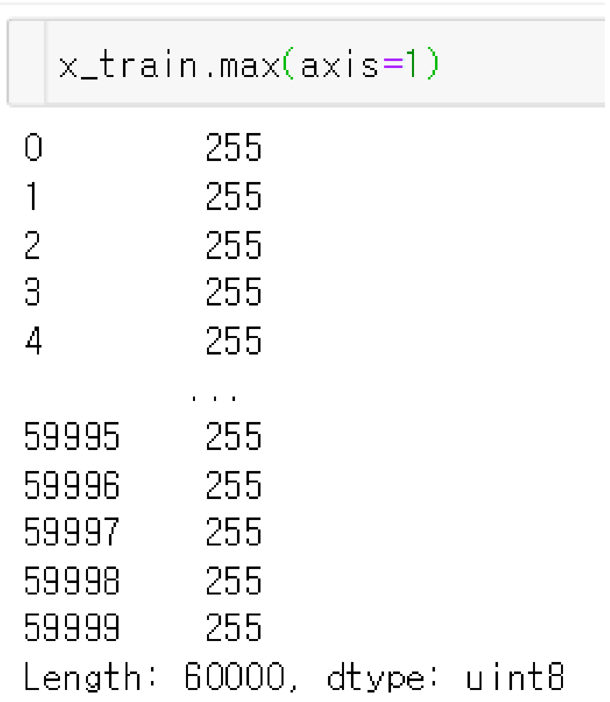
Step 2) Data 스케일(단위) 조정

현재 우리 데이터(MNIST)의 Scale은?



```
x_train.min(axis=1)
```

```
0        0
1        0
2        0
3        0
4        0
        ..
59995    0
59996    0
59997    0
59998    0
59999    0
Length: 60000, dtype: uint8
```



```
x_train.max(axis=1)
```

```
0        255
1        255
2        255
3        255
4        255
        ...
59995    255
59996    255
59997    255
59998    255
59999    255
Length: 60000, dtype: uint8
```

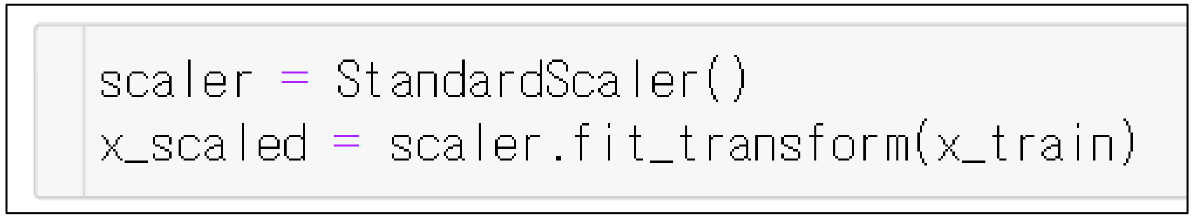
필수적이라고 보기는 어려우나,

해서 나쁠건 없음!

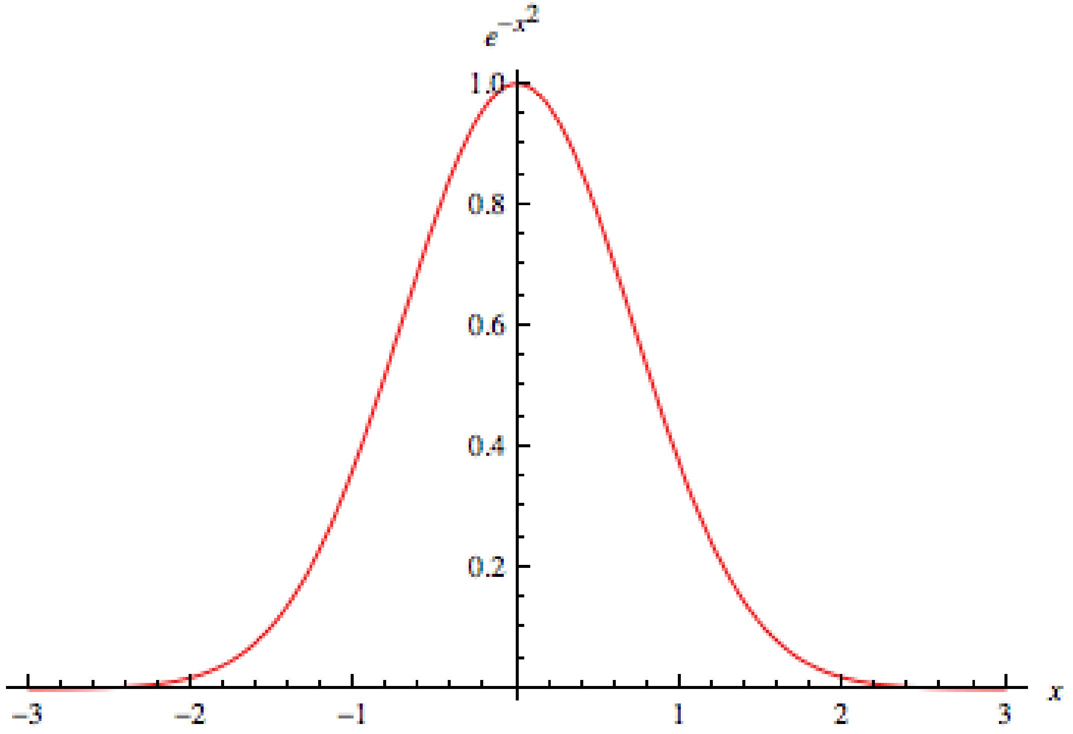
Standard Scaler를 사용해서

변수들의 단위를 통일시켜줄 것!

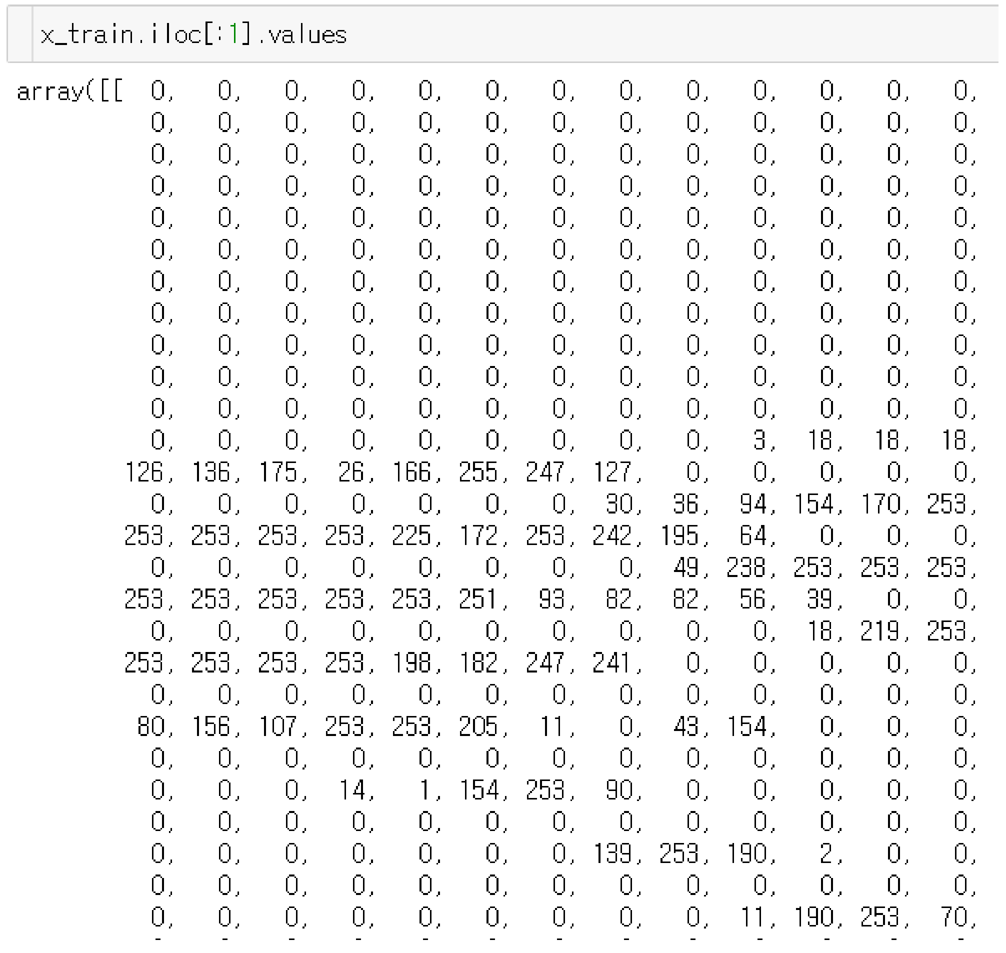
Step 2) Data 스케일(단위) 조정



```
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x_train)
```

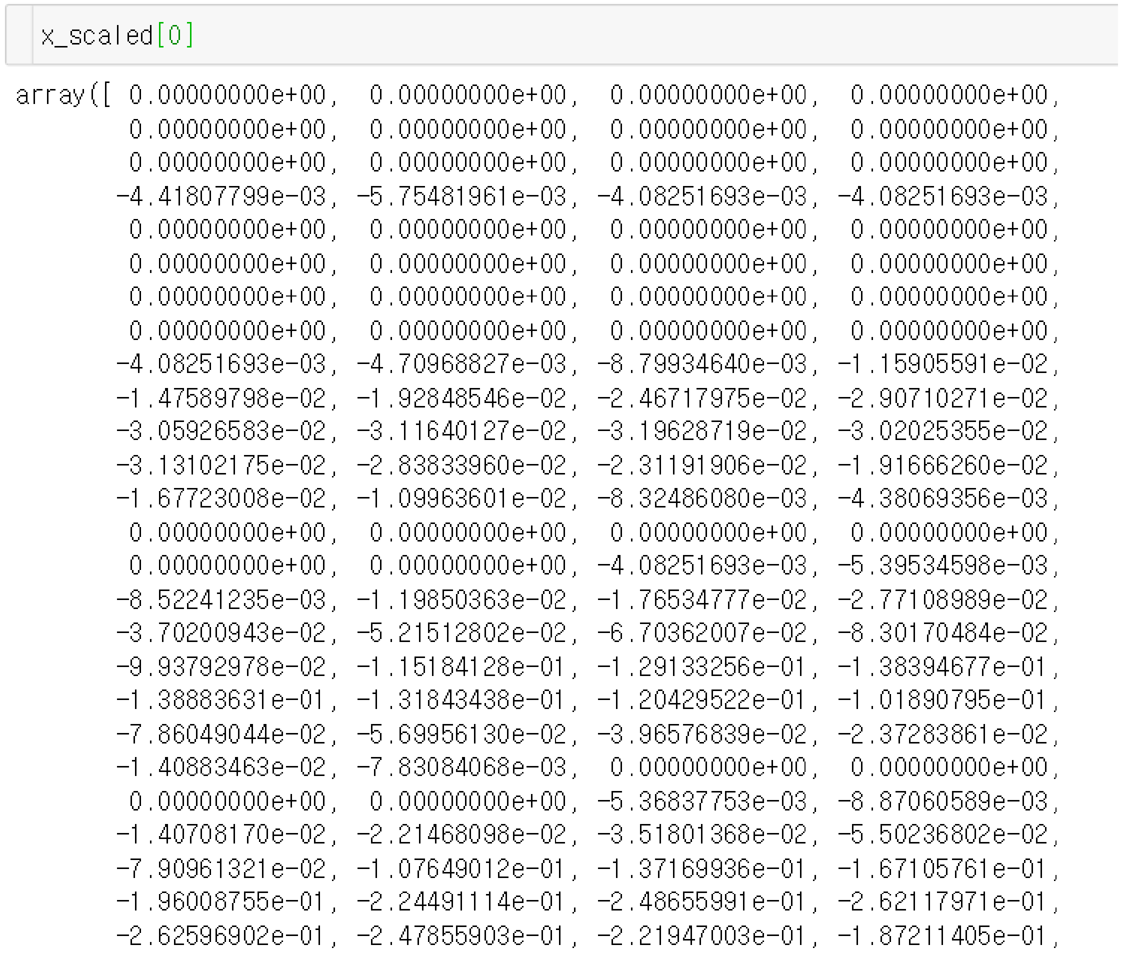


Step 2) Data 스케일(단위) 조정



```
x_train.iloc[:1].values
```

```
array([[  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,   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,   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,   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,   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,   0,   0,   3,  18,  18,  18,
        126, 136, 175,  26, 166, 255, 247, 127,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170, 253,
        253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253,
        253, 253, 253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  18, 219, 253,
        253, 253, 253, 253, 198, 182, 247, 241,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         80, 156, 107, 253, 253, 205,  11,   0,  43, 154,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,  14,   1, 154, 253,  90,   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, 139, 253, 190,   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,  11, 190, 253,  70,
```



```
x_scaled[0]
```

```
array([ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
       -4.41807799e-03, -5.75481961e-03, -4.08251693e-03, -4.08251693e-03,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
       -4.08251693e-03, -4.70968827e-03, -8.79934640e-03, -1.15905591e-02,
       -1.47589798e-02, -1.92848546e-02, -2.46717975e-02, -2.90710271e-02,
       -3.05926583e-02, -3.11640127e-02, -3.19628719e-02, -3.02025355e-02,
       -3.13102175e-02, -2.83833960e-02, -2.31191906e-02, -1.91666260e-02,
       -1.67723008e-02, -1.09963601e-02, -8.32486080e-03, -4.38069356e-03,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00, -4.08251693e-03, -5.39534598e-03,
       -8.52241235e-03, -1.19850363e-02, -1.76534777e-02, -2.77108989e-02,
       -3.70200943e-02, -5.21512802e-02, -6.70362007e-02, -8.30170484e-02,
       -9.93792978e-02, -1.15184128e-01, -1.29133256e-01, -1.38394677e-01,
       -1.38883631e-01, -1.31843438e-01, -1.20429522e-01, -1.01890795e-01,
       -7.86049044e-02, -5.69956130e-02, -3.96576839e-02, -2.37283861e-02,
       -1.40883463e-02, -7.83084068e-03,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00, -5.36837753e-03, -8.87060589e-03,
       -1.40708170e-02, -2.21468098e-02, -3.51801368e-02, -5.50236802e-02,
       -7.90961321e-02, -1.07649012e-01, -1.37169936e-01, -1.67105761e-01,
       -1.96008755e-01, -2.24491114e-01, -2.48655991e-01, -2.62117971e-01,
       -2.62596902e-01, -2.47855903e-01, -2.21947003e-01, -1.87211405e-01,
```

Step 3) 차원 축소

PCA (Principal Component Analysis, 주성분 분석)를 사용해서 축소할 것

총 D개의 변수

사람ID	키	몸무게	시력 평균	…	특징 D
사람 1	170	60	2.0		
사람 2	175	65	1.8		
사람 3	168	58	-2.0		
사람 4	180	70	0.6		
…	…	…	…		
…	…	…	…		
사람 999	177	66	0.3		

총 d개의 변수 (d=2)

사람ID	PC 1	PC 2
사람 1	170	60
사람 2	175	65
사람 3	168	58
사람 4	180	70
…	…	…
…	…	…
사람 999	177	66

주성분 2) 학업능력 지수

Step 3) 차원 축소

주성분 1) 신체 종합 지수

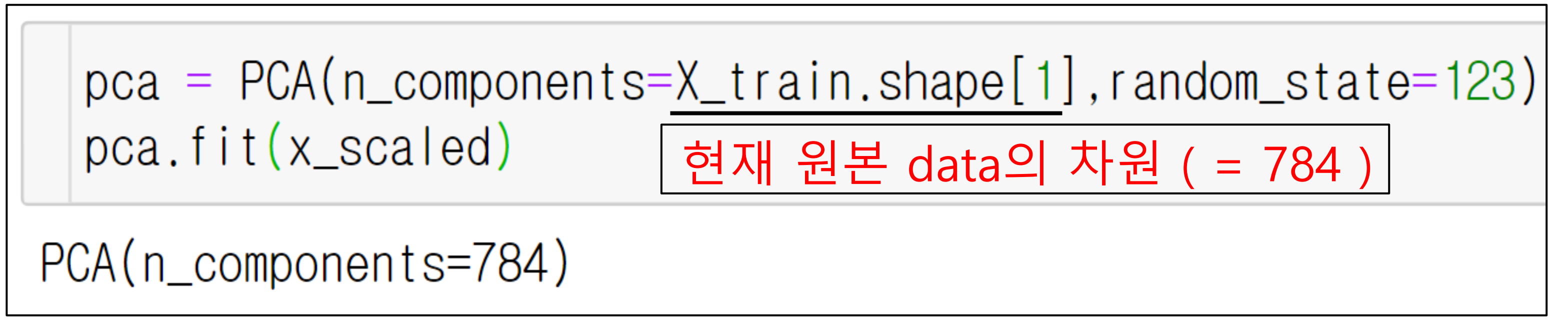
PCA (Principal Component Analysis, 주성분 분석)를 사용해서 축소할 것

총 D개의 변수

사람ID	키	몸무게	시력 평균	…	특징 D
사람 1	170	60	2.0		
사람 2	175	65	1.8		
사람 3	168	58	-2.0		
사람 4	180	70	0.6		
…	…	…	…		
…	…	…	…		
사람 999	177	66	0.3		

사람ID	PC 1	PC 2
사람 1	170	60
사람 2	175	65
사람 3	168	58
사람 4	180	70
…	…	…
…	…	…
사람 999	177	66

Step 3) 차원 축소

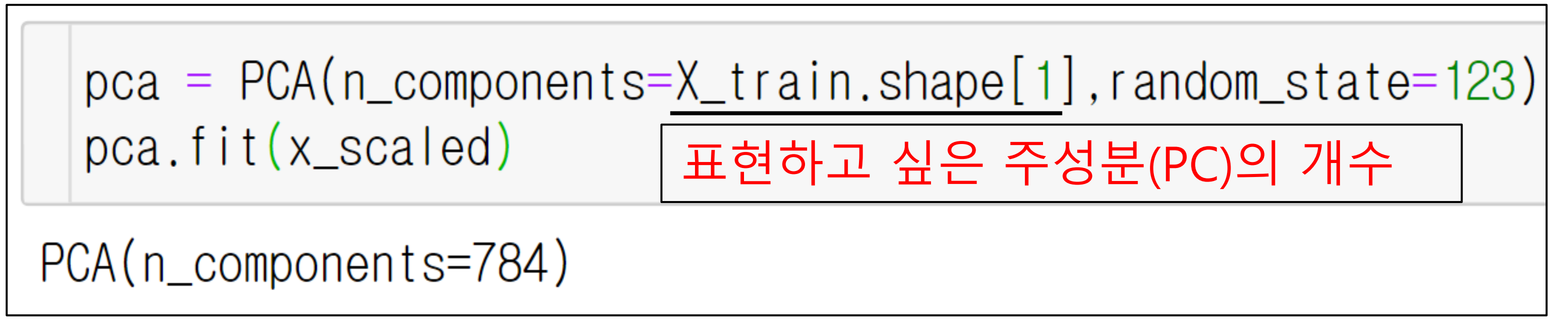


```
pca = PCA(n_components=X_train.shape[1],random_state=123)
pca.fit(x_scaled)
```

현재 원본 data의 차원 (= 784)

```
PCA(n_components=784)
```

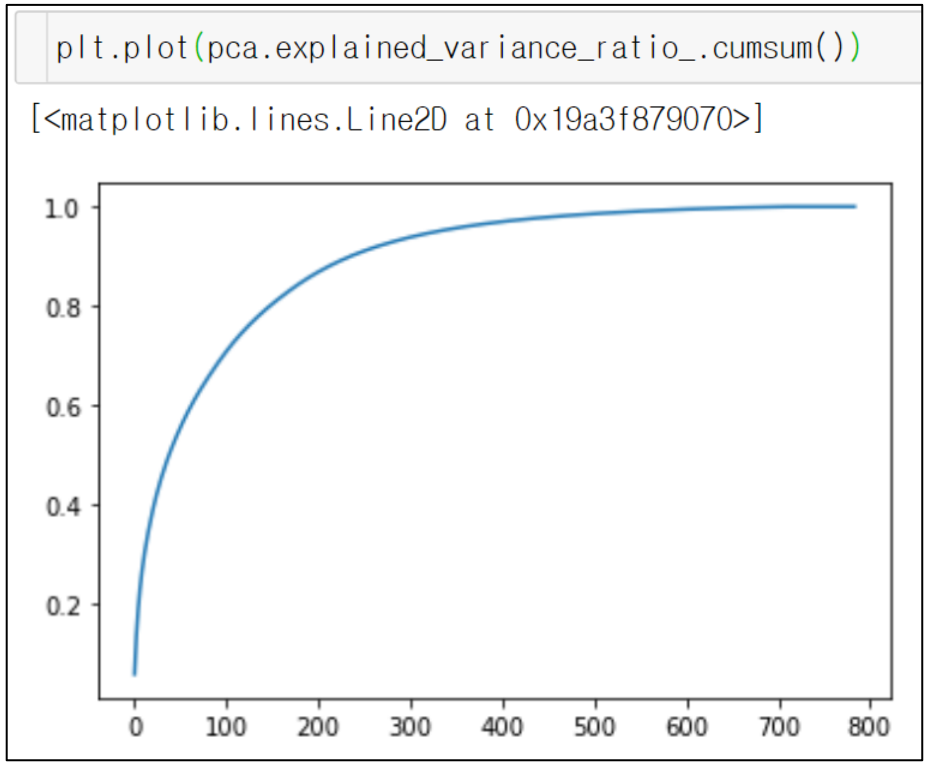
Step 3) 차원 축소



```
pca = PCA(n_components=X_train.shape[1],random_state=123)
pca.fit(x_scaled)
```

표현하고 싶은 주성분(PC)의 개수

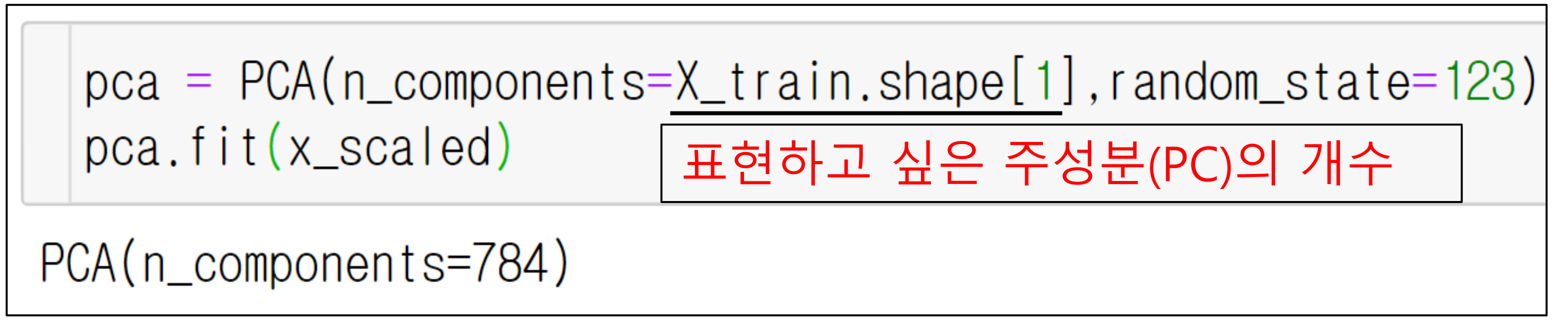
```
PCA(n_components=784)
```



```
plt.plot(pca.explained_variance_ratio_.cumsum())
```

```
[<matplotlib.lines.Line2D at 0x19a3f879070>]
```

Step 3) 차원 축소



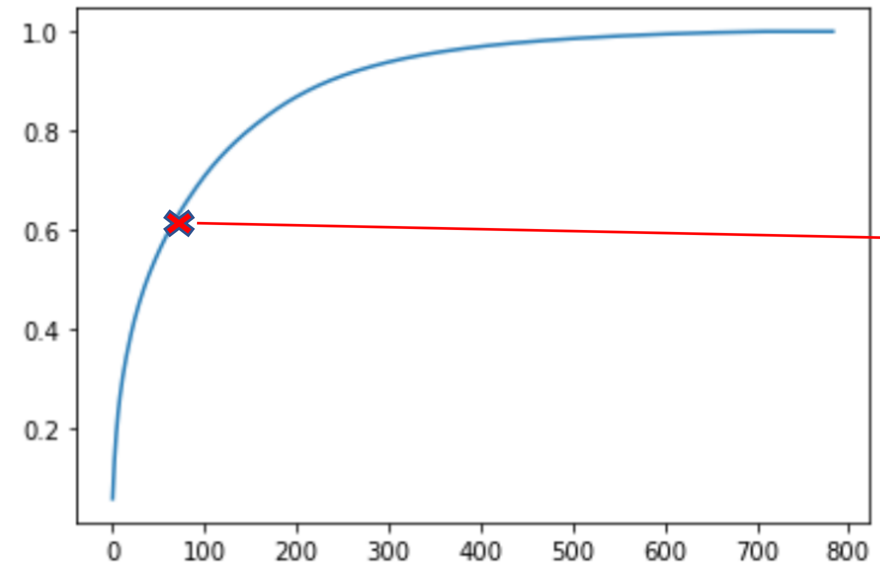
```
pca = PCA(n_components=X_train.shape[1],random_state=123)
pca.fit(x_scaled)
```

표현하고 싶은 주성분(PC)의 개수

```
PCA(n_components=784)
```

```
plt.plot(pca.explained_variance_ratio_.cumsum())
```

```
[<matplotlib.lines.Line2D at 0x19a3f879070>]
```

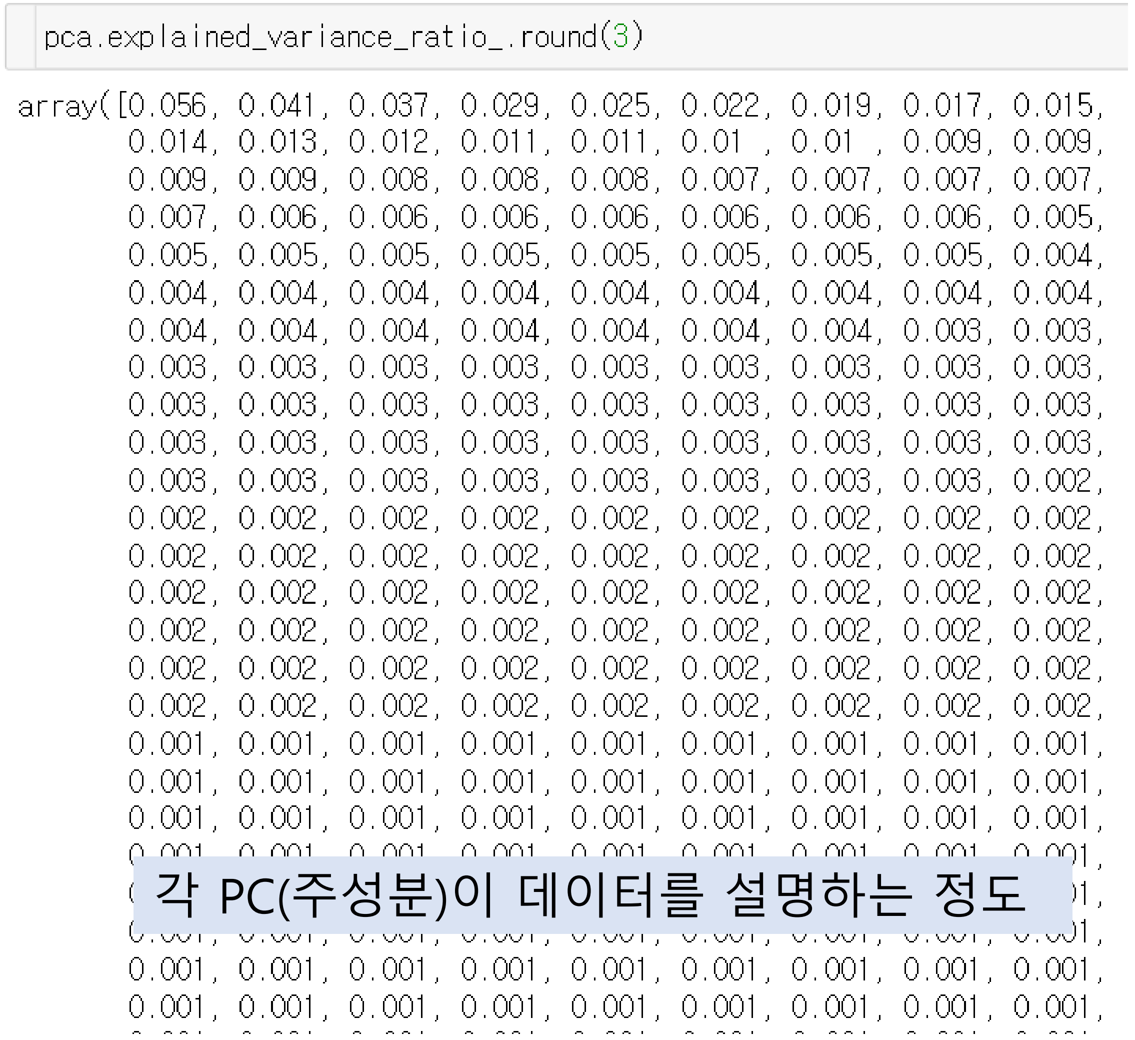


쉽게 말해, "축소된 변수(PC1,PC2...PCd)"가 기존의 데이터를 얼마나 잘 설명하는지!

- 기존 데이터 : 784차원
- 축소 후 데이터 : 100차원

이 100차원의 데이터 만으로도, 기존 데이터가 가지는 정보의 60%이상을 설명한다!

Step 3) 차원 축소

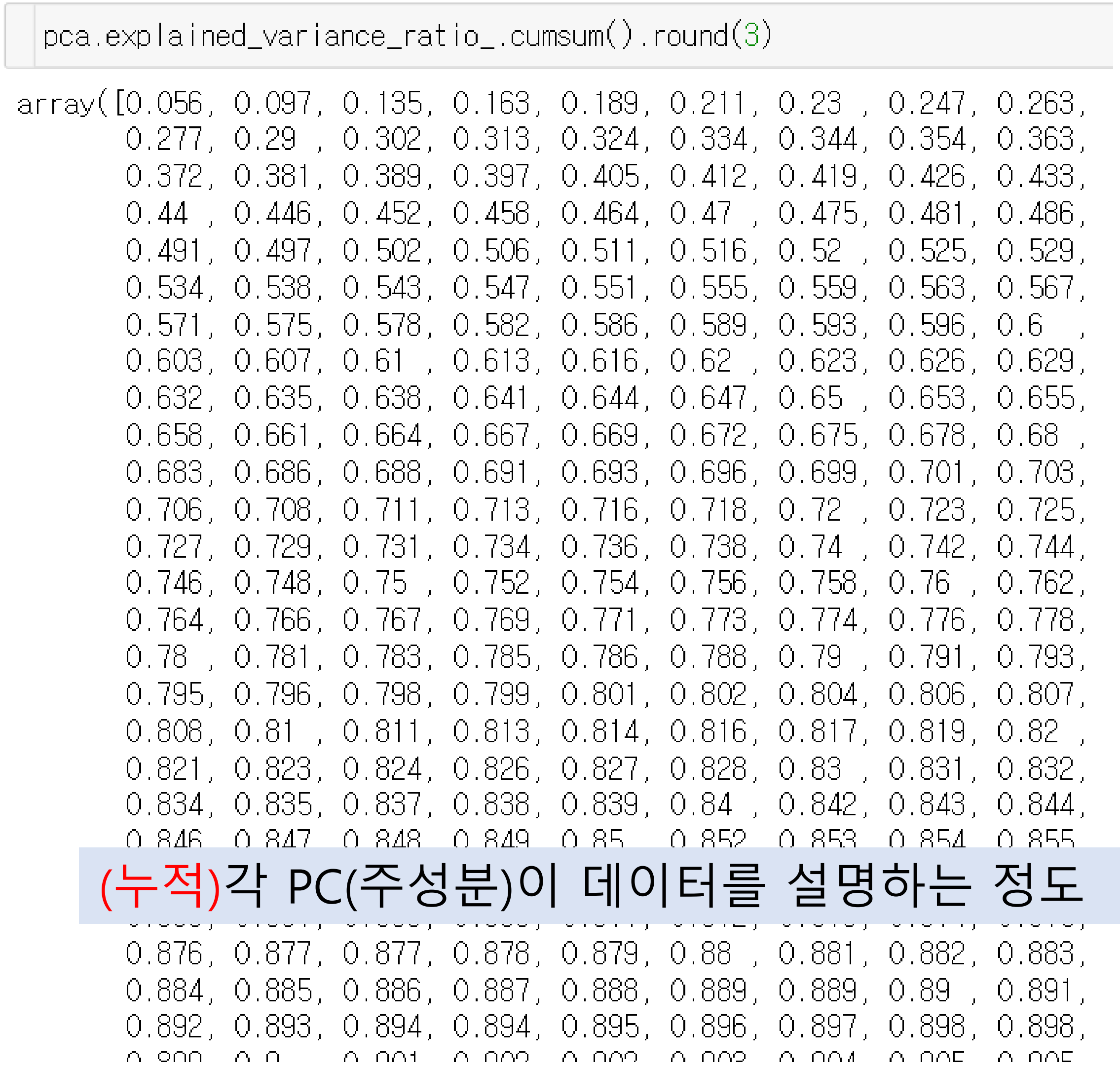


```
pca.explained_variance_ratio_.round(3)
```

```
array([0.056, 0.041, 0.037, 0.029, 0.025, 0.022, 0.019, 0.017, 0.015,
       0.014, 0.013, 0.012, 0.011, 0.011, 0.01 , 0.01 , 0.009, 0.009,
       0.009, 0.009, 0.008, 0.008, 0.008, 0.007, 0.007, 0.007, 0.007,
       0.007, 0.006, 0.006, 0.006, 0.006, 0.006, 0.006, 0.006, 0.005,
       0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.004,
       0.004, 0.004, 0.004, 0.004, 0.004, 0.004, 0.004, 0.004, 0.004,
       0.004, 0.004, 0.004, 0.004, 0.004, 0.004, 0.004, 0.003, 0.003,
       0.003, 0.003, 0.003, 0.003, 0.003, 0.003, 0.003, 0.003, 0.003,
       0.003, 0.003, 0.003, 0.003, 0.003, 0.003, 0.003, 0.003, 0.003,
       0.003, 0.003, 0.003, 0.003, 0.003, 0.003, 0.003, 0.003, 0.003,
       0.003, 0.003, 0.003, 0.003, 0.003, 0.003, 0.003, 0.003, 0.002,
       0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002,
       0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002,
       0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002,
       0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002,
       0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002,
       0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002,
       0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001,
       0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001,
       0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001,
       0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001,
```

각 PC(주성분)이 데이터를 설명하는 정도

```
       0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001,
       0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001,
```



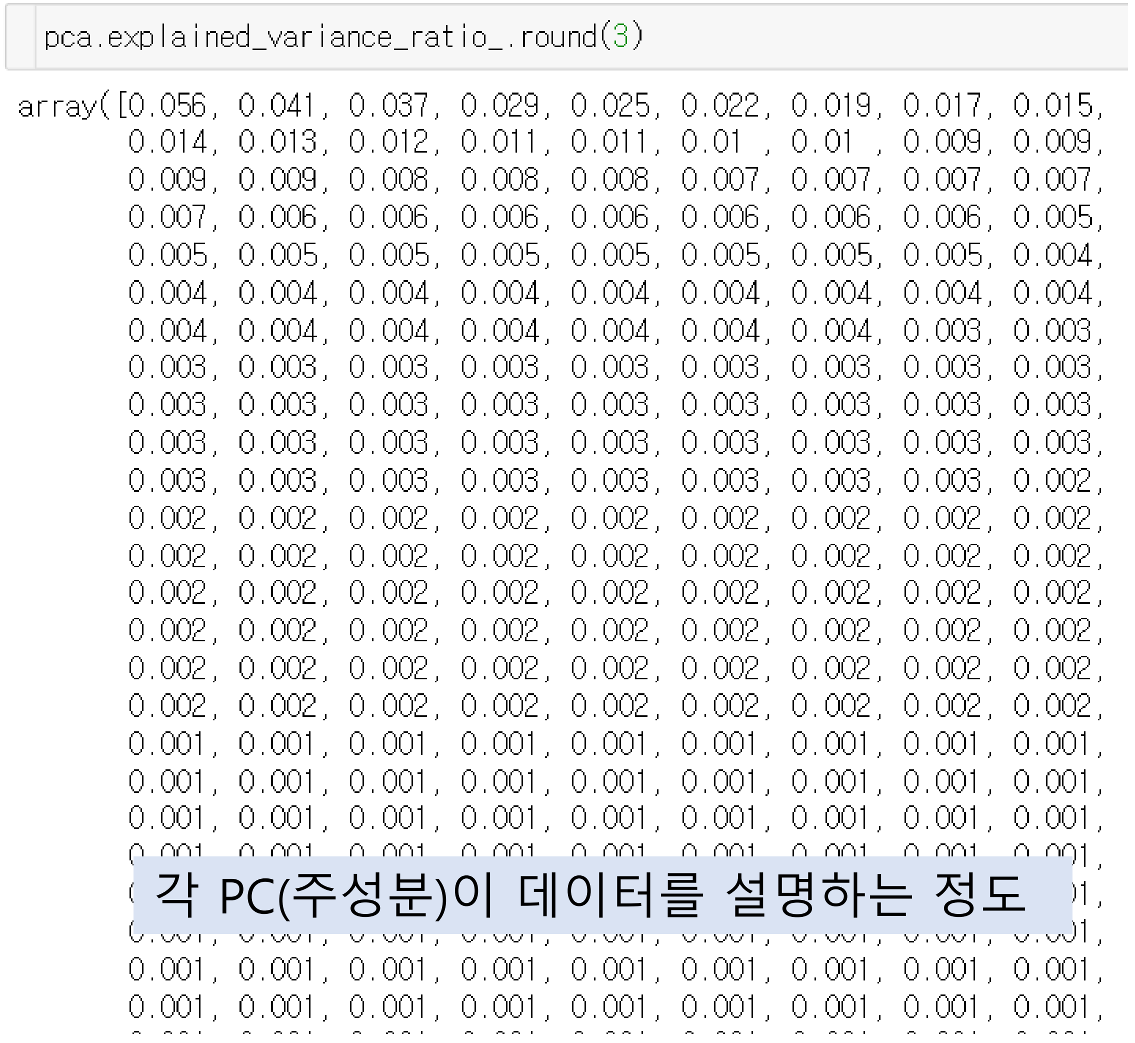
```
pca.explained_variance_ratio_.cumsum().round(3)
```

```
array([0.056, 0.097, 0.135, 0.163, 0.189, 0.211, 0.23 , 0.247, 0.263,
       0.277, 0.29 , 0.302, 0.313, 0.324, 0.334, 0.344, 0.354, 0.363,
       0.372, 0.381, 0.389, 0.397, 0.405, 0.412, 0.419, 0.426, 0.433,
       0.44 , 0.446, 0.452, 0.458, 0.464, 0.47 , 0.475, 0.481, 0.486,
       0.491, 0.497, 0.502, 0.506, 0.511, 0.516, 0.52 , 0.525, 0.529,
       0.534, 0.538, 0.543, 0.547, 0.551, 0.555, 0.559, 0.563, 0.567,
       0.571, 0.575, 0.578, 0.582, 0.586, 0.589, 0.593, 0.596, 0.6  ,
       0.603, 0.607, 0.61 , 0.613, 0.616, 0.62 , 0.623, 0.626, 0.629,
       0.632, 0.635, 0.638, 0.641, 0.644, 0.647, 0.65 , 0.653, 0.655,
       0.658, 0.661, 0.664, 0.667, 0.669, 0.672, 0.675, 0.678, 0.68 ,
       0.683, 0.686, 0.688, 0.691, 0.693, 0.696, 0.699, 0.701, 0.703,
       0.706, 0.708, 0.711, 0.713, 0.716, 0.718, 0.72 , 0.723, 0.725,
       0.727, 0.729, 0.731, 0.734, 0.736, 0.738, 0.74 , 0.742, 0.744,
       0.746, 0.748, 0.75 , 0.752, 0.754, 0.756, 0.758, 0.76 , 0.762,
       0.764, 0.766, 0.767, 0.769, 0.771, 0.773, 0.774, 0.776, 0.778,
       0.78 , 0.781, 0.783, 0.785, 0.786, 0.788, 0.79 , 0.791, 0.793,
       0.795, 0.796, 0.798, 0.799, 0.801, 0.802, 0.804, 0.806, 0.807,
       0.808, 0.81 , 0.811, 0.813, 0.814, 0.816, 0.817, 0.819, 0.82 ,
       0.821, 0.823, 0.824, 0.826, 0.827, 0.828, 0.83 , 0.831, 0.832,
       0.834, 0.835, 0.837, 0.838, 0.839, 0.84 , 0.842, 0.843, 0.844,
       0.846, 0.847, 0.848, 0.849, 0.85 , 0.852, 0.853, 0.854, 0.855,
```

(누적)각 PC(주성분)이 데이터를 설명하는 정도

```
       0.876, 0.877, 0.877, 0.878, 0.879, 0.88 , 0.881, 0.882, 0.883,
       0.884, 0.885, 0.886, 0.887, 0.888, 0.889, 0.889, 0.89 , 0.891,
       0.892, 0.893, 0.894, 0.894, 0.895, 0.896, 0.897, 0.898, 0.898,
```

Step 3) 차원 축소

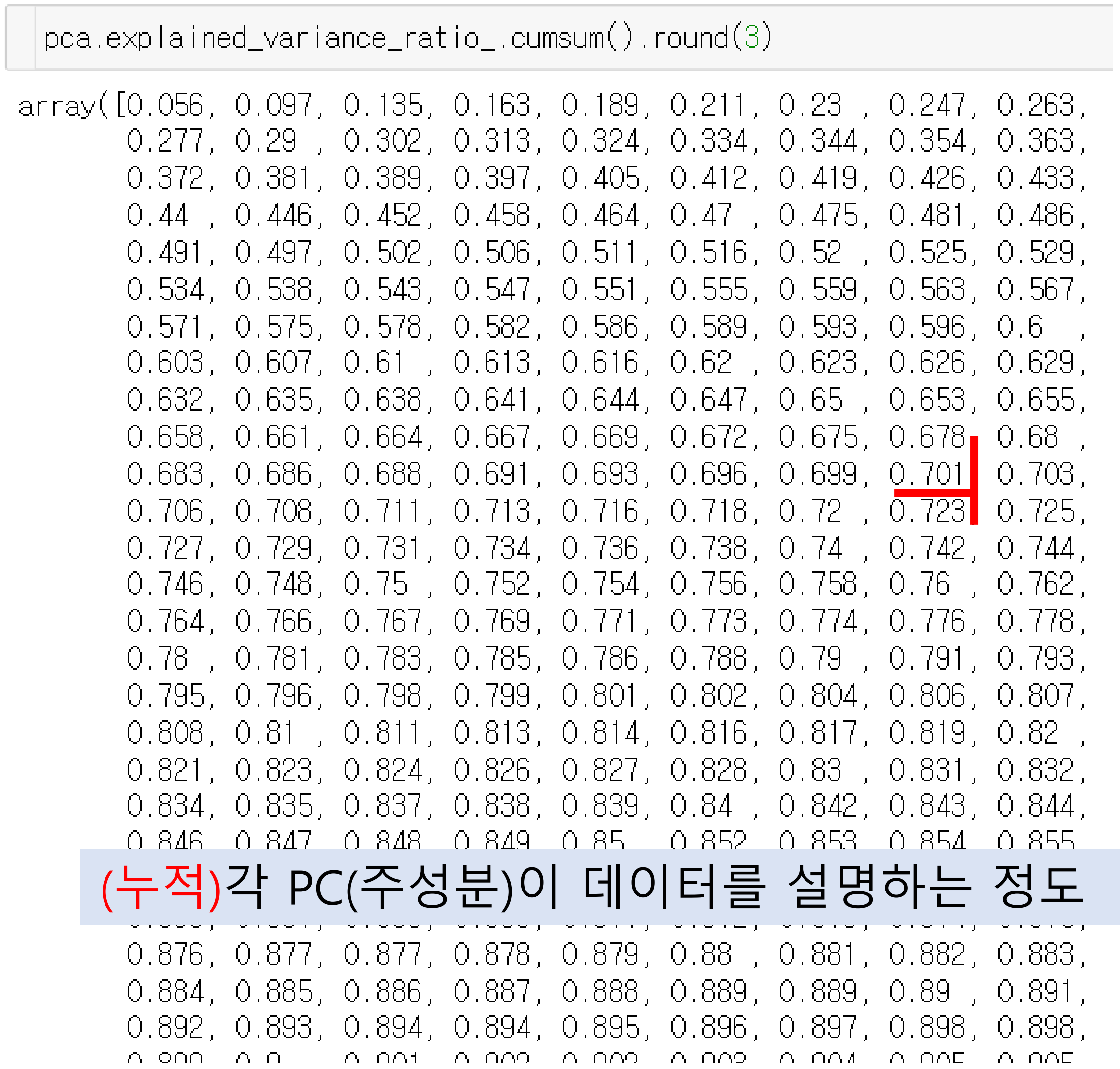


```
pca.explained_variance_ratio_.round(3)
```

```
array([0.056, 0.041, 0.037, 0.029, 0.025, 0.022, 0.019, 0.017, 0.015,
       0.014, 0.013, 0.012, 0.011, 0.011, 0.01 , 0.01 , 0.009, 0.009,
       0.009, 0.009, 0.008, 0.008, 0.008, 0.007, 0.007, 0.007, 0.007,
       0.007, 0.006, 0.006, 0.006, 0.006, 0.006, 0.006, 0.006, 0.005,
       0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.004,
       0.004, 0.004, 0.004, 0.004, 0.004, 0.004, 0.004, 0.004, 0.004,
       0.004, 0.004, 0.004, 0.004, 0.004, 0.004, 0.004, 0.003, 0.003,
       0.003, 0.003, 0.003, 0.003, 0.003, 0.003, 0.003, 0.003, 0.003,
       0.003, 0.003, 0.003, 0.003, 0.003, 0.003, 0.003, 0.003, 0.003,
       0.003, 0.003, 0.003, 0.003, 0.003, 0.003, 0.003, 0.003, 0.003,
       0.003, 0.003, 0.003, 0.003, 0.003, 0.003, 0.003, 0.003, 0.002,
       0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002,
       0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002,
       0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002,
       0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002,
       0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002,
       0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002,
       0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001,
       0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001,
       0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001,
       0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001,
```

각 PC(주성분)이 데이터를 설명하는 정도

```
       0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001,
       0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001,
```



```
pca.explained_variance_ratio_.cumsum().round(3)
```

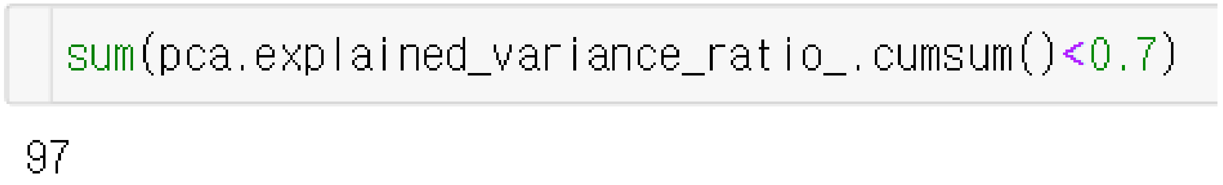
```
array([0.056, 0.097, 0.135, 0.163, 0.189, 0.211, 0.23 , 0.247, 0.263,
       0.277, 0.29 , 0.302, 0.313, 0.324, 0.334, 0.344, 0.354, 0.363,
       0.372, 0.381, 0.389, 0.397, 0.405, 0.412, 0.419, 0.426, 0.433,
       0.44 , 0.446, 0.452, 0.458, 0.464, 0.47 , 0.475, 0.481, 0.486,
       0.491, 0.497, 0.502, 0.506, 0.511, 0.516, 0.52 , 0.525, 0.529,
       0.534, 0.538, 0.543, 0.547, 0.551, 0.555, 0.559, 0.563, 0.567,
       0.571, 0.575, 0.578, 0.582, 0.586, 0.589, 0.593, 0.596, 0.6  ,
       0.603, 0.607, 0.61 , 0.613, 0.616, 0.62 , 0.623, 0.626, 0.629,
       0.632, 0.635, 0.638, 0.641, 0.644, 0.647, 0.65 , 0.653, 0.655,
       0.658, 0.661, 0.664, 0.667, 0.669, 0.672, 0.675, 0.678, 0.68 ,
       0.683, 0.686, 0.688, 0.691, 0.693, 0.696, 0.699, 0.701, 0.703,
       0.706, 0.708, 0.711, 0.713, 0.716, 0.718, 0.72 , 0.723, 0.725,
       0.727, 0.729, 0.731, 0.734, 0.736, 0.738, 0.74 , 0.742, 0.744,
       0.746, 0.748, 0.75 , 0.752, 0.754, 0.756, 0.758, 0.76 , 0.762,
       0.764, 0.766, 0.767, 0.769, 0.771, 0.773, 0.774, 0.776, 0.778,
       0.78 , 0.781, 0.783, 0.785, 0.786, 0.788, 0.79 , 0.791, 0.793,
       0.795, 0.796, 0.798, 0.799, 0.801, 0.802, 0.804, 0.806, 0.807,
       0.808, 0.81 , 0.811, 0.813, 0.814, 0.816, 0.817, 0.819, 0.82 ,
       0.821, 0.823, 0.824, 0.826, 0.827, 0.828, 0.83 , 0.831, 0.832,
       0.834, 0.835, 0.837, 0.838, 0.839, 0.84 , 0.842, 0.843, 0.844,
       0.846, 0.847, 0.848, 0.849, 0.85 , 0.852, 0.853, 0.854, 0.855,
```

(누적)각 PC(주성분)이 데이터를 설명하는 정도

```
       0.876, 0.877, 0.877, 0.878, 0.879, 0.88 , 0.881, 0.882, 0.883,
       0.884, 0.885, 0.886, 0.887, 0.888, 0.889, 0.889, 0.89 , 0.891,
       0.892, 0.893, 0.894, 0.894, 0.895, 0.896, 0.897, 0.898, 0.898,
```

Step 3) 차원 축소

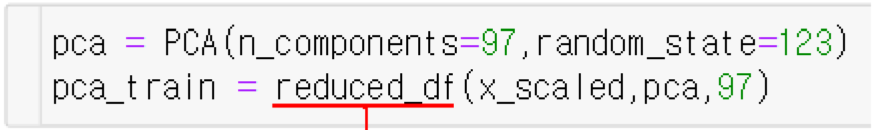
확인 해보니, 97개의 주성분만으로도, 전체 데이터의 약 70%의 정보를 설명할 수 있다!



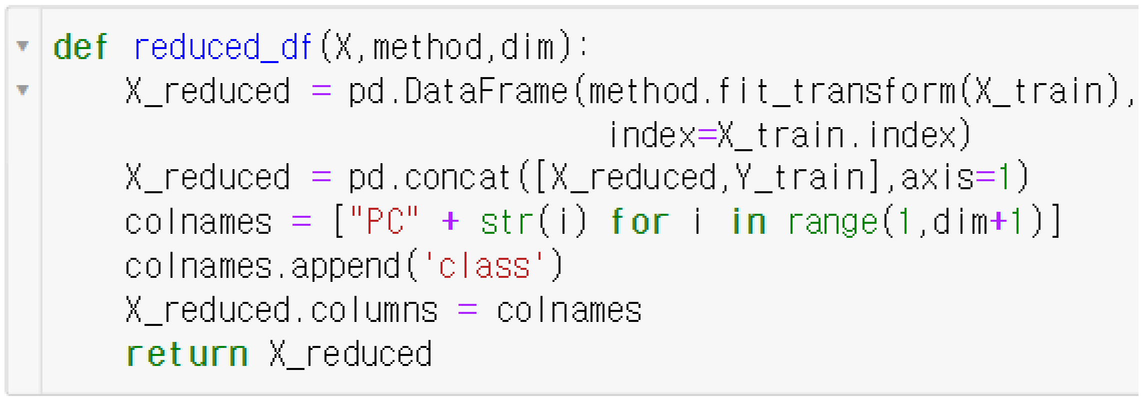
```
sum(pca.explained_variance_ratio_.cumsum()<0.7)
```

```
97
```

차원이 97개로 축소된 dataset 완성시키기!

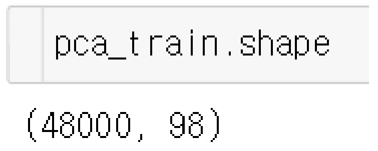


```
pca = PCA(n_components=97,random_state=123)
pca_train = reduced_df(x_scaled,pca,97)
```



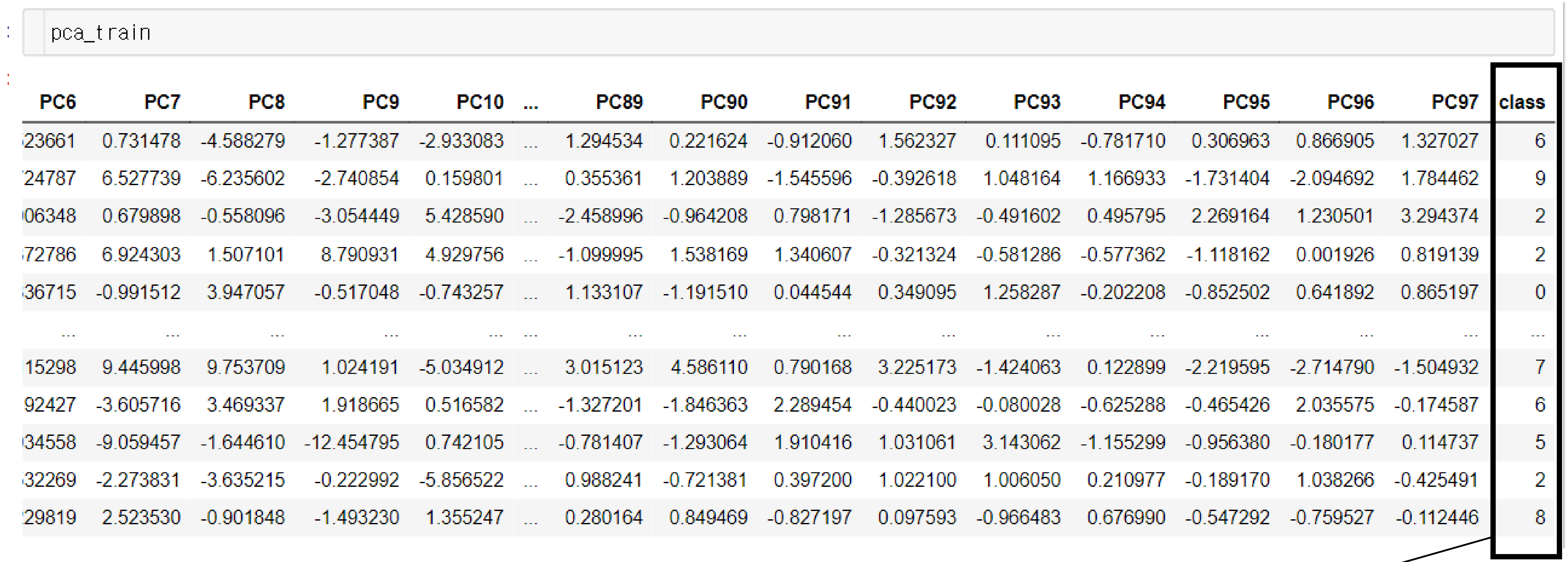
```
def reduced_df(X,method,dim):
    X_reduced = pd.DataFrame(method.fit_transform(X_train),
                             index=X_train.index)
    X_reduced = pd.concat([X_reduced,Y_train],axis=1)
    colnames = ["PC" + str(i) for i in range(1,dim+1)]
    colnames.append('class')
    X_reduced.columns = colnames
    return X_reduced
```

Step 3) 차원 축소



```
pca_train.shape
```

```
(48000, 98)
```



```
pca_train
```

PC6	PC7	PC8	PC9	PC10	...	PC89	PC90	PC91	PC92	PC93	PC94	PC95	PC96	PC97	class
23661	0.731478	-4.588279	-1.277387	-2.933083	...	1.294534	0.221624	-0.912060	1.562327	0.111095	-0.781710	0.306963	0.866905	1.327027	6
24787	6.527739	-6.235602	-2.740854	0.159801	...	0.355361	1.203889	-1.545596	-0.392618	1.048164	1.166933	-1.731404	-2.094692	1.784462	9
06348	0.679898	-0.558096	-3.054449	5.428590	...	-2.458996	-0.964208	0.798171	-1.285673	-0.491602	0.495795	2.269164	1.230501	3.294374	2
72786	6.924303	1.507101	8.790931	4.929756	...	-1.099995	1.538169	1.340607	-0.321324	-0.581286	-0.577362	-1.118162	0.001926	0.819139	2
36715	-0.991512	3.947057	-0.517048	-0.743257	...	1.133107	-1.191510	0.044544	0.349095	1.258287	-0.202208	-0.852502	0.641892	0.865197	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
15298	9.445998	9.753709	1.024191	-5.034912	...	3.015123	4.586110	0.790168	3.225173	-1.424063	0.122899	-2.219595	-2.714790	-1.504932	7
92427	-3.605716	3.469337	1.918665	0.516582	...	-1.327201	-1.846363	2.289454	-0.440023	-0.080028	-0.625288	-0.465426	2.035575	-0.174587	6
34558	-9.059457	-1.644610	-12.454795	0.742105	...	-0.781407	-1.293064	1.910416	1.031061	3.143062	-1.155299	-0.956380	-0.180177	0.114737	5
32269	-2.273831	-3.635215	-0.222992	-5.856522	...	0.988241	-0.721381	0.397200	1.022100	1.006050	0.210977	-0.189170	1.038266	-0.425491	2
29819	2.523530	-0.901848	-1.493230	1.355247	...	0.280164	0.849469	-0.827197	0.097593	-0.966483	0.676990	-0.547292	-0.759527	-0.112446	8

우리에게 있어서 반드시 필요한 것은 아님

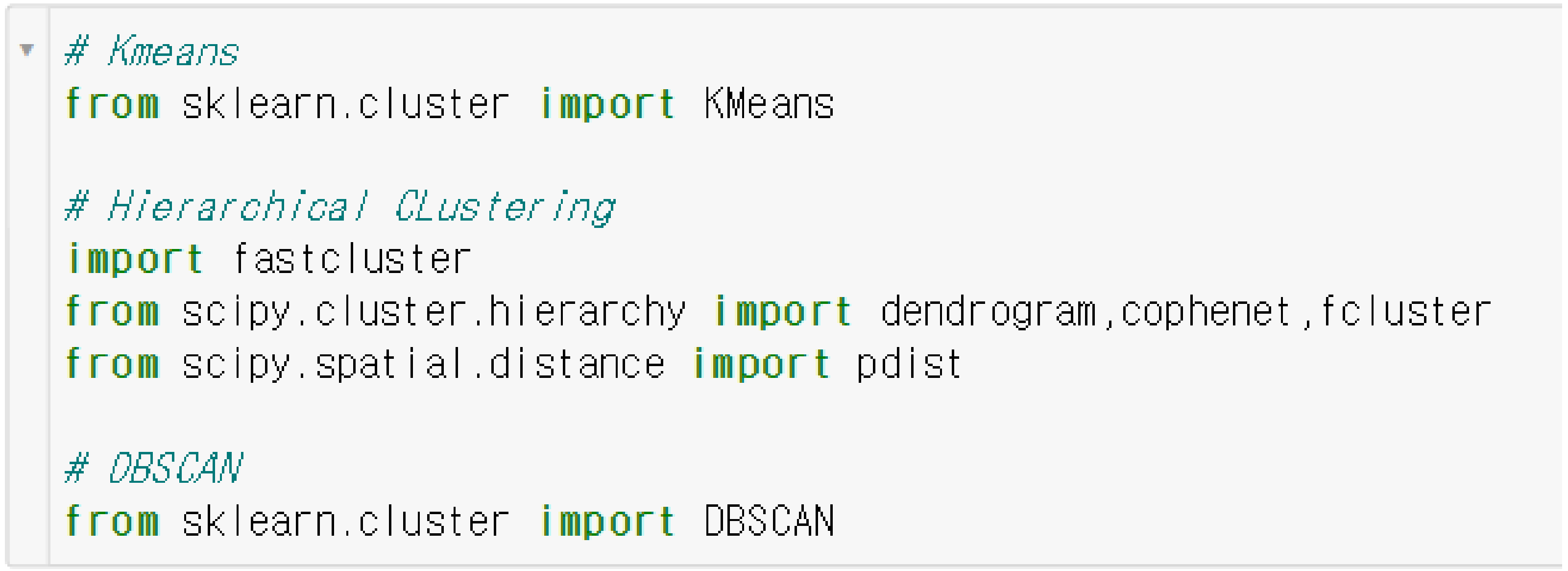
(복습 : Clustering은 " 비지도" 학습 (즉 Y값,정답이 필요하지 않음))

1. K-Means

Step 4) Clustering

총 3가지 방법을 사용해 볼 것

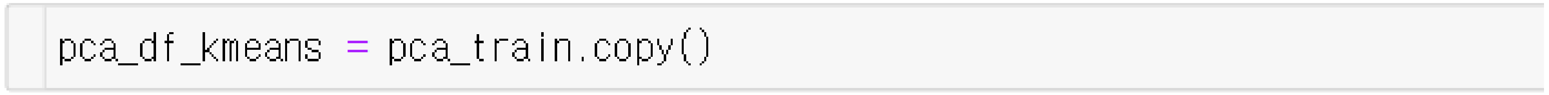
- **K Means**
- Hierarchical Clustering
- DBSCAN



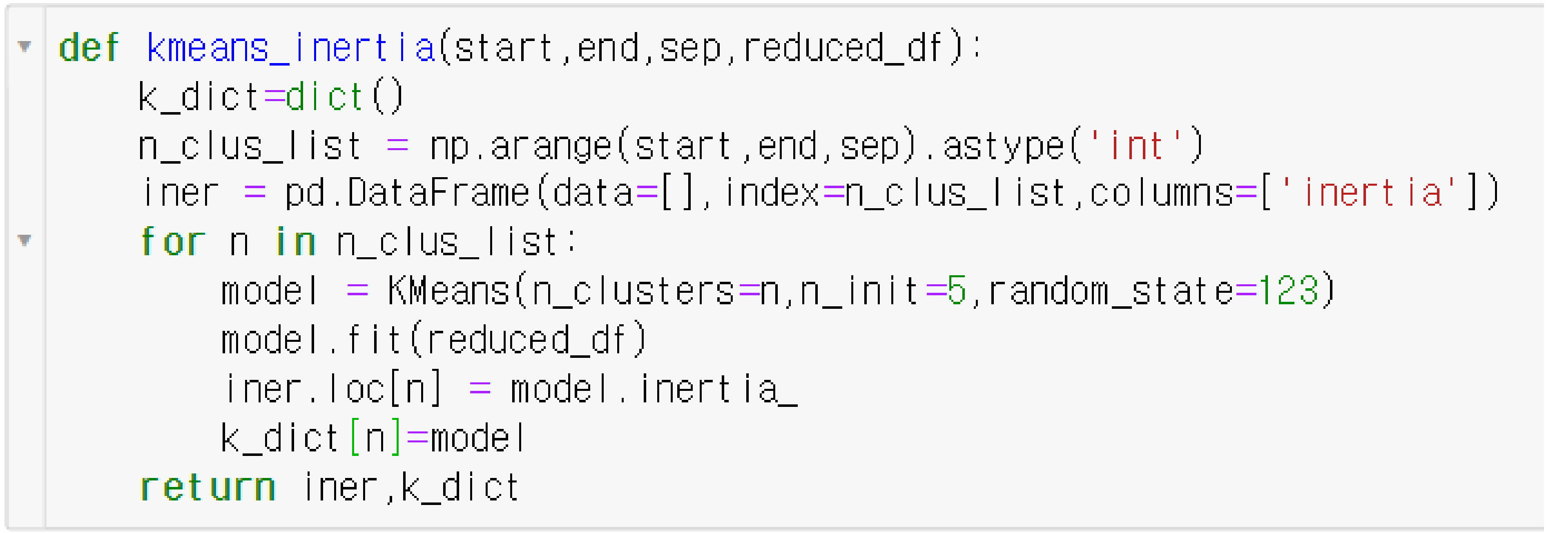
```
# Kmeans
from sklearn.cluster import KMeans

# Hierarchical Clustering
import fastcluster
from scipy.cluster.hierarchy import dendrogram,cophenet,fcluster
from scipy.spatial.distance import pdist

# DBSCAN
from sklearn.cluster import DBSCAN
```

1. K-Means

Step 4) Clustering



```
pca_df_kmeans = pca_train.copy()
```



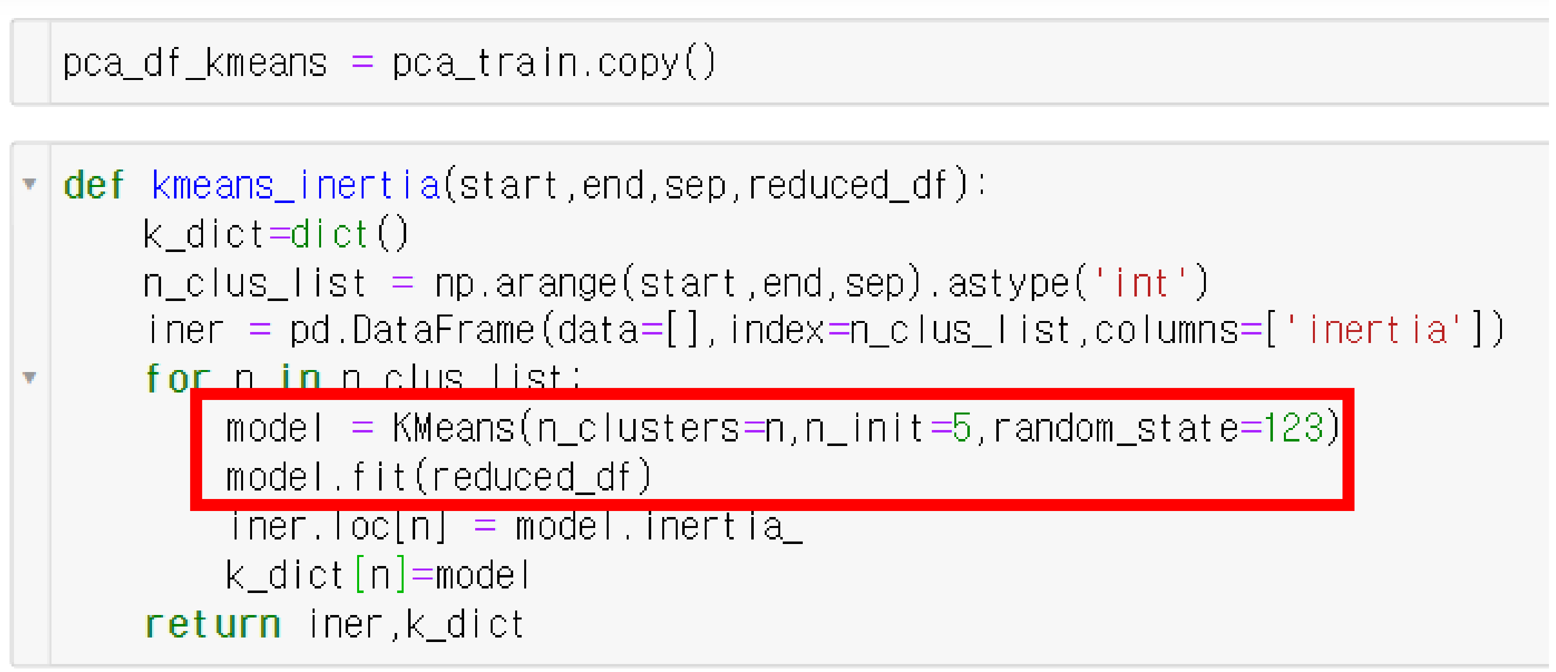
```
def kmeans_inertia(start,end,sep,reduced_df):
    k_dict=dict()
    n_clus_list = np.arange(start,end,sep).astype('int')
    iner = pd.DataFrame(data=[],index=n_clus_list,columns=['inertia'])
    for n in n_clus_list:
        model = KMeans(n_clusters=n,n_init=5,random_state=123)
        model.fit(reduced_df)
        iner.loc[n] = model.inertia_
        k_dict[n]=model
    return iner,k_dict
```

최적의 "K"를 찾기 위한 함수

- ex. start=4부터 end=20까지, 2를 간격으로 K를 설정하여 fitting 시켜보기
- 매 K마다의 inerti를 함께 저장

Step 4) Clustering

1. K-Means



```
pca_df_kmeans = pca_train.copy()
```

```
def kmeans_inertia(start,end,sep,reduced_df):
    k_dict=dict()
    n_clus_list = np.arange(start,end,sep).astype('int')
    iner = pd.DataFrame(data=[],index=n_clus_list,columns=['inertia'])
    for n in n_clus_list:
        model = KMeans(n_clusters=n,n_init=5,random_state=123)
        model.fit(reduced_df)
        iner.loc[n] = model.inertia_
        k_dict[n]=model
    return iner,k_dict
```

최적의 "K"를 찾기 위한 함수

- ex. start=4부터 end=20까지, 2를 간격으로 K를 설정하여 fitting 시켜보기
- 매 K마다의 inerti를 함께 저장

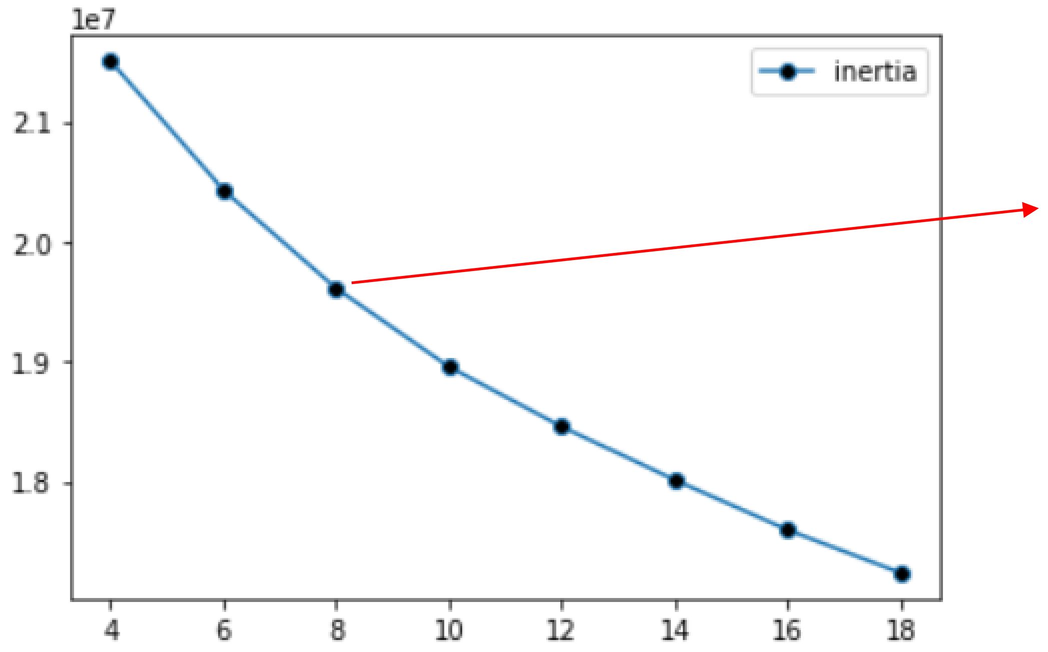
Step 4) Clustering

1. K-Means

```
kmeans_iner, kmeans_dict = kmeans_inertia(4,20,2,pca_df_kmeans)
```

```
kmeans_iner.plot(style='.-',marker='o', markerfacecolor='black')
```

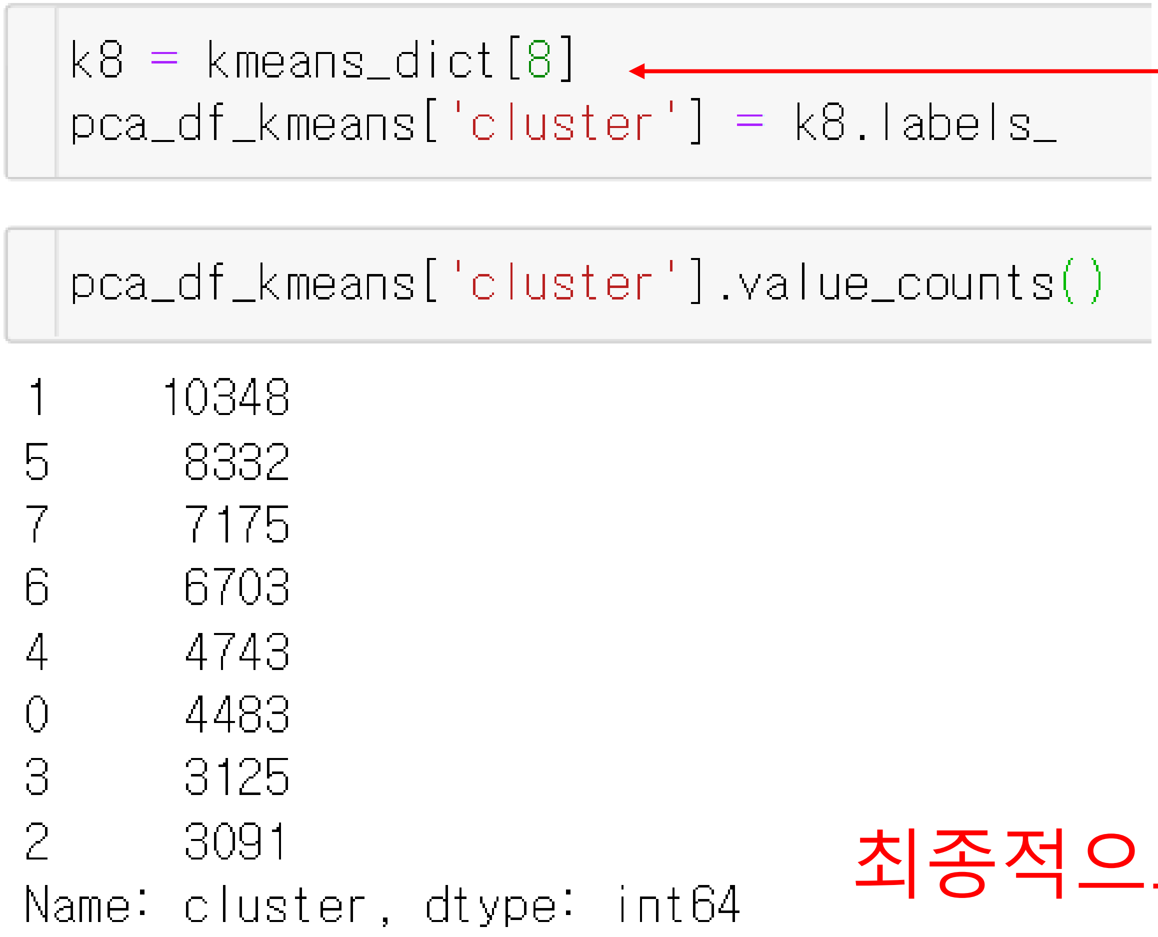
```
<matplotlib.axes._subplots.AxesSubplot at 0x19a3f6dab80>
```



(확실히 딱 꺾이는 지점이 있다고 보긴 어렵지만...)
그나마 적당해 보이는 K=8로 지정하기!

Step 4) Clustering

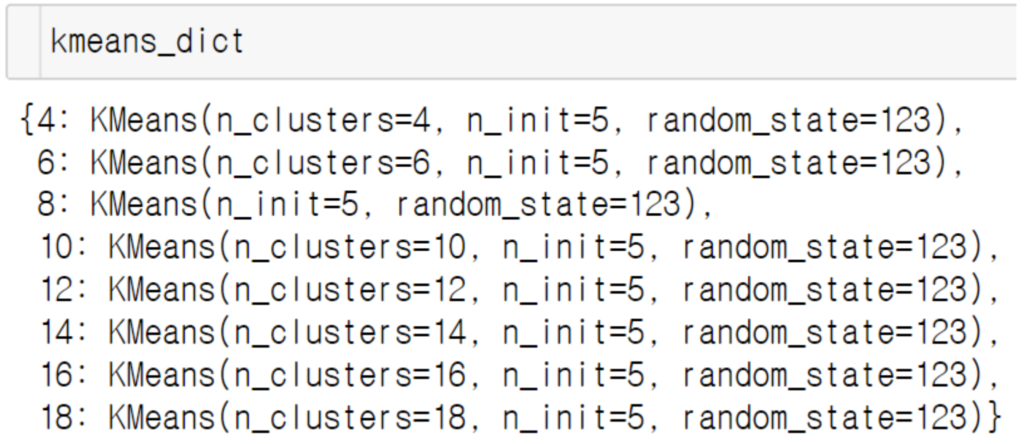
1. K-Means



```
k8 = kmeans_dict[8]
pca_df_kmeans['cluster'] = k8.labels_
```

```
pca_df_kmeans['cluster'].value_counts()
```

```
1    10348
5     8332
7     7175
6     6703
4     4743
0     4483
3     3125
2     3091
Name: cluster, dtype: int64
```



```
kmeans_dict
```

```
{4: KMeans(n_clusters=4, n_init=5, random_state=123),
 6: KMeans(n_clusters=6, n_init=5, random_state=123),
 8: KMeans(n_init=5, random_state=123),
 10: KMeans(n_clusters=10, n_init=5, random_state=123),
 12: KMeans(n_clusters=12, n_init=5, random_state=123),
 14: KMeans(n_clusters=14, n_init=5, random_state=123),
 16: KMeans(n_clusters=16, n_init=5, random_state=123),
 18: KMeans(n_clusters=18, n_init=5, random_state=123)}
```

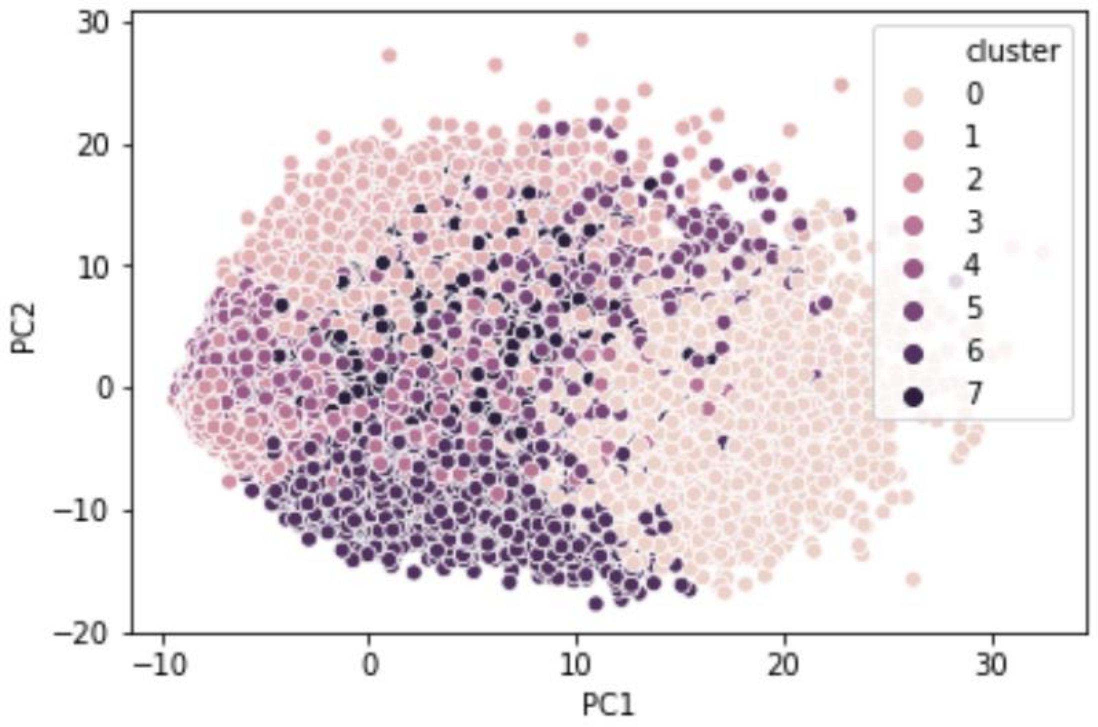
최종적으로 Clustering된 결과!

Step 4) Clustering

1. K-Means

```
sns.scatterplot(x='PC1',y='PC2',hue='cluster',legend='full',data=pca_df_kmeans)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x19a3f7c0940>
```



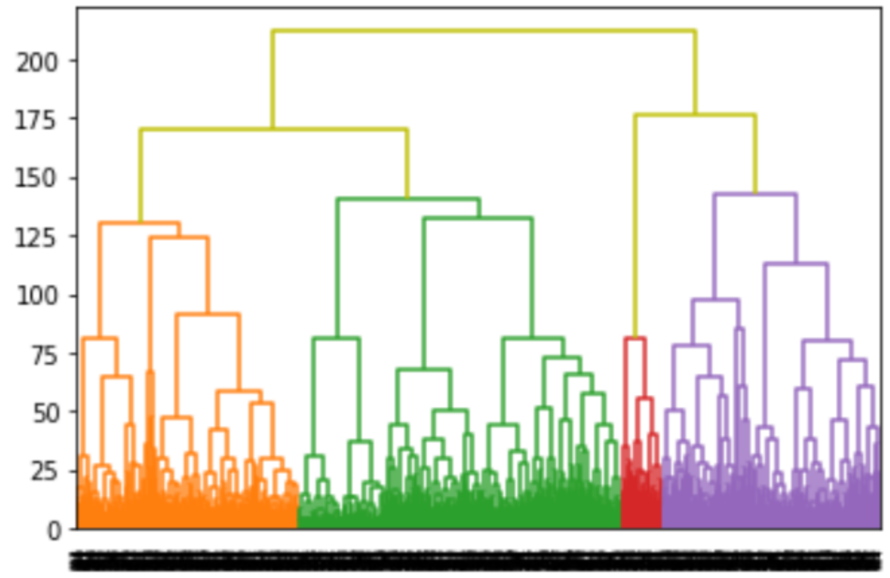
Step 4) Clustering

2.2 Hierarchical Clustering

```
pca_df_hc = pca_train.copy()
pca_df_hc = pca_df_hc.iloc[0:1000,:]
```

```
Z = fastcluster.linkage_vector(pca_df_hc.iloc[:,0:20],method='ward',metric='euclidean')
Z_df = pd.DataFrame(Z, columns=['cl_1','cl_2','dist','new_cl_size'])
```

```
dend = dendrogram(Z, above_threshold_color='y',orientation='top')
```



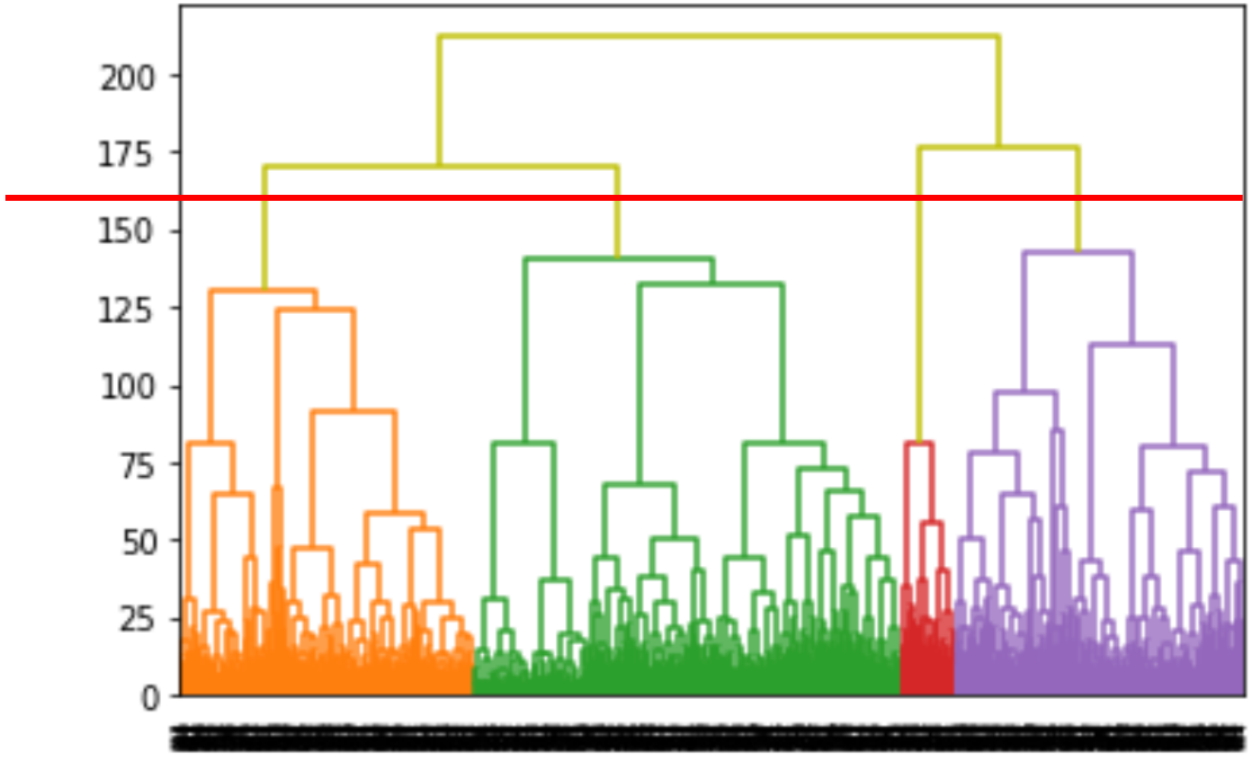
2. Hierarchical Clustering

Step 4) Clustering

```
dist_thres = 160
hclust = fcluster(Z,dist_thres,criterion='distance') # cut tree
hclust_df = pd.DataFrame(hclust, index=pca_df_hc.index, columns=['cluster'])
pca_df_hc['cluster'] = hclust_df['cluster']
print('Number of clusters :', hclust_df['cluster'].nunique())
```

```
Number of clusters : 4
```

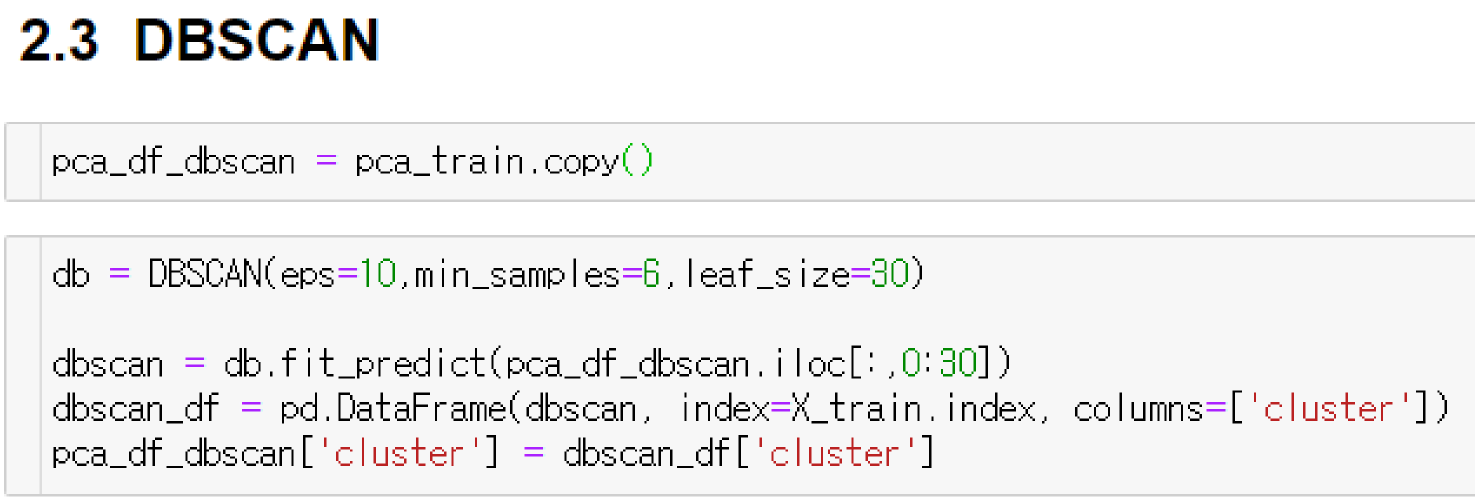
높이 160선에서 자르기 -> 4개의 Cluster



- 장점 : 상황을 눈으로 보아가면서,
  원하는 Cluster의 개수를 바로바로 정할 수 있음
- 단점 : 데이터가 많을 경우에 부적합

Step 4) Clustering

3. DBSCAN



2.3 DBSCAN

```
pca_df_dbscan = pca_train.copy()
```

```
db = DBSCAN(eps=10,min_samples=6,leaf_size=30)

dbscan = db.fit_predict(pca_df_dbscan.iloc[:,0:30])
dbscan_df = pd.DataFrame(dbscan, index=X_train.index, columns=['cluster'])
pca_df_dbscan['cluster'] = dbscan_df['cluster']
```

차원이 크면 적용하기 어려움

(기존 data 784개의 차원 충 앞의 30개만 사용했을 때에도, 다 돌아가지 못하고 멈춤…)

SUMMARY

1. Clustering은 ML의 " 비지도학습 " 방법 중 하나로,

   (Y 없이) X만을 사용하여 데이터를 비슷한 특성끼리 묶는 방법이다.

2. 대표적인 3가지 Clustering 방법

   1. Kmeans : 거리 기반의 Clustering

   2. DBSCAN : 밀도 기반의 Clustering

   3. Hierarchical Clustering : 계층적으로 Cluster 형성 (top-down & bottom- up)

3. Clustering하기 전에, (변수들 간의 scale을 통일 시킨 후) 차원 축소를 거칠 것!

4. (K-means 의 경우) Elbow method를 사용하여 적절한 K 찾기

Thank You!