머신러닝-비지도학습

군집 알고리즘

K-Means

주성분분석(PCA)

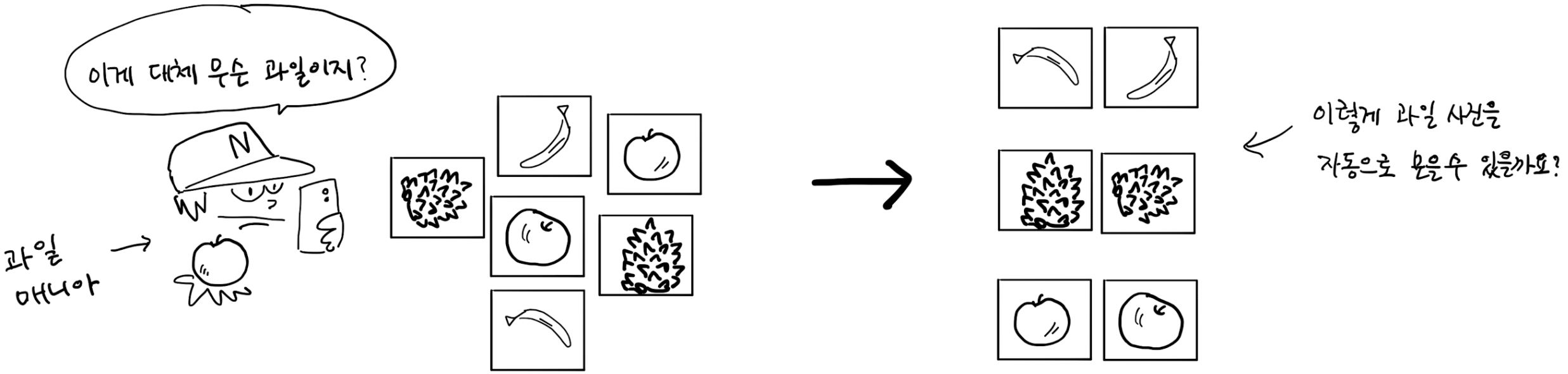
비지도학습: 클러스터링

- 군집clustering : 비슷한 샘플끼리 그룹으로 모으는 작업
- 클러스터: 군집알고리즘으로 만들어진 그룹
- 대표적인 비지도학습 알고리즘: k-Means(평균) 알고리즘

비지도 학습

한빛 마켓은 농산물 판매로 확대하며 새 이벤트를 기획 하고 있습니 다. 고객이 한빛 마켓에서 사고 싶은 과일 사진을 보내면 그중 가장 많이 요청 하는 과일을 판매 품목으로 선정하려 합니 다. 또 1 위 로 선정된 과일 사전을 보낸 고객 중 몇 명 을 뽑아 이벤트 당첨자로 선정할 겁니 다

고객이 올린 사진을 사람이 하나씩 분류하기는 어렵겠죠. 그렇다고 생선처럼 미리 과일 분류기를 훈련하기에는 고객들이 어떤 과일 사진을 보낼지 알 수 없으니 곤란합니다. 사진에 대한 **정답 (타깃) 을 알지 못하는데** 어떻 게 이 사진을 종류대로 모을 수 있을까요?



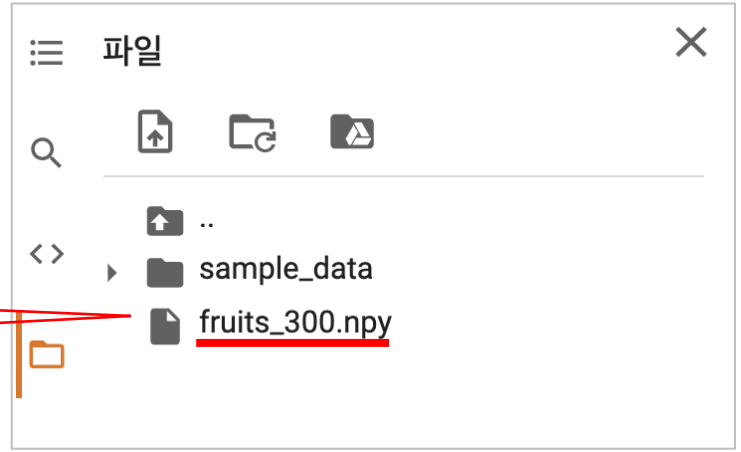
비지도학습

"사진의 **픽셀값**을 모두 **평균** 내면 비슷한 과일끼리 모이지 않을까?"

- 데이터에 대한 정답(라벨)을 알려주지 않는다.
- 정답없이 제공된 데이터의 특징과 패턴을 분석하는 학습 방법

```
!wget https://bit.ly/fruits_300 -0 fruits_300.npy
```

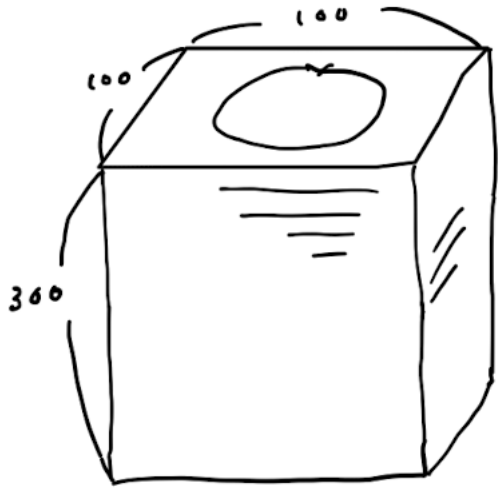
사과, 바나나, 파인애플이 각각 100개씩 포함됨



학습데이터(과일) 분석

- 과일 데이터 준비하기

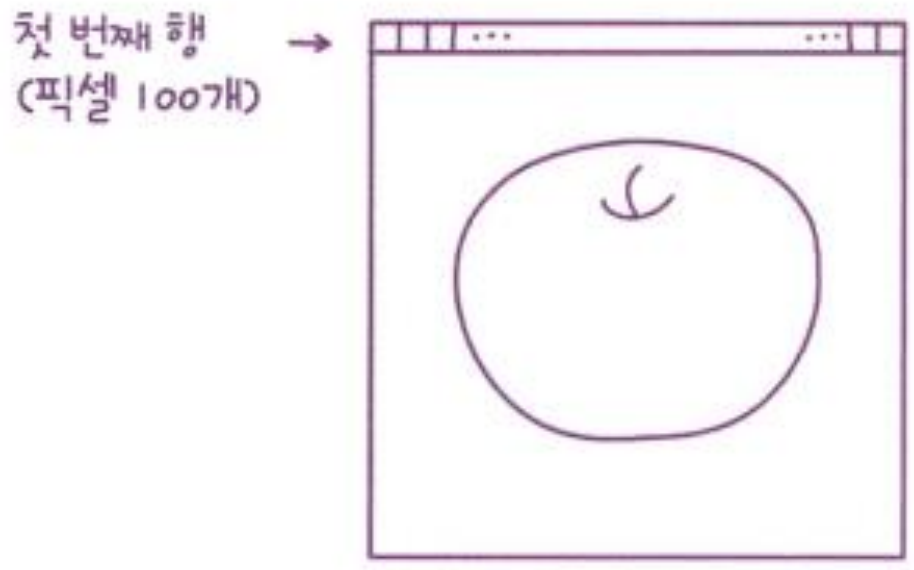
```
import numpy as np
import matplotlib.pyplot as plt

fruits = np.load('fruits_300.npy')
print(fruits.shape)
(300, 100, 100)
```



학습데이터(과일) 분석

- 샘플 확인

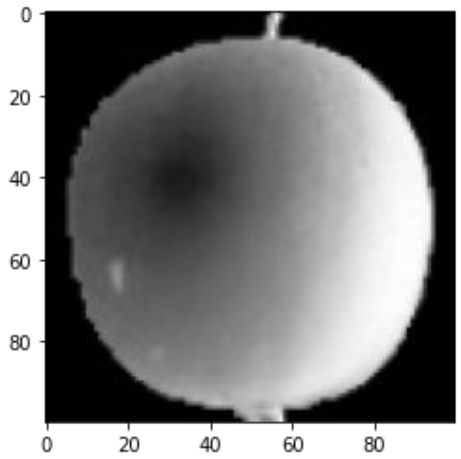
```
print(fruits[0, 0, :])
[  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   2   1
   2   2   2   2   2   2   1   1   1   1   1   1   1   1   2   3   2   1
   2   1   1   1   1   2   1   3   2   1   3   1   4   1   2   5   5   5
  19 148 192 117  28   1   1   2   1   4   1   1   3   1   1   1   1   1
   2   2   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1]
```

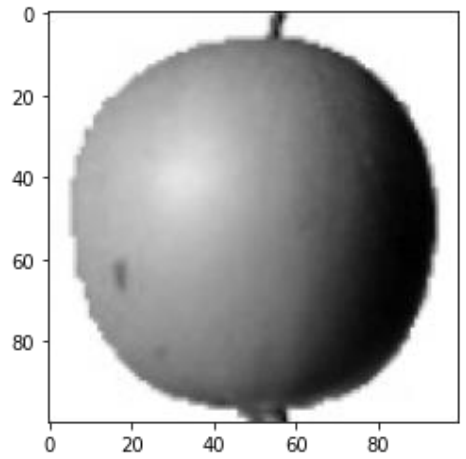


학습데이터(과일) 분석

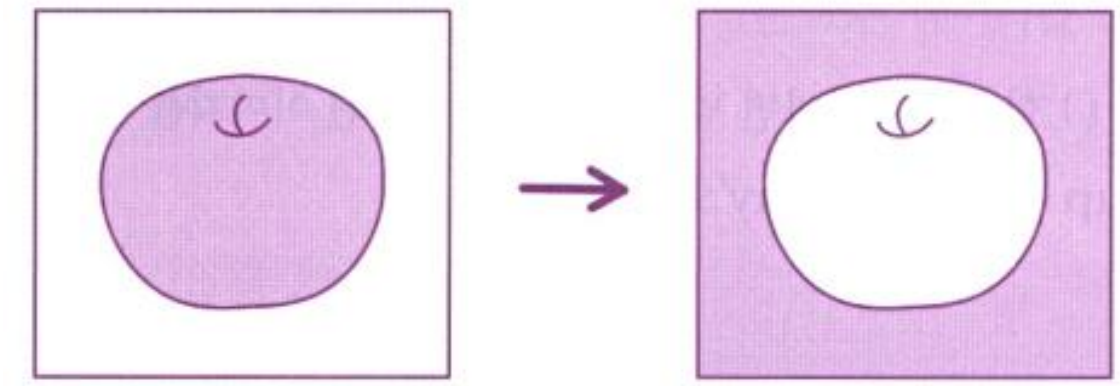
- 샘플 확인

```
plt.imshow(fruits[0], cmap='gray')
plt.show()
```





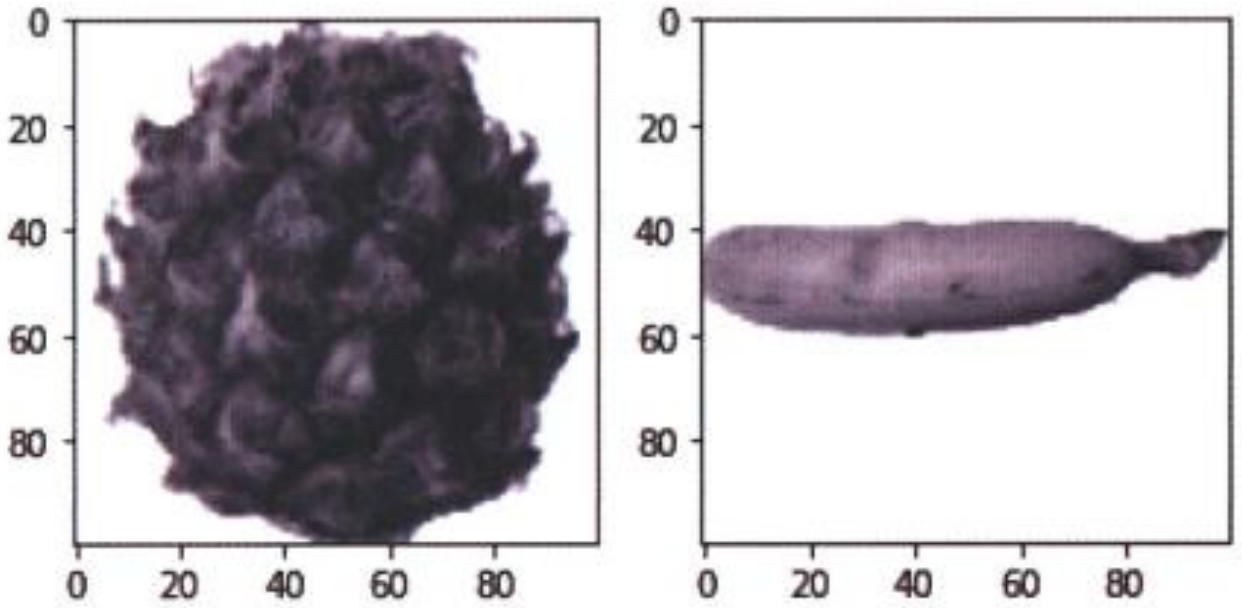
픽셀값이 0->검정색, 255->흰색



학습데이터(과일) 분석

- 샘플 확인

```
fig, axs = plt.subplots(1, 2)   # 1행 2열로 이미지 그리기
axs[0].imshow(fruits[100], cmap='gray_r')
axs[1].imshow(fruits[200], cmap='gray_r')
plt.show()
```

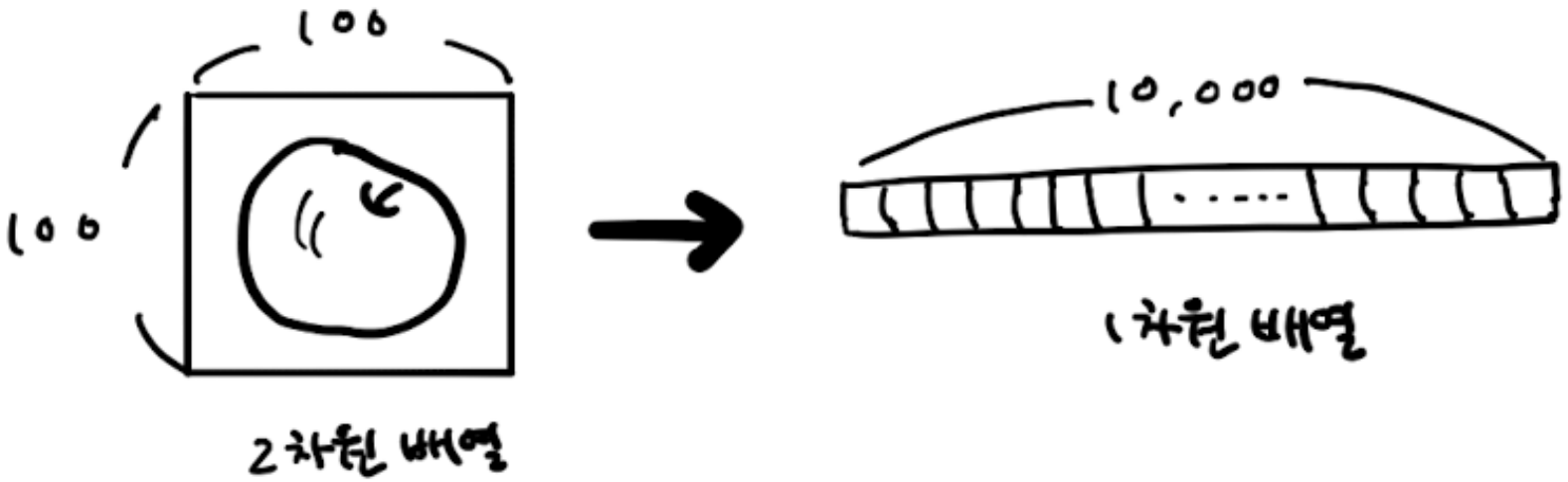


학습데이터(과일) 분석

➢ 샘플 차원 변경하기

배열 계산을 위해 2차원을 1차원으로 변환

```
apple = fruits[0:100].reshape(-1, 100*100)
pineapple = fruits[100:200].reshape(-1, 100*100)
banana = fruits[200:300].reshape(-1, 100*100)

print(apple.shape)
(100, 10000)
```

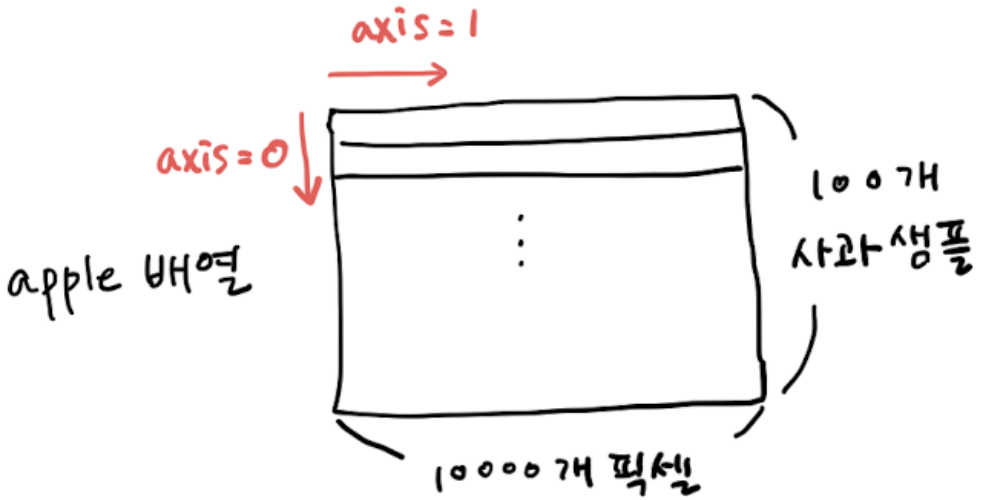


학습데이터(과일) 분석

➢ 샘플 평균 구하기

2번째 축인 **열**을 따라 평균 계산

```
np.mean(apple, axis=1)
```



A

0	1	2
3	4	5

A.mean(axis=0)

array([1.5, 2.5, 3.5])

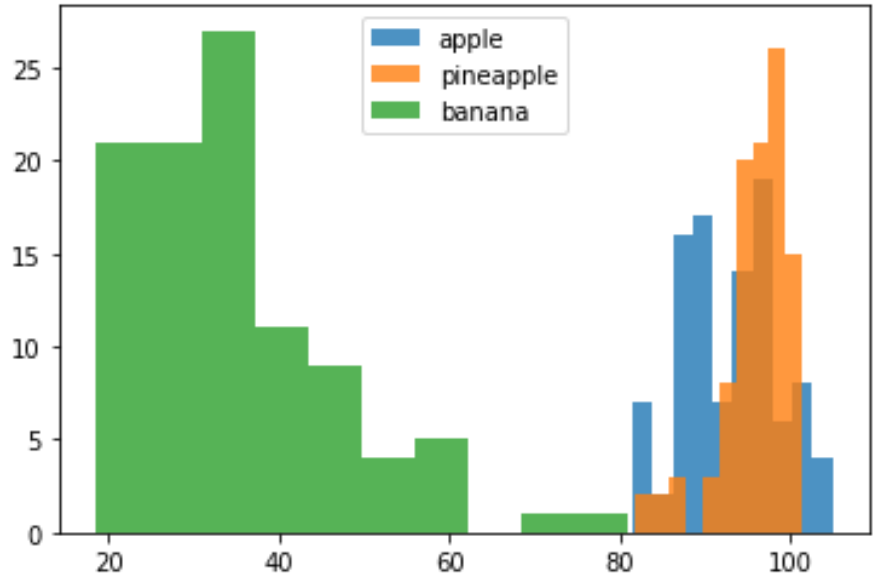
A.mean(axis=1)

array([1., 4.])

학습데이터(과일) 분석

- 샘플 평균의 히스토그램

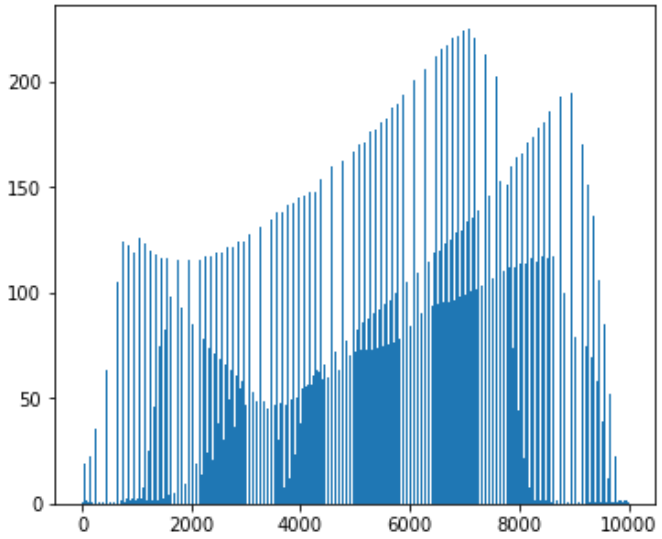
```
plt.hist(np.mean(apple, axis=1), alpha=0.8)
plt.hist(np.mean(pineapple, axis=1), alpha=0.8)
plt.hist(np.mean(banana, axis=1), alpha=0.8)
plt.legend(['apple', 'pineapple', 'banana'])
plt.show()
```

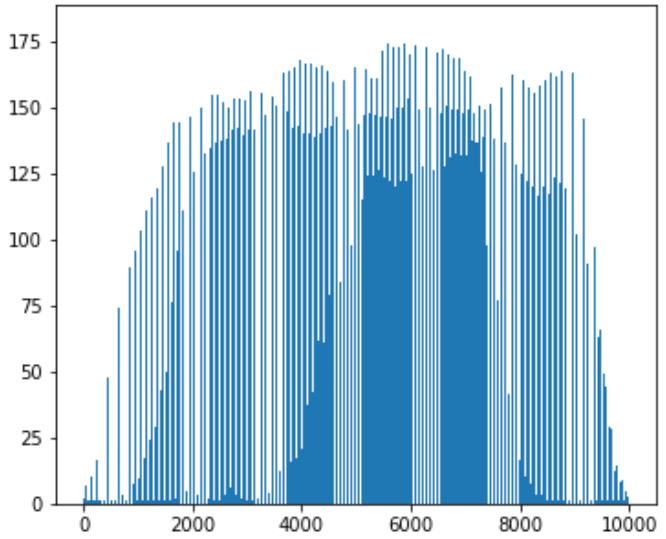


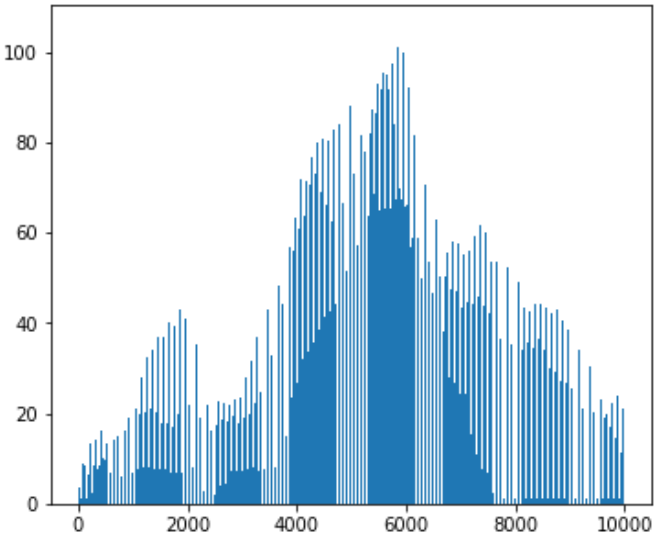
학습데이터(과일) 분석

- 픽셀별 평균의 히스토그램

```
fig, axs = plt.subplots(1, 3, figsize=(20, 5))
axs[0].bar(range(10000), np.mean(apple, axis=0))
axs[1].bar(range(10000), np.mean(pineapple, axis=0))
axs[2].bar(range(10000), np.mean(banana, axis=0))
plt.show()
```



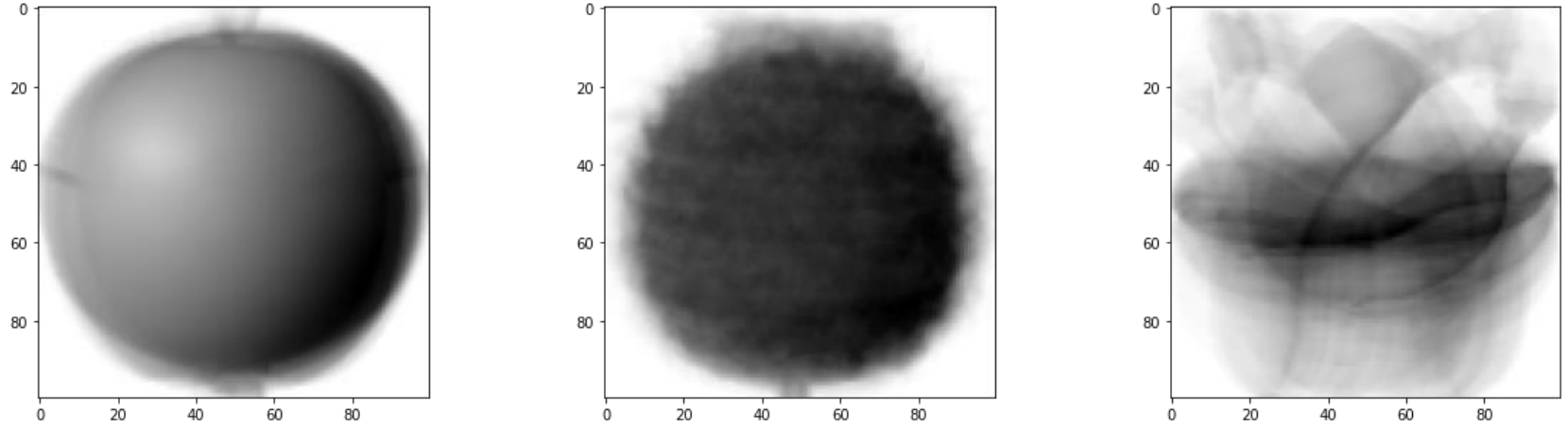




학습데이터(과일) 분석

- 평균 픽셀값 이미지 그리기

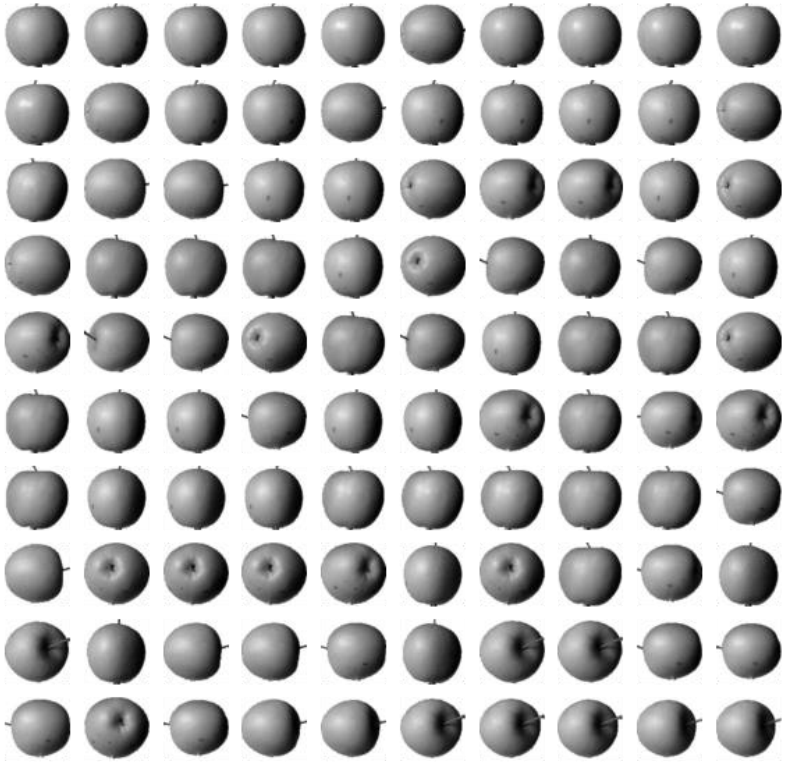
```
apple_mean = np.mean(apple, axis=0).reshape(100, 100)
pineapple_mean = np.mean(pineapple, axis=0).reshape(100, 100)
banana_mean = np.mean(banana, axis=0).reshape(100, 100)

fig, axs = plt.subplots(1, 3, figsize=(20, 5))
axs[0].imshow(apple_mean, cmap='gray_r')
axs[1].imshow(pineapple_mean, cmap='gray_r')
axs[2].imshow(banana_mean, cmap='gray_r')
plt.show()
```



학습데이터(과일) 분석

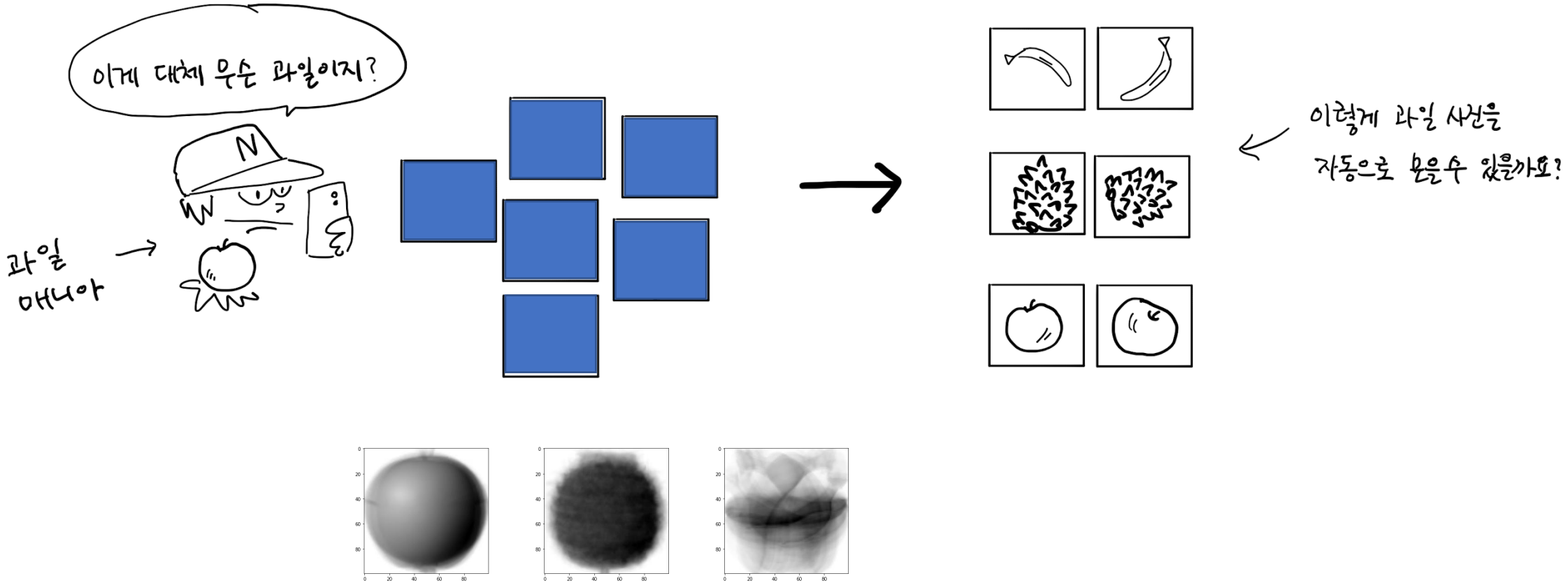
- 평균과 가까운 사진 고르기

```
abs_diff = np.abs(fruits - apple_mean)
abs_mean = np.mean(abs_diff, axis=(1,2))
print(abs_mean.shape)
(300,)

apple_index = np.argsort(abs_mean)[:100]
fig, axs = plt.subplots(10, 10, figsize=(10,10))
for i in range(10):
    for j in range(10):
        axs[i, j].imshow(fruits[apple_index[i*10 + j]], cmap='gray_r')
        axs[i, j].axis('off')
plt.show()
```



군집화(Clustering)

사진에 어떤 과일이 있는지 알지 못할 경우에는 어떻게 군집화할까?



k— Means 작동 방식

1.Initialization:

- 데이터로부터 임의의 K개 선택, 클러스터 중심으로 초기화

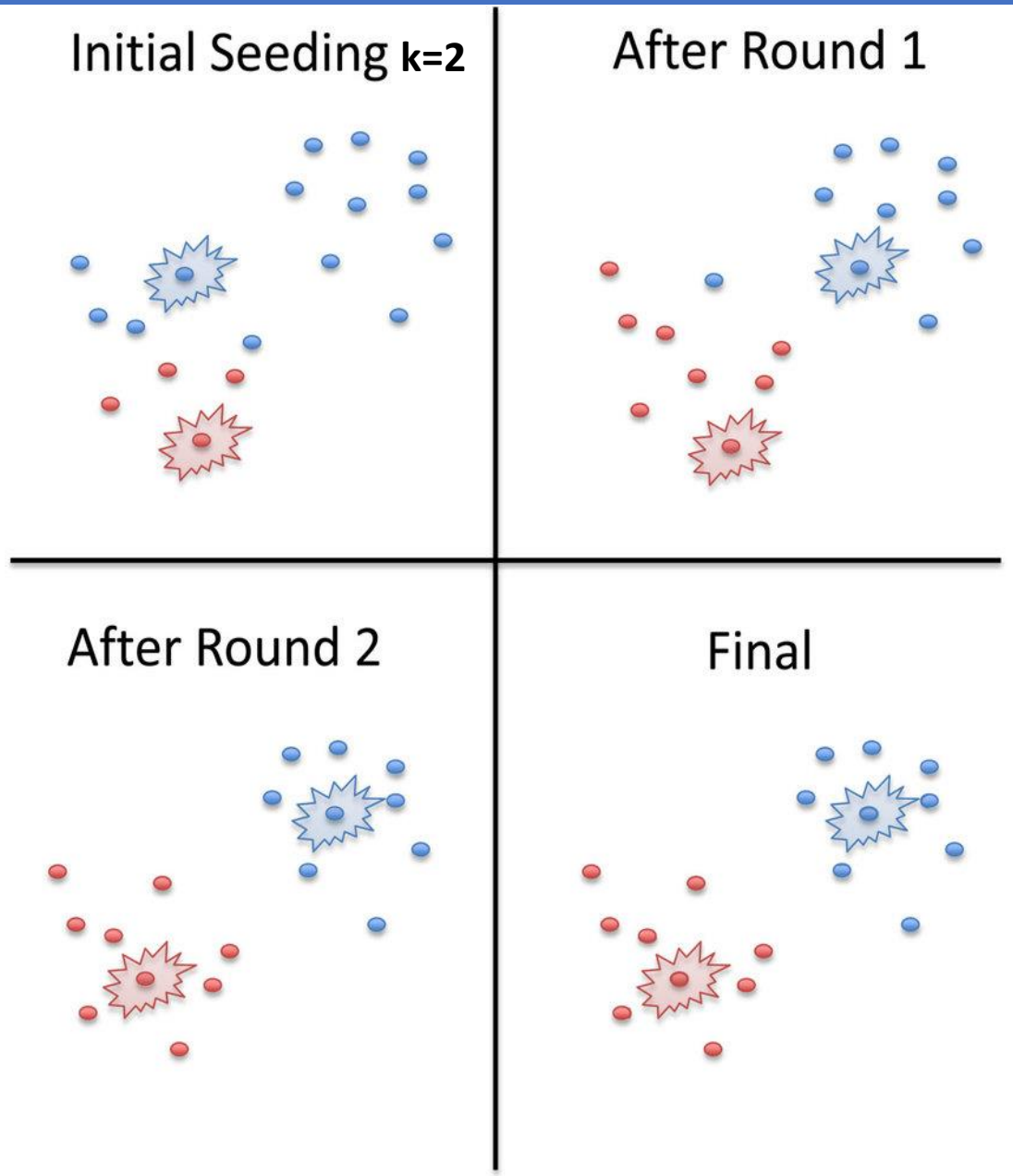
2.Assignment:

- 각 데이터와 클러스터 중심과의 유클리드 거리(Euclidean distance) 계산, 가장 가까운 클러스터 중심으로 데이터를 클러스터링(묶음)함.

3.Update

- 각 클러스터에 대한 중심을 클러스터 속한 데이터 평균 계산으로 정함

4. 클러스터의 중심점이 더 이상 변하지 않을 때까지 2와 3단계를 반복함



Scikit-Learn의 k-Means 클래스 활용

1. 훈련 데이터 준비

!wget https://bit.ly/fruits_300 -0 fruits_300.npy

```
import numpy as np
fruits = np.load('fruits_300.npy')
fruits_2d = fruits.reshape(-1, 100*100)
```

k— 평균 모델 훈련을 위해 (샘플 개수, 너비 , 높이) 크기의 3차원 배열을 (샘플 개수, 너 비 x 높이) 크기를 가진 2차원 배열로 변경

Scikit-Learn의 k-Means 클래스 활용

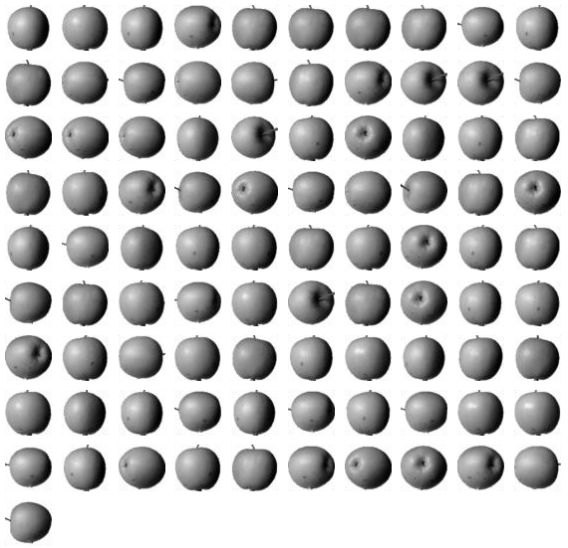
2. 모델 훈련

```
from sklearn.cluster import KMeans

km = KMeans(n_clusters=3, random_state=42)
km.fit(fruits_2d) #훈련데이터만 전달

print(km.labels_) # 모델의 label_ 속성값에 결과 저장됨
[0 0 0 0 0 2 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 0 0 0 2 0 2 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 2 2 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 2 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1]

print(np.unique(km.labels_, return_counts=True))
(array([0, 1, 2], dtype=int32), array([ 91,  98, 111]))
```

Scikit-Learn의 k-Means 클래스 활용

- km.label_==0 인 클러스터 샘플 확인

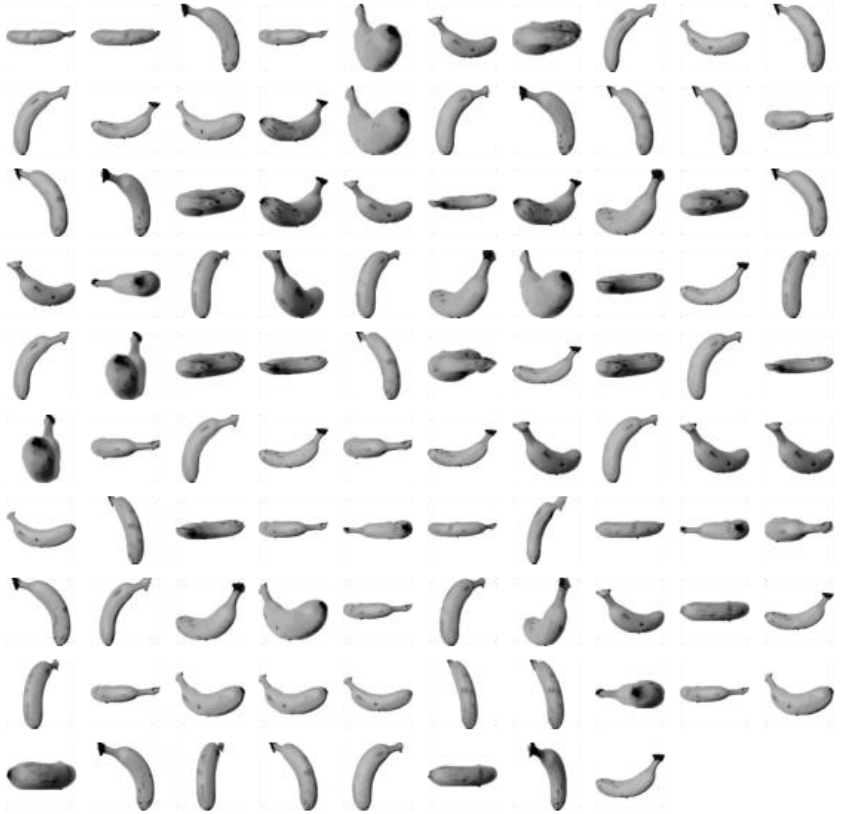
```
def draw_fruits(arr, ratio=1):
    n = len(arr)    # n은 샘플 개수입니다
    # 한 줄에 10개씩 이미지를 그립니다. 샘플 개수를 10으로 나누어 전체 행 개수를 계산합니다.
    rows = int(np.ceil(n/10))
    # 행이 1개 이면 열 개수는 샘플 개수입니다. 그렇지 않으면 10개입니다.
    cols = n if rows < 2 else 10
    fig, axs = plt.subplots(rows, cols,
                            figsize=(cols*ratio, rows*ratio), squeeze=False)
    for i in range(rows):
        for j in range(cols):
            if i*10 + j < n:    # n 개까지만 그립니다.
                axs[i, j].imshow(arr[i*10 + j], cmap='gray_r')
            axs[i, j].axis('off')
    plt.show()


draw_fruits(fruits[km.labels_==0])
```



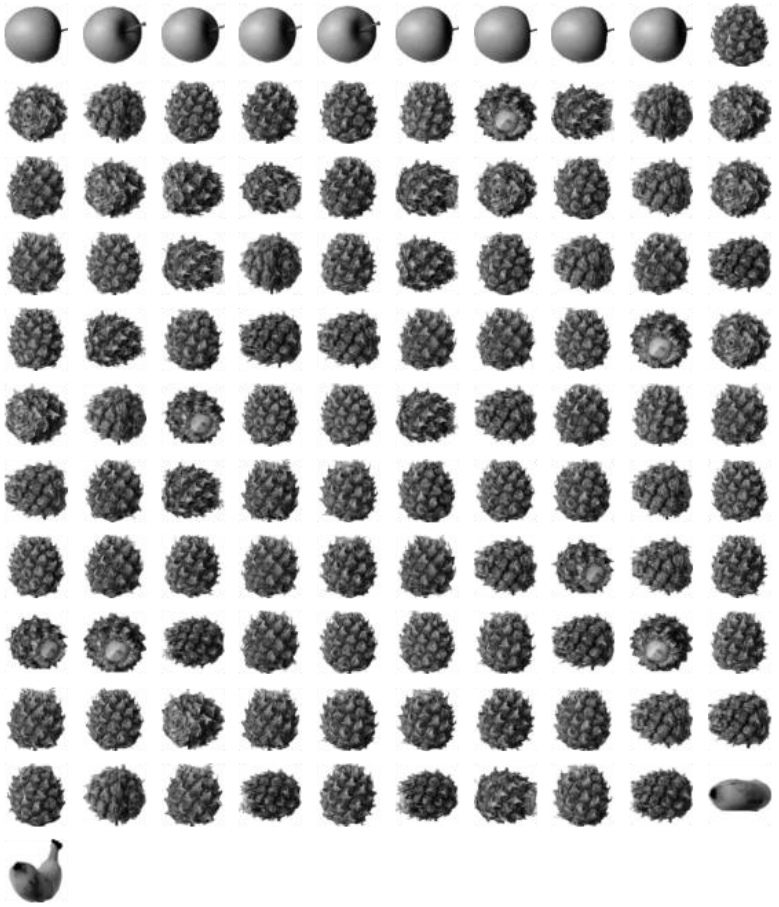
Scikit-Learn의 k-Means 클래스 활용

- km.label_==1, km.label_==2 인 클러스터 샘플 확인

```
draw_fruits(fruits[km.labels_==1])
```



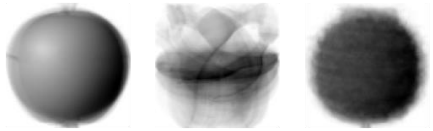
```
draw_fruits(fruits[km.labels_==2])
```



Scikit-Learn의 k-Means 클래스 활용

- 클러스터 중심 속성(cluster_centers_)

```
draw_fruits(km.cluster_centers_.reshape(-1, 100, 100), ratio=3)
```



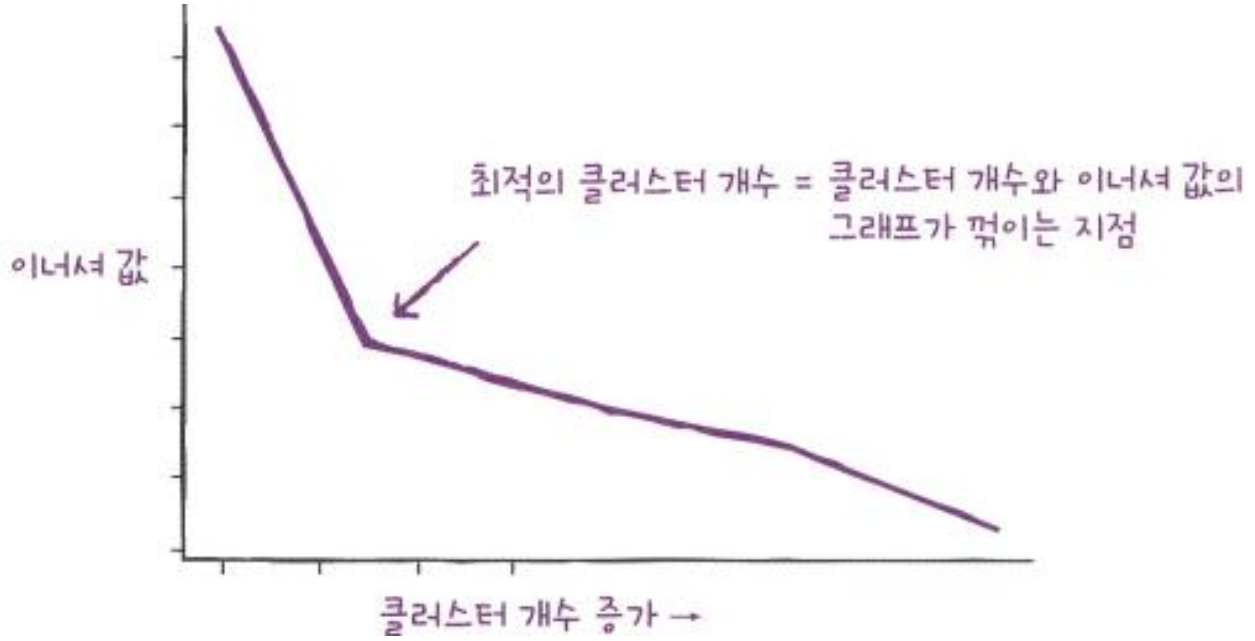
#사과, 바나나, 파인애플 픽셀의 평균 낸 이미지와 비슷한 결과 출력

```
print(km.transform(fruits_2d[100:101])) # 입력데이터에서 클러스터 중심까지의 거리 계산 함수
[[5267.70439881 8837.37750892 3393.8136117 ]]

print(km.predict(fruits_2d[100:101])) # 입력데이터가 속한  클러스터 예측 함수
[2]

draw_fruits(fruits[100:101])
```

```
print(km.n_iter_) #입력데이터가 # 최적의 클러스터를 찾는 반복 횟수
3
```

k-Means 에서 최적의 k 찾기

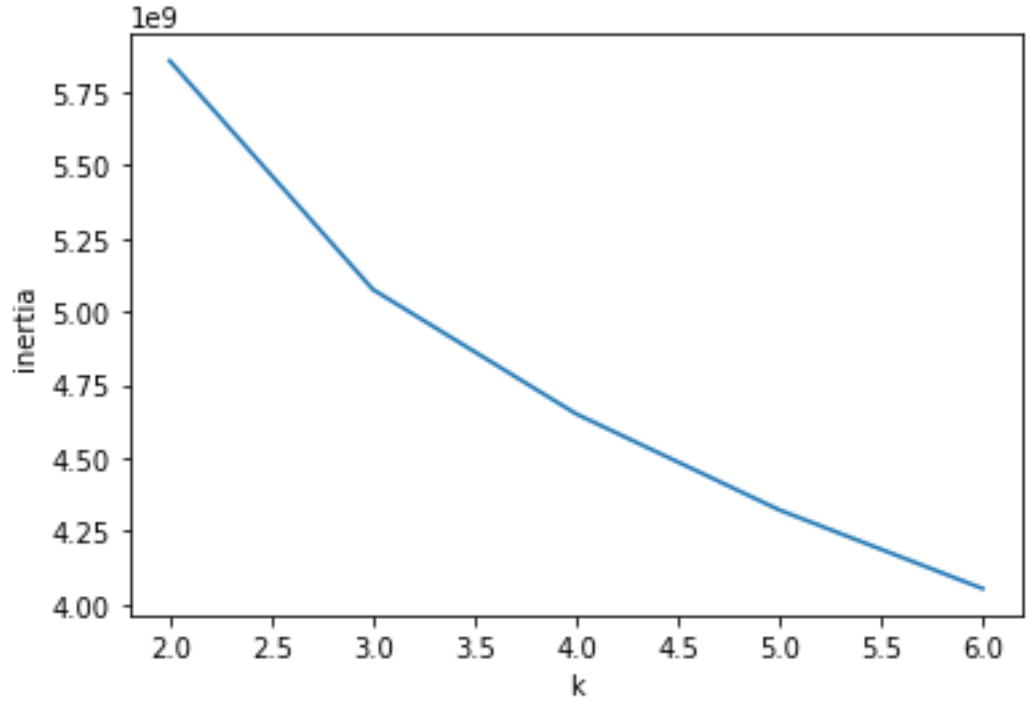
- 엘보우(elbow) 기법
  - 클러스터의 개수를 늘려가면서 **이너셔**의 변화를 관찰, 최적의 클러스터 개수 찾는 방법
  - K-Means 모델의 inertia_ 속성에 저장됨



- 이너셔(inertia)
  - 클러스터에 속한 샘플이 얼마나 가깝게 모여 있는지를 나타내는 값
  - 클러스터 중심과 클러스터에 속한 샘플 사이의 거리 제곱합
  - 클러스터 개수 늘어나면 클러스터에 속한 샘플 개수가 줄어들므로 이너셔도 작아짐

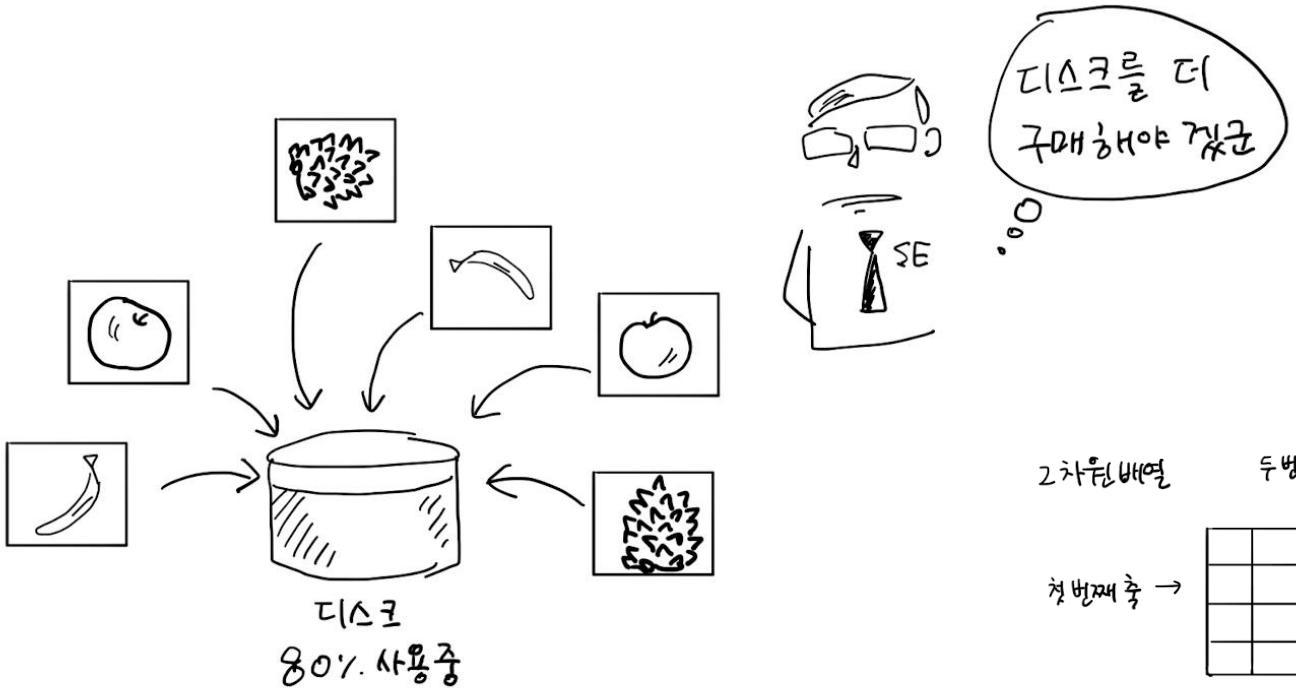
k-Means 에서 최적의 k 찾기

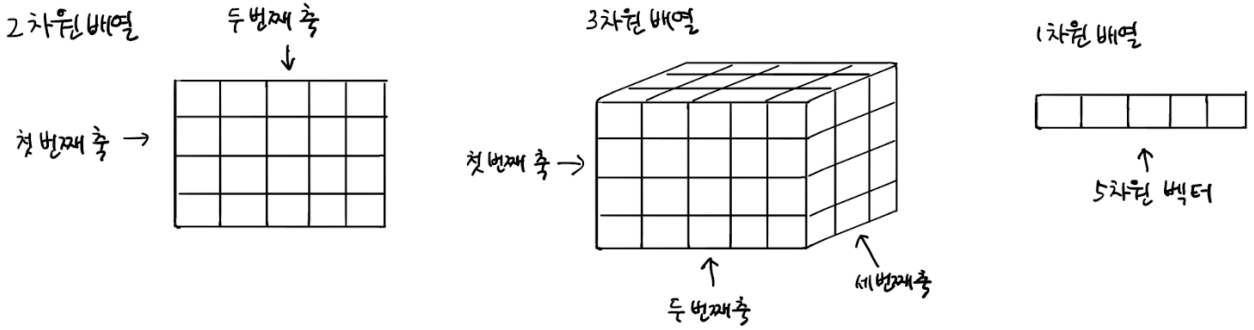
```
inertia = []
for k in range(2, 7):
    km = KMeans(n_clusters=k, random_state=42)
    km.fit(fruits_2d)
    inertia.append(km.inertia_)

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



차원 축소(Dimensionality Reduction)

- 데이터가 가진 속성=특성=차원(dimension)
- 차원축소는 데이터의 특성 수(차원 수)를 줄이는 과정으로 고차원 데이터(특성이 많은 데이터)를 보다 적은 차원으로 변환하여, 데이터의 핵심 정보를 유지하면서 분석 및 시각화를 용이하게 하는 기술
  예) 주성분분석(PCA, Principal Component Analysis)

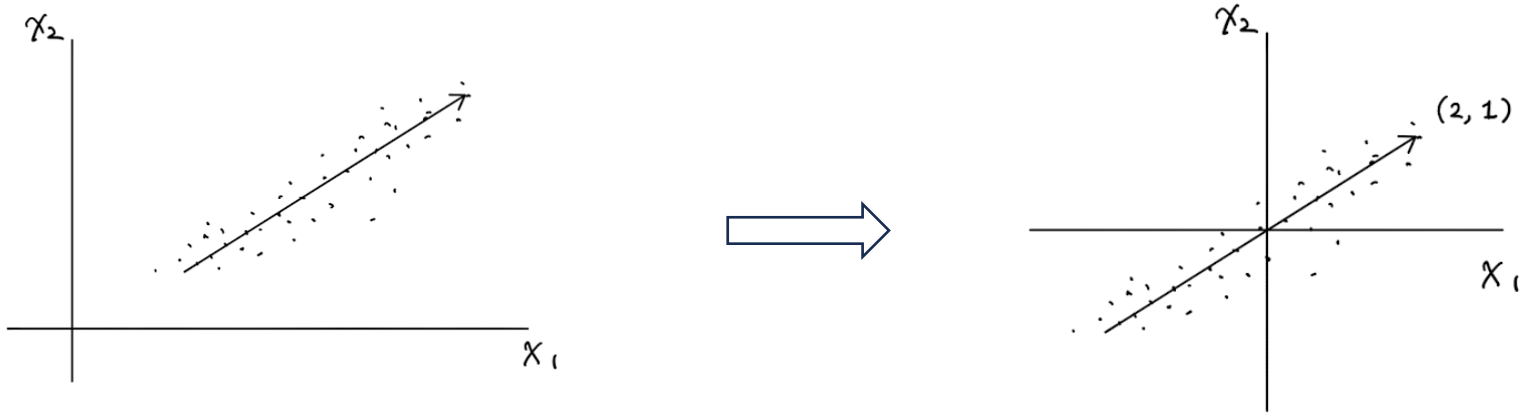




주성분 분석(PCA)

➢ 데이터의 분산이 가장 큰 축을 찾아서 데이터를 재구성하는 방법

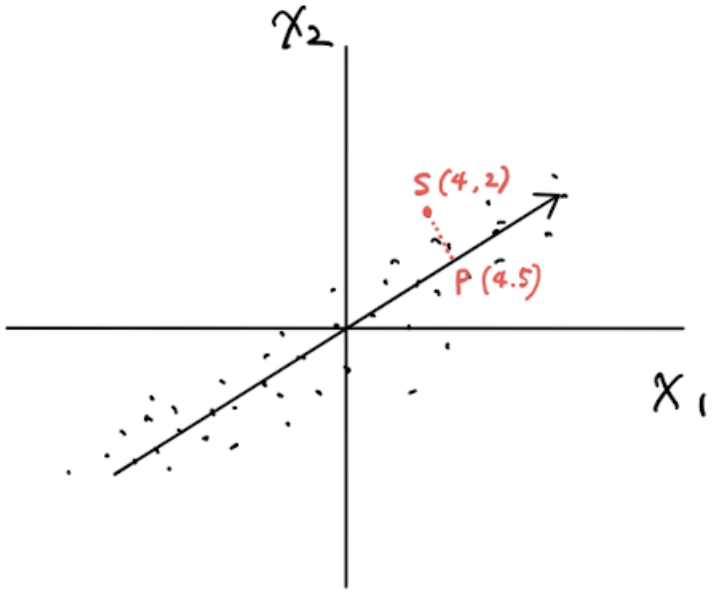
예) x1, x2의 2특성을 갖는 데이터 표현

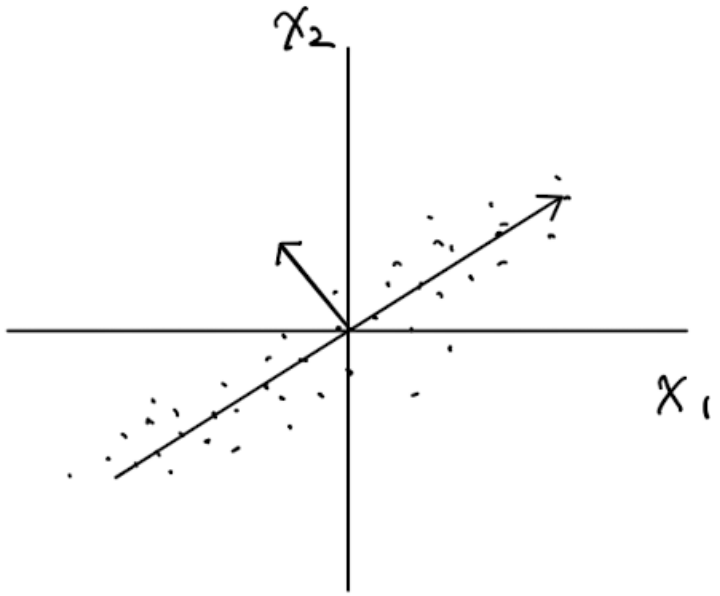


• x1, x2 축을 갖는 데이터의 산점도를 그리고, 산점도를 가장 잘 표현하는 방향을 찾음

• 산점도를 가장 잘 표현하는 방향을 x1, x2 축의 원점으로 이동시킴(벡터화). 찾아진 벡터는 첫번째 주성분(벡터)이 된다.

주성분 분석(PCA)



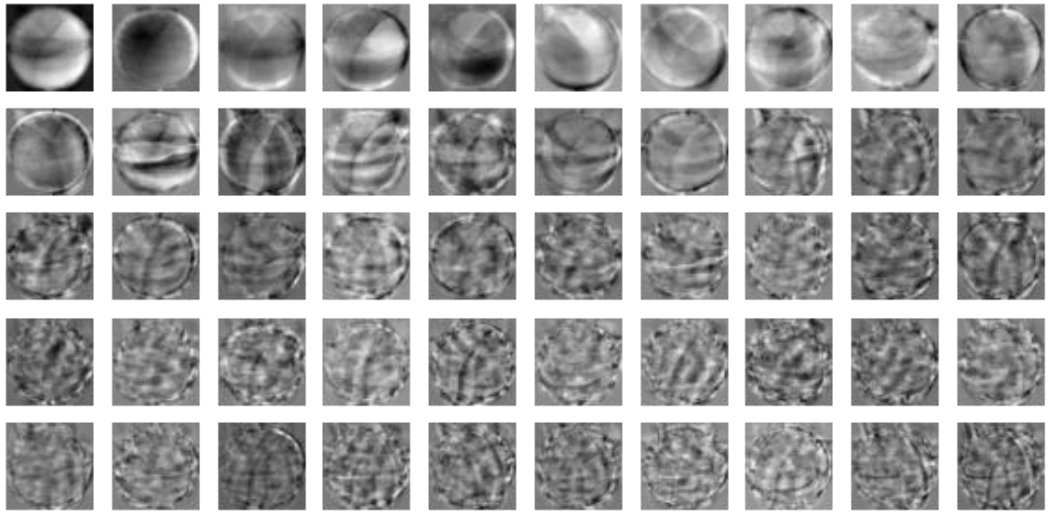


예를 들어 x1, x2 축으로 표현되는 샘플 S(4, 2)는 첫번째 주성분(벡터)에 직각으로 투영하면 1차원 데이터 p(4.5)가 됨

첫번째 주성분(벡터)에 수직이고 분산이 다음으로 큰 방양을 찾으면 두번째 주성분이 된다.
X1, x2 특성축을 가질 경우에는 2개의 주성분을 갖는다. (원본 특성의 개수만큼)

Scikit-Learn PCA 클래스 활용

```
from sklearn.decomposition import PCA

pca = PCA(n_components=50) # 주성분개수 50으로 지정
pca.fit(fruits_2d)

print(pca.components_.shape)
(50, 10000)  # 10000 차원의 데이터를 50차원으로 줄임

draw_fruits(pca.components_.reshape(-1, 100, 100))

print(fruits_2d.shape)
(300, 10000)

fruits_pca = pca.transform(fruits_2d)  # pca 모델을 사용하여 데이터를 50차원 벡터 공간으로 변환
print(fruits_pca.shape)
(300, 50)  # 300개의 이미지가 50차원의 벡터로 변환
```

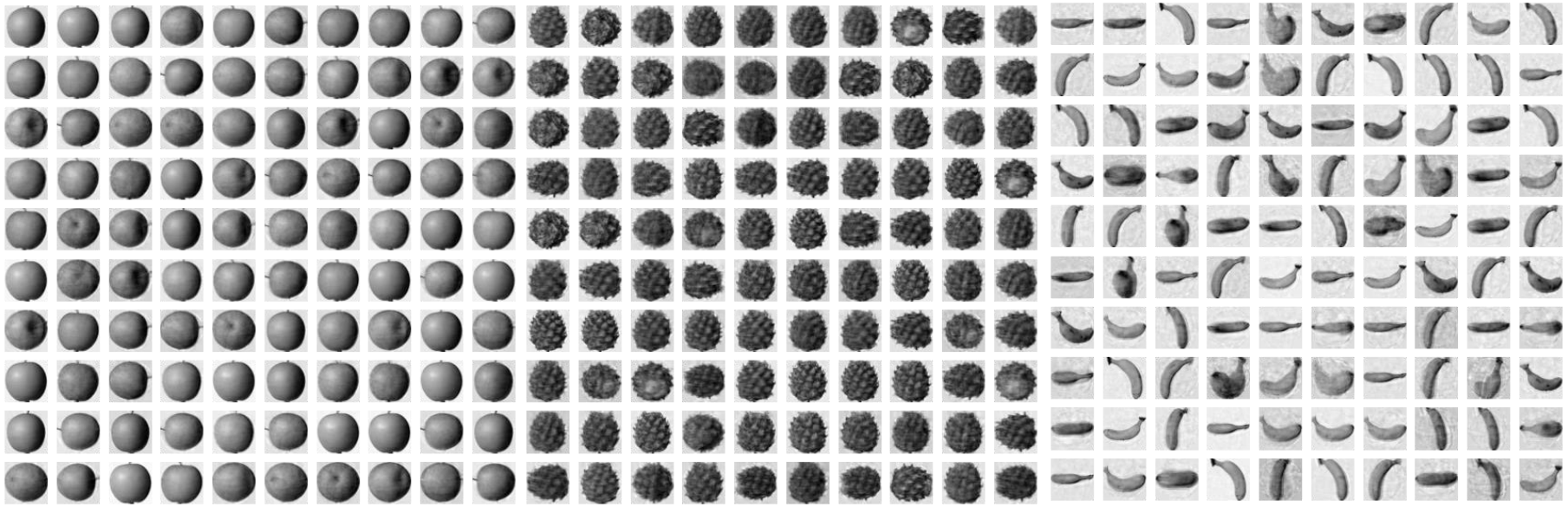


Scikit-Learn PCA 클래스 활용

- 이미지 복원

```
fruits_inverse = pca.inverse_transform(fruits_pca)
print(fruits_inverse.shape)
(300, 10000)

fruits_reconstruct = fruits_inverse.reshape(-1, 100, 100)

for start in [0, 100, 200]:
    draw_fruits(fruits_reconstruct[start:start+100])
    print("\n")
```

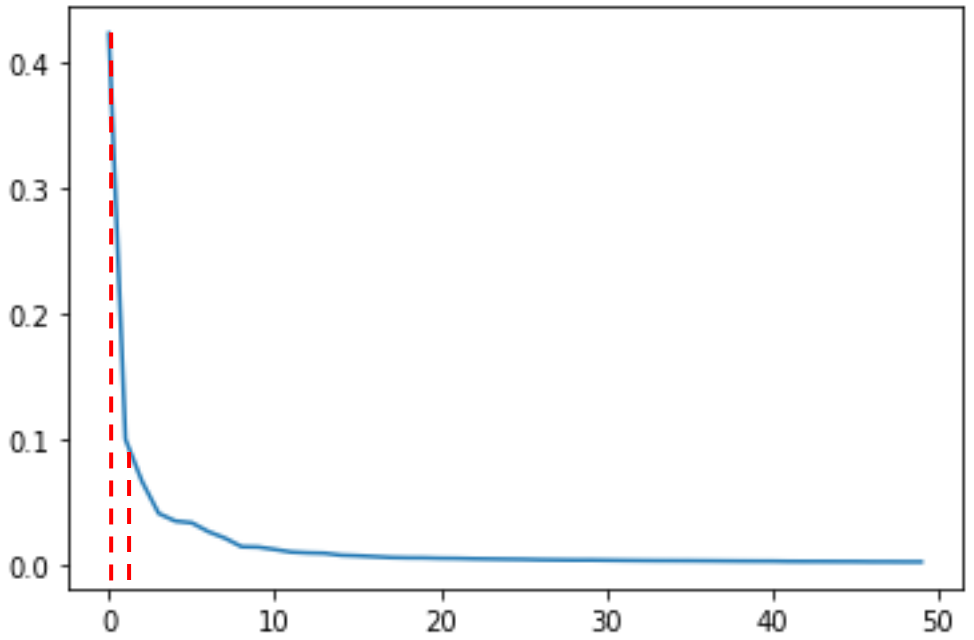
저차원공간 (50차원) 에서 다시 고차원 공간(10000차원)으로 데이터를 변환



Scikit-Learn PCA 클래스 활용

```
print(np.sum(pca.explained_variance_ratio_))
0.9215015476605593

plt.plot(pca.explained_variance_ratio_)
```

각 주성분이 데이터 분산의 비율 합--# 50개의 주성분이 데이터를 얼마나 설명하는지를 나타냄
이 값이 1에 가까울수록, 선택한 50개의 주성분이 원래 데이터의 정보를 거의 모두 보존하고 있다는 의미



Scikit-Learn PCA 클래스 활용

➢ pca.explained_variance_ratio_

- PCA 모델에서 각 주성분이 데이터의 분산을 얼마나 설명하는지를 나타내는 값들을 저장한 속성
- 각 값은 특정 주성분이 원래 데이터의 전체 분산 중에서 차지하는 비율을 의미

예) pca.explained_variance_ratio_가 [0.3, 0.2, 0.1, 0.1, 0.1, 0.1, 0.05, 0.05]인 경우:

- 첫 번째 주성분이 30%를 설명 / 두 번째 주성분이 20%를 설명 / ... / 여덟 번째 주성분이 5%를 설명.

- 이 경우 np.sum(pca.explained_variance_ratio_)는 1.0 (또는 100%)으로 모든 주성분이 원래 데이터의 전체 분산을 완전히 설명한다는 의미를 나타냄.

- PCA에서 n_components 값 설정
  1. 수동 설정: PCA(n_components=50) 와 같이 명시적으로 주성분의 개수를 설정하면 pca.n_components_는 50이 된다.
  2. 자동 결정: PCA 객체를 생성할 때 n_components를 명시하지 않거나, 특정 조건(예: 주성분이 분산의 50% 비율을 설명하도록 설정한다면 n_components=0.5)으로 지정하면 PCA는 데이터에 따라 최적의 주성분 개수를 결정한다.

PCA를 분류기와 함께 사용하기

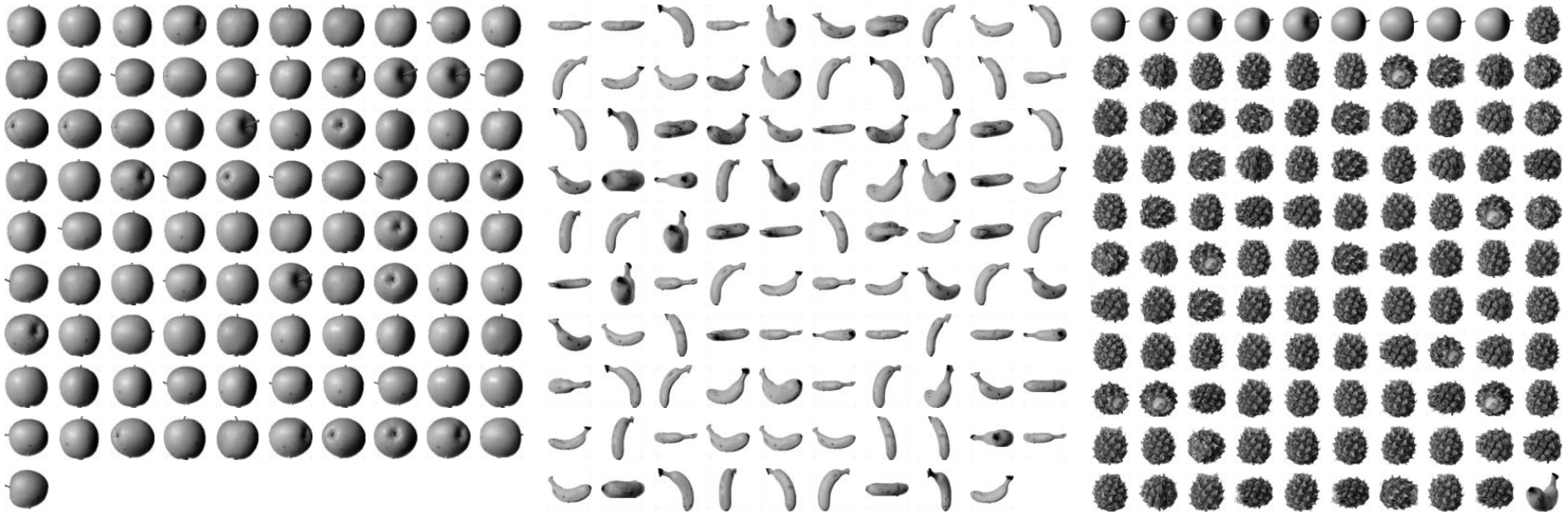
```
lr = LogisticRegression()
target = np.array([0] * 100 + [1] * 100 + [2] * 100)

scores = cross_validate(lr, fruits_2d, target)
print(np.mean(scores['test_score']))
0.9966666666666667
print(np.mean(scores['fit_time']))
1.8380496025085449

scores = cross_validate(lr, fruits_pca, target)
print(np.mean(scores['test_score']))
1.0
print(np.mean(scores['fit_time']))
0.03938336372375488
```

PCA를 분류기와 함께 사용하기

```
pca = PCA(n_components=0.5) # 주성분의 분산 합이 50% 비율이 되도록 최적의 개수 결정
pca.fit(fruits_2d)
print(pca.n_components_)
2

fruits_pca = pca.transform(fruits_2d)
print(fruits_pca.shape)
(300, 2)

scores = cross_validate(lr, fruits_pca, target)
print(np.mean(scores['test_score']))
0.9933333333333334
print(np.mean(scores['fit_time']))
0.04815754890441894
```

PCA를 군집과 함께 사용하기

```
km = KMeans(n_clusters=3, random_state=42)
km.fit(fruits_pca)

print(np.unique(km.labels_, return_counts=True))
(array([0, 1, 2], dtype=int32), array([ 91,  99, 110]))
```



PCA 차원축소 시각화

```
for label in range(0, 3):
    data = fruits_pca[km.labels_ == label]
    plt.scatter(data[:,0], data[:,1])
plt.legend(['apple', 'banana', 'pineapple'])
plt.show()
```