

## 학습목표

- 타깃이 없는 데이터를 사용하는 비지도 학습과 대표적인 알고리즘을 소개합니다.
- 대표적인 군집 알고리즘인 k-평균을 배웁니다.
- 대표적인 차원 축소 알고리즘인 주성분 분석(PCA)을 배웁니다.

Chapter

# 06

## 비지도 학습

비슷한 과일끼리 모으자!

# 06-1

## 군집 알고리즘

핵심 키워드

비지도 학습

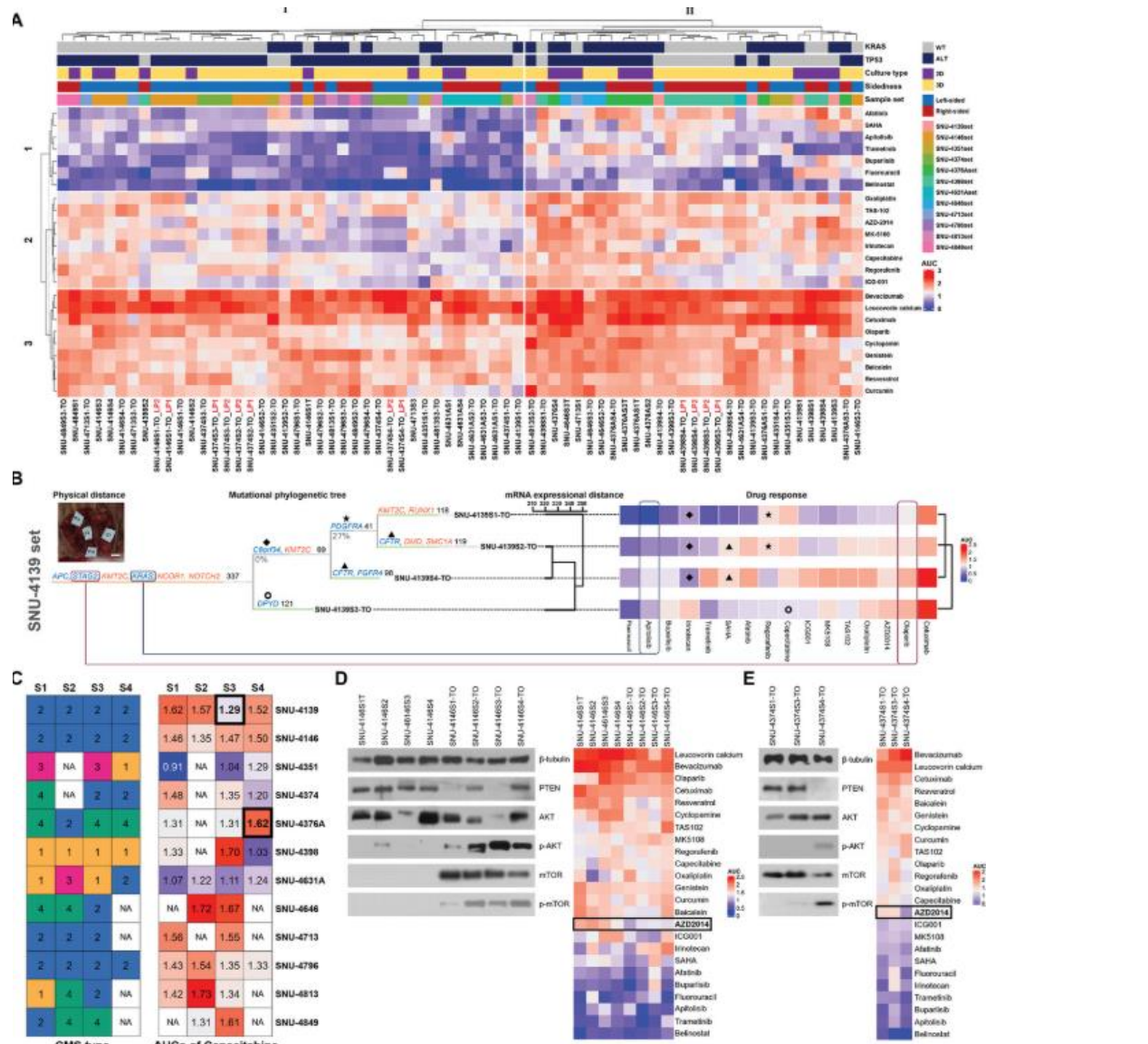
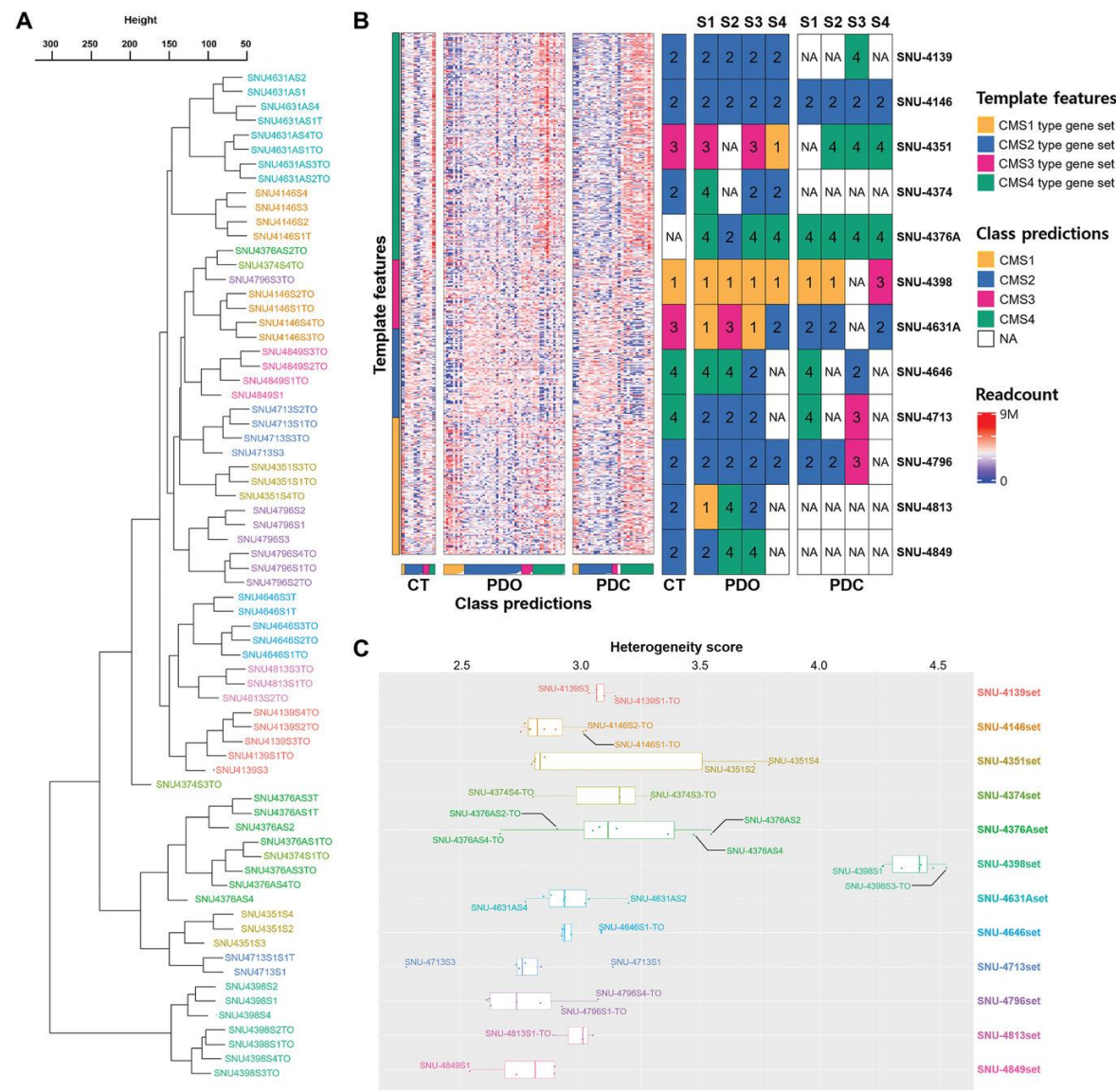
히스토그램

군집

흑백 사진을 분류하기 위해 여러 가지 아이디어를 내면서 비지도 학습과 군집 알고리즘에 대해 이해합니다.

## ▶ 키워드로 끝내는 핵심 포인트

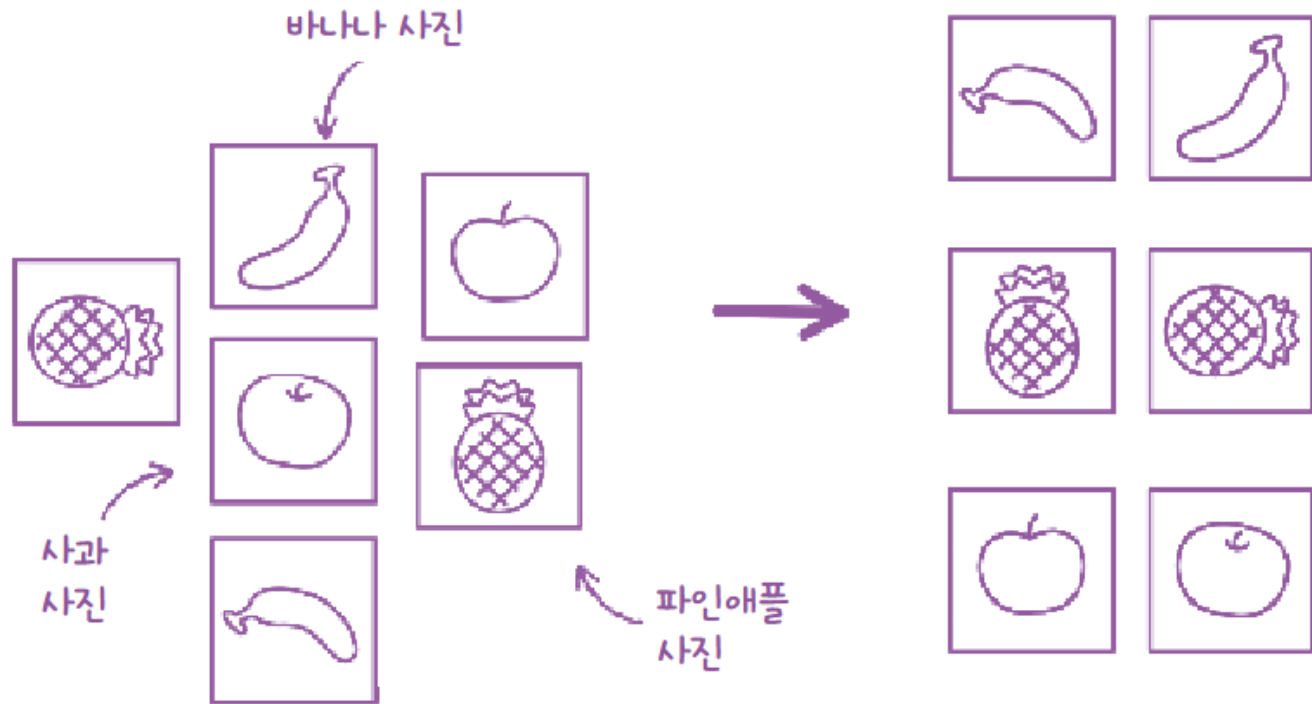
- **비지도 학습**은 머신러닝의 한 종류로 훈련 데이터에 타깃이 없습니다. 타깃이 없기 때문에 외부의 도움 없이 스스로 유용한 무언가를 학습해야 합니다. 대표적인 비지도 학습 작업은 군집, 차원 축소 등입니다.
- **히스토그램**은 구간별로 값이 발생한 빈도를 그래프로 표시한 것입니다. 보통 x축이 값의 구간(계급)이고 y축은 발생 빈도(도수)입니다.
- ★ **군집**은 비슷한 샘플끼리 하나의 그룹으로 모으는 대표적인 비지도 학습 작업입니다. 군집 알고리즘으로 모은 샘플 그룹을 클러스터라고 부릅니다.



과일은 누가 뭐래도  
사과가 제일이지.



과일 매니아



← 이렇게 과일 사진을  
자동으로 모을  
수 있을까요?



## 타깃을 모르는 비지도 학습

혼공머신은 타깃을 모르는 사진을 종류별로 분류하려 합니다. 이렇게 타깃이 없을 때 사용하는 머신러닝 알고리즘이 있습니다. 바로 **비지도 학습** unsupervised learning입니다. 사람이 가르쳐 주지 않아도 데이터에 있는 무언가를 학습하는 거죠. 혼공머신은 대체 어떻게 해야 할지 한참을 고민했습니다. 그때 김 팀장이 흥미로운 아이디어를 제안했습니다.

“사진의 픽셀값을 모두 평균 내면 비슷한 과일끼리 모이지 않을까?”

“글쎄요. 확신할 수는 없지만 해 봐야 알 것 같습니다.”

“모델을 만들기 위해 개발 팀에서 사진 300장을 받아 봤네. 같이 고민해 보자구.”

그럼 데이터를 준비하고 픽셀값을 이용해서 사진을 분류하겠습니다.

## 과일 사진 데이터 준비하기

김 팀장이 준비한 과일 데이터는 사과, 바나나, 파인애플을 담고 있는 흑백 사진입니다. 이 데이터는 넘파이 배열의 기본 저장 포맷인 npy 파일로 저장되어 있습니다. 넘파이에서 이 파일을 읽으려면 먼저 코랩으로 다운로드해야 합니다. 코랩에서 다음 명령을 실행해 파일을 다운로드하세요.

### + 여기서 잠깐

### 과일 데이터셋의 출처

이 과일 데이터는 캐글에 공개된 데이터셋입니다.


- <https://www.kaggle.com/moltean/fruits>

### 손코딩

```
!wget https://bit.ly/fruits_300_data -O fruits_300.npy
```


↓  
리눅스 shell 명령으로 이해

그럼 이 파일에서 데이터를 로드하겠습니다. 먼저 넘파이와 맷플롯립 패키지를 임포트합니다.

손코딩

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


넘파이에서 npy 파일을 로드하는 방법은 아주 간단합니다. load() 메서드에 파일 이름을 전달하는 것이 전부죠.

손코딩

```
fruits = np.load('fruits_300.npy')
```



fruits는 넘파이 배열이고 fruits\_300.npy 파일에 들어 있는 모든 데이터를 담고 있습니다. fruits 배열의 크기를 확인하겠습니다.

손코딩

```
print(fruits.shape)
```



```
(300, 100, 100)
```

---

샘플 수, 이미지 높이, 이미지 너비

첫 번째 이미지의 첫 번째 행을 출력하겠습니다. 3차원 배열이기 때문에 처음 2개의 인덱스를 0으로 지정하고 마지막 인덱스는 지정하지 않거나 슬라이싱 연산자를 쓰면 첫 번째 이미지의 첫 번째 행을 모두 선택할 수 있습니다.

손코딩

```
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  1]
```

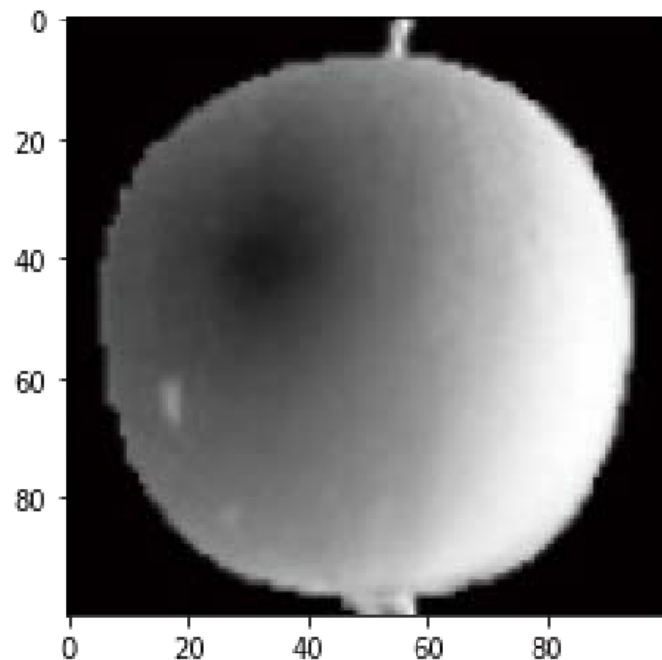
맷플롯립의 imshow() 함수를 사용하면 넘파이 배열로 저장된 이미지를 쉽게 그릴 수 있습니다. 흑백 이미지이므로 cmap 매개변수를 'gray'로 지정합니다.

손코딩

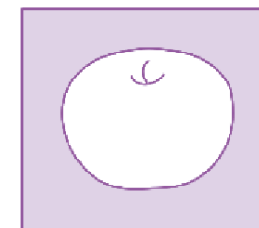
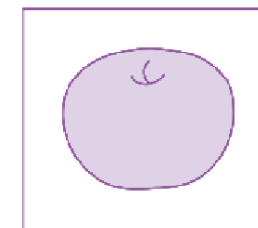
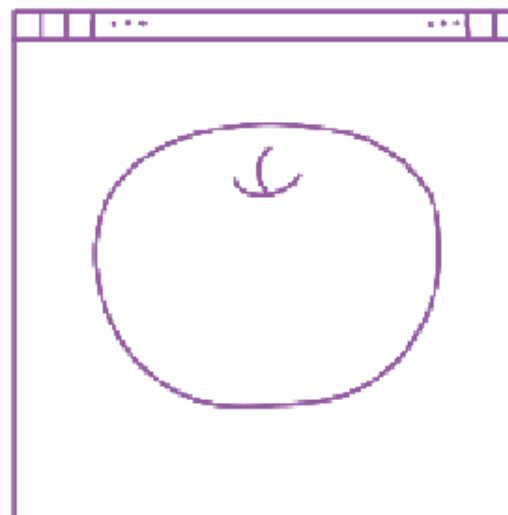
```
plt.imshow(fruits[0], cmap='gray')
plt.show()
```

> 숫자가 0에 가까울수록 검게 표시

> 픽셀값이 0이면 출력도 0이되어 알고리즘의 출력값에 의미를 부여하기가 어려움 -> 이미지 반전



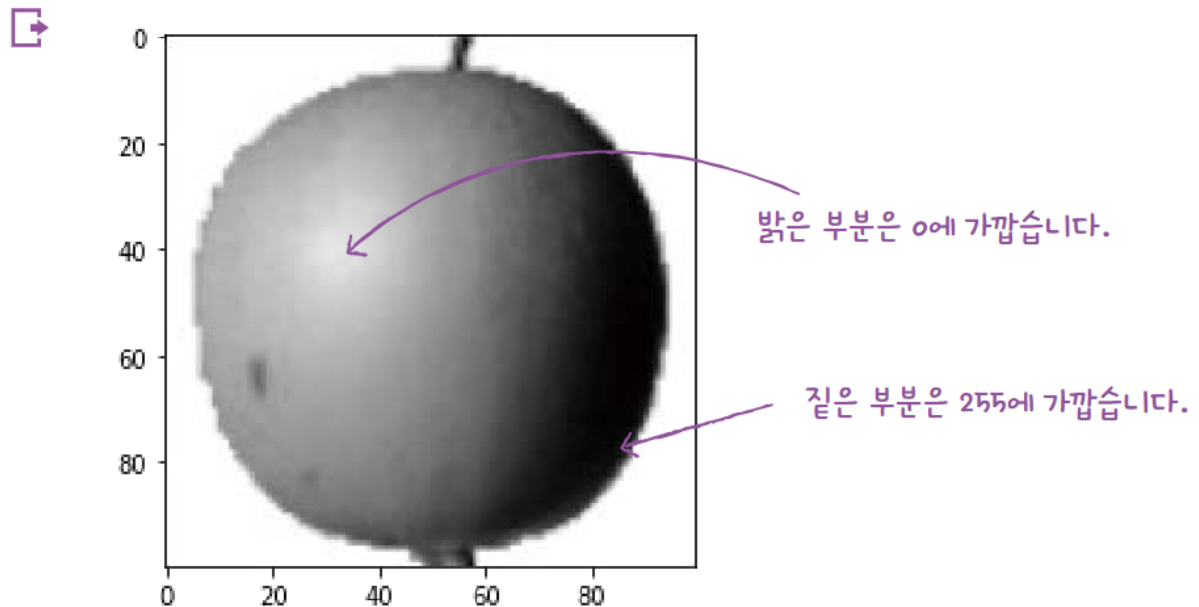
첫 번째 행  
(픽셀 100개)



우리가 보는 것과 컴퓨터가 처리하는 방식이 다르기 때문에 종종 흑백 이미지를 이렇게 반전하여 사용합니다. 관심 대상의 영역을 높은 값으로 바꾸었지만 맷플롯립으로 출력할 때 바탕이 검게 나오므로 보기에는 썩 좋지 않네요. cmap 매개변수를 'gray\_r'로 지정하면 다시 반전하여 우리 눈에 보기 좋게 출력합니다.

손코딩

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

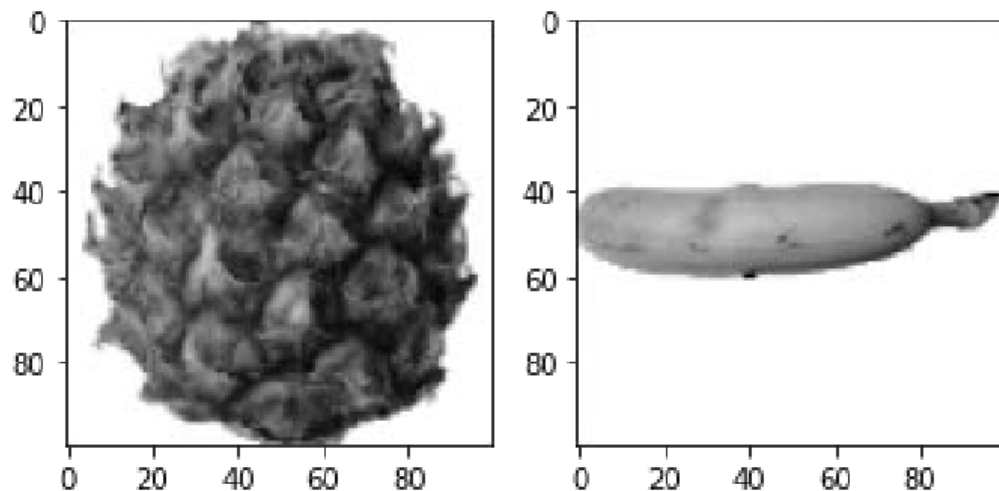


이 데이터는 사과, 바나나, 파인애플이 각각 100개씩 들어 있습니다. 바나나와 파인애플 이미지도 출력하겠습니다.

손코딩

그래프 배열

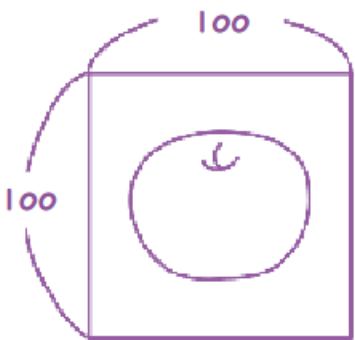
```
fig, axs = plt.subplots(1, 2)
axs[0].imshow(fruits[100], cmap='gray_r')
axs[1].imshow(fruits[200], cmap='gray_r')
plt.show()
```



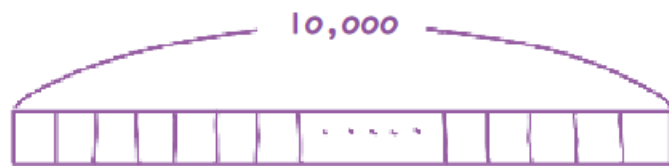


## 픽셀값 분석하기

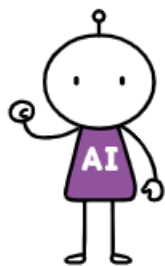
사용하기 쉽게 fruits 데이터를 사과, 파인애플, 바나나로 각각 나누어 봅시다. 넘파이 배열을 나눌 때  $100 \times 100$  이미지를 펼쳐서 길이가 10,000인 1차원 배열로 만들겠습니다. 이렇게 펼치면 이미지로 출력하긴 어렵지만 배열을 계산할 때 편리합니다.



2차원 배열



1차원 배열



fruits 배열에 3개의 과일이 100개씩 있다고 알고 있습니다. 하지만 실전에서는 어떤 과일이 몇 개가 입력될지 알 수 없습니다. 여기에서는 예를 위해서 만든 데이터임을 잊지 마세요.

순서대로 100개씩 선택

샘플 개수  
(남은 차원  
자동 할당)

두 번째, 세 번째 차원을 합침

손코딩

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

손코딩

```
print(apple.shape)
```



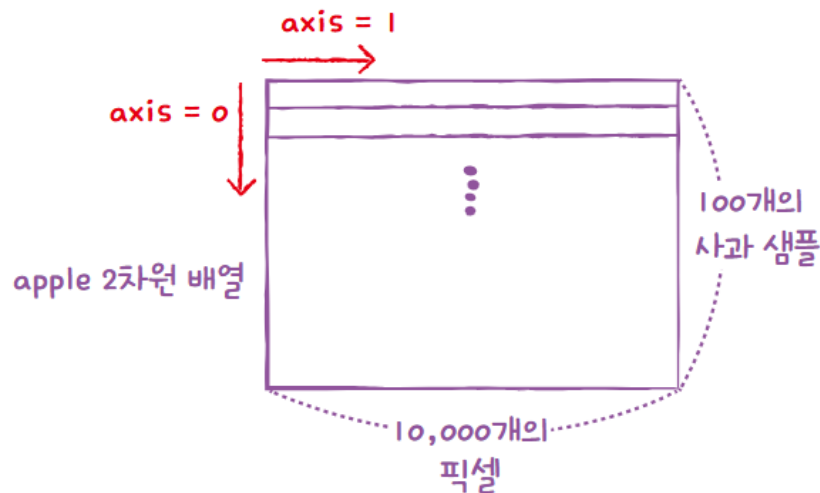
(100, 10000)

우리가 필요한 것은 샘플의 평균값입니다. 샘플은 모두 가로로 값을 나열했으니  $\text{axis}=1$ 로 지정하여 평균을 계산하겠습니다(앞서 2차원 배열을 1차원 배열로 치환했기에 가능한 계산입니다). 평균을 계산하는 넘파이 `np.mean()` 함수를 사용해도 되지만 넘파이 배열은 이런 함수들을 메서드로도 제공합니다. `apple` 배열의 `mean()` 메서드로 각 샘플의 픽셀 평균값을 계산해 보죠.

### + 여기서 잠깐

### axis 인수가 뭔가요?


2장에서도 잠깐 나오긴 했는데 정확히 `axis`에 대해 언급하지는 않았습니다. 먼저 다음 그림을 보세요. 그림의 `axis`는 배열의 '축'을 의미합니다. 다음의 `apple` 2차원 배열에서  $\text{axis}=1$ 일 때는 열 방향으로 계산하고,  $\text{axis}=0$ 일 때는 행 방향으로 계산합니다.



```
print(apple.mean(axis=1))
```

```
[ 88.3346  97.9249  87.3709  98.3703  92.8705  82.6439  94.4244  95.5999
 90.681   81.6226  87.0578  95.0745  93.8416  87.017   97.5078  87.2019
 88.9827 100.9158  92.7823 100.9184 104.9854  88.674   99.5643  97.2495
 94.1179  92.1935  95.1671  93.3322 102.8967  94.6695  90.5285  89.0744
 97.7641  97.2938 100.7564  90.5236 100.2542  85.8452  96.4615  97.1492
 90.711   102.3193  87.1629  89.8751  86.7327  86.3991  95.2865  89.1709
 96.8163  91.6604  96.1065  99.6829  94.9718  87.4812  89.2596  89.5268
 93.799   97.3983  87.151   97.825  103.22   94.4239  83.6657  83.5159
102.8453  87.0379  91.2742 100.4848  93.8388  90.8568  97.4616  97.5022
 82.446   87.1789  96.9206  90.3135  90.565   97.6538  98.0919  93.6252
 87.3867  84.7073  89.1135  86.7646  88.7301  86.643   96.7323  97.2604
 81.9424  87.1687  97.2066  83.4712  95.9781  91.8096  98.4086 100.7823
101.556  100.7027  91.6098  88.8976]
```

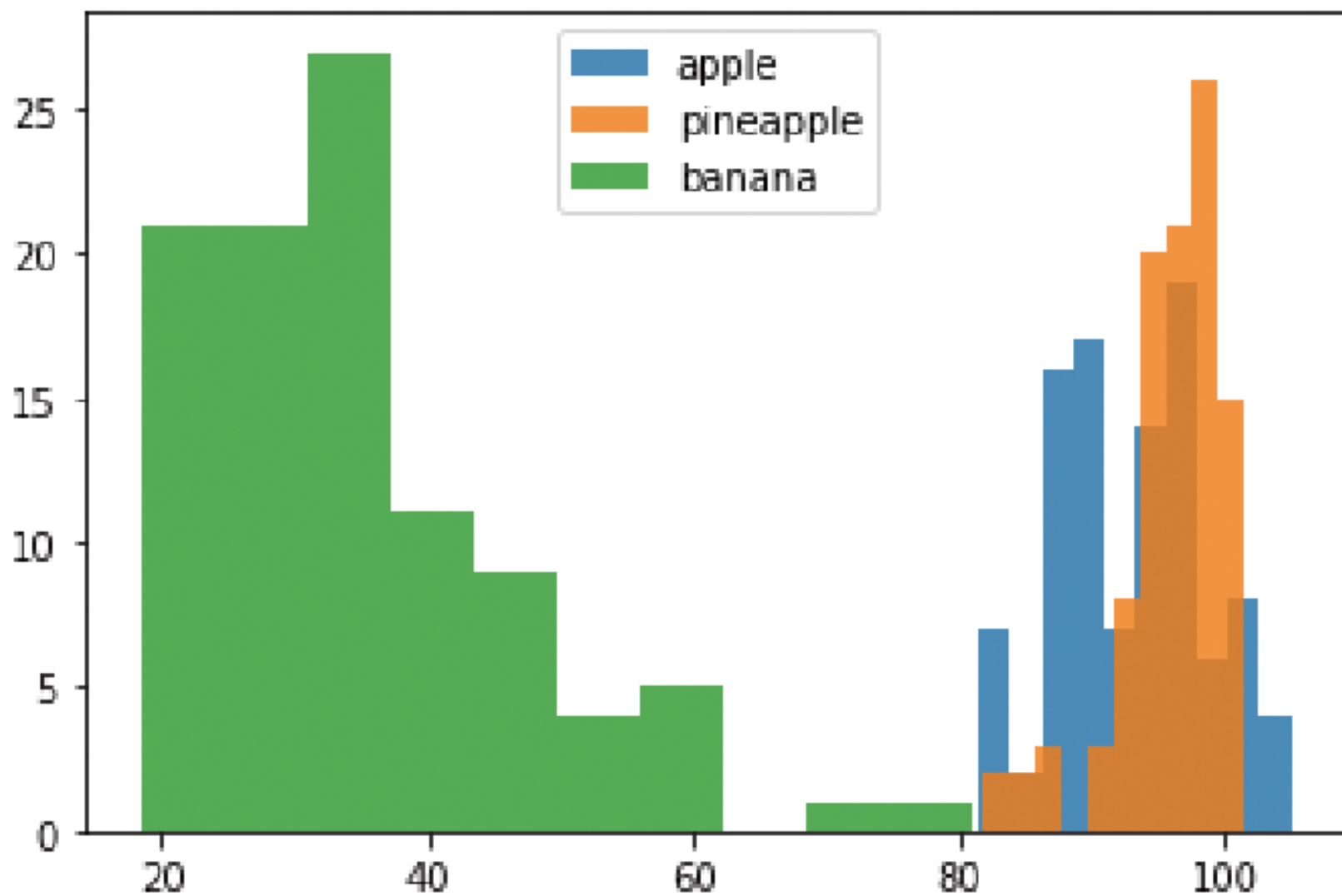
맷플롯립의 `hist()` 함수를 사용해 히스토그램을 그려 보죠. 사과, 파인애플, 바나나에 대한 히스토그램을 모두 겹쳐 그려 보겠습니다. 이렇게 하려면 조금 투명하게 해야 겹친 부분을 잘 볼 수 있습니다. `alpha` 매개변수를 1보다 작게 하면 투명도를 줄 수 있습니다. 또 맷플롯립의 `legend()` 함수를 사용해 어떤 과일의 히스토그램인지 범례를 만들어 보죠.

손코딩

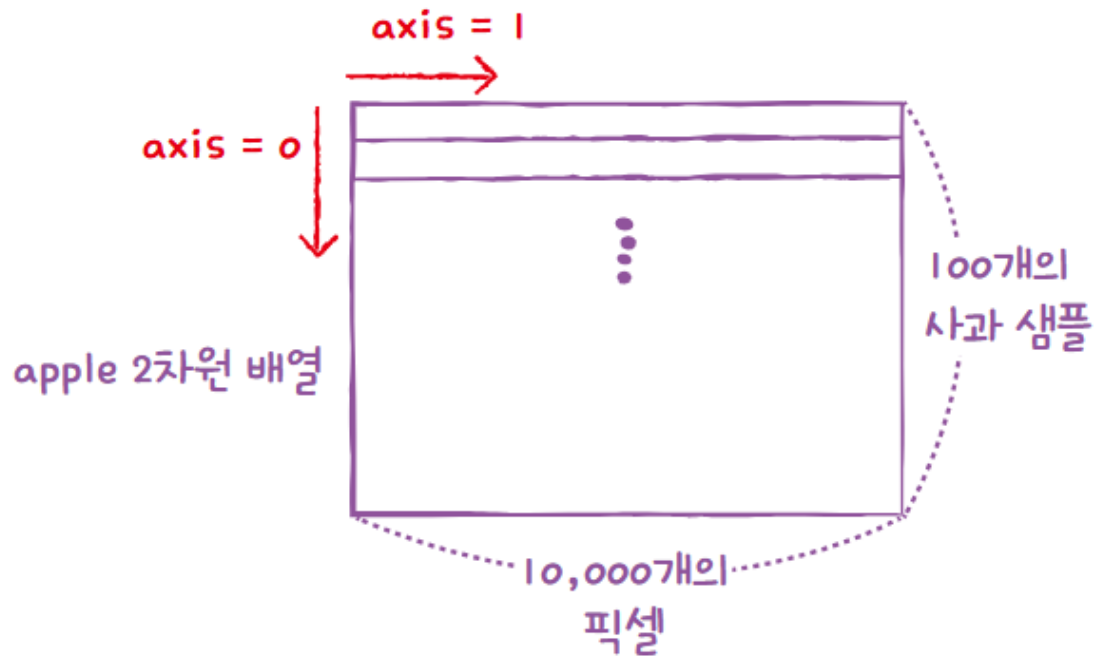
```
plt.hist(np.mean(apple, axis=1), alpha=0.8)
plt.hist(np.mean(pineapple, axis=1), alpha=0.8)
plt.hist(np.mean(banana, axis=1), alpha=0.8)

plt.legend(['apple', 'pineapple', 'banana'])
plt.show()
```



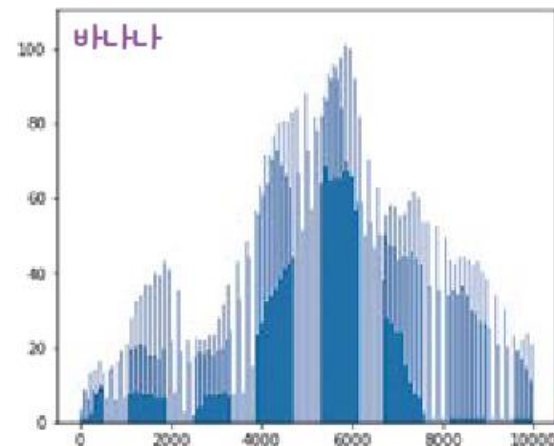
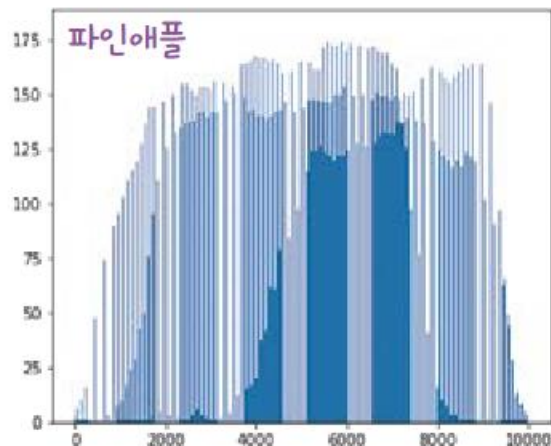
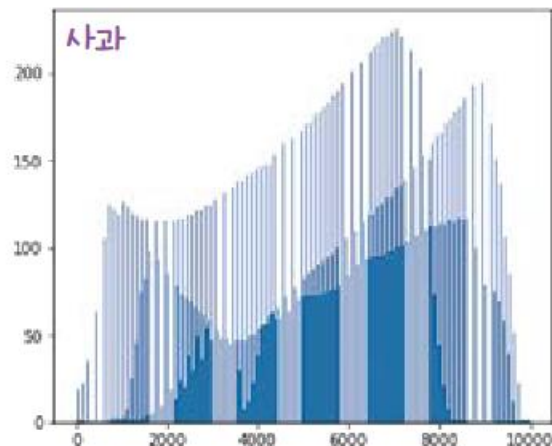


좀 더 나은 방법은 없을까요? 혼공머신은 골똥히 생각하다가 샘플의 평균값이 아니라 픽셀별 평균값을 비교해 보면 어떨까 생각했습니다. 전체 샘플에 대해 각 픽셀의 평균을 계산하는 거죠. 세 과일은 모양이 다르므로 픽셀값이 높은 위치가 조금 다를 것 같습니다.



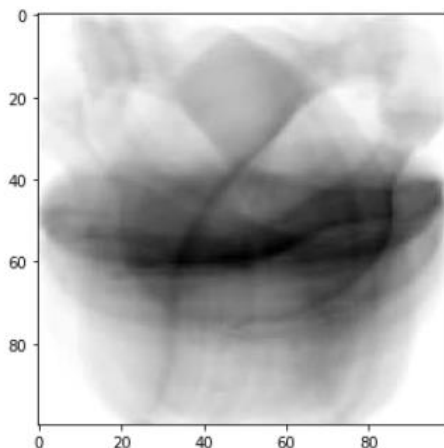
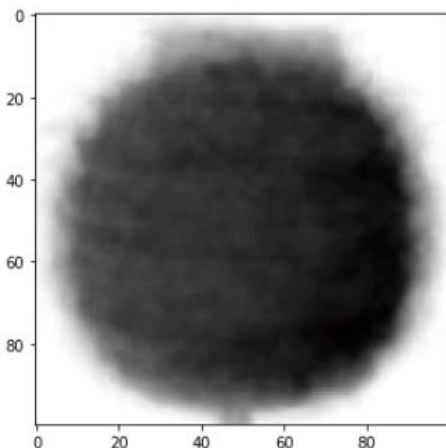
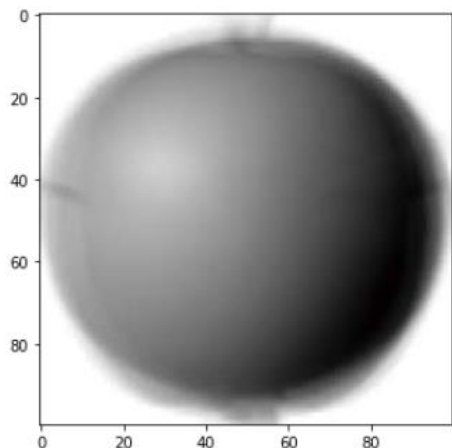
손코딩

```
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()
```



손코딩

```
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()
```



## 평균값과 가까운 사진 고르기

사과 사진의 평균값인 `apple_mean`과 가장 가까운 사진을 골라보죠. 3장에서 봤던 절댓값 오차를 사용하겠습니다. `fruits` 배열에 있는 모든 샘플에서 `apple_mean`을 뺀 절댓값의 평균을 계산하면 됩니다.

**손코딩**

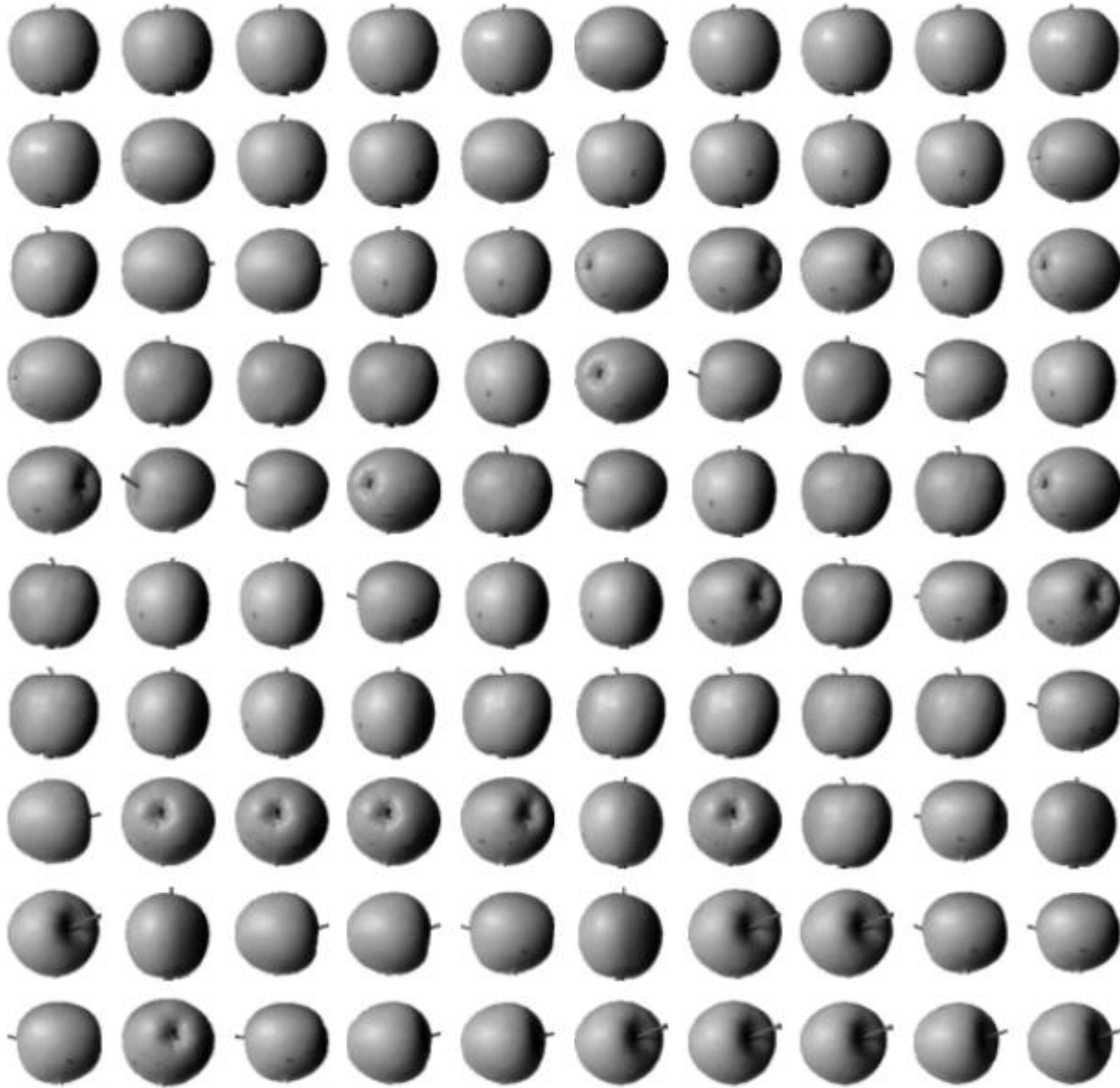
```
abs_diff = np.abs(fruits - apple_mean) #절댓값 계산
abs_mean = np.mean(abs_diff, axis=(1,2)) # Axis에 두 번째, 세 번째
print(abs_mean.shape)                  차원을 저장
```

➡ (300,)



손코딩

```
apple_index = np.argsort(abs_mean)[:100] #가장 작은 순서대로 100개
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()
```



흑백 사진에 있는 픽셀값을 사용해 과일 사진을 모으는 작업을 해 보았습니다. 이렇게 비슷한 샘플끼리 그룹으로 모으는 작업을 **군집**clustering이라고 합니다. 군집은 대표적인 비지도 학습 작업 중 하나입니다. 군집 알고리즘에서 만든 그룹을 **클러스터**cluster라고 부릅니다.

하지만 우리는 이미 사과, 파인애플, 바나나가 있다는 것을 알고 있었습니다. 즉 타깃값을 알고 있었기 때문에 사과, 파인애플, 바나나의 사진 평균값을 계산해서 가장 가까운 과일을 찾을 수 있었습니다. 실제 비지도 학습에서는 타깃값을 모르기 때문에 이처럼 샘플의 평균값을 미리 구할 수 없습니다.

타깃값을 모르면서 어떻게 세 과일의 평균값을 찾을 수 있을까요? 다음 2절에서 배울 k-평균 알고리즘이 이 문제를 해결해 줍니다.

# 06-2

## k-평균

핵심 키워드

k-평균

클러스터 중심

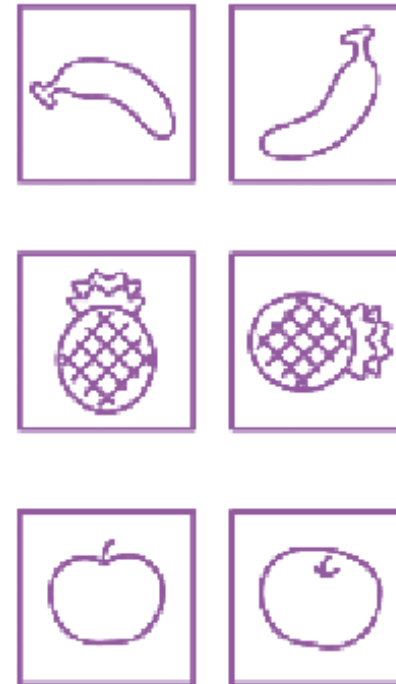
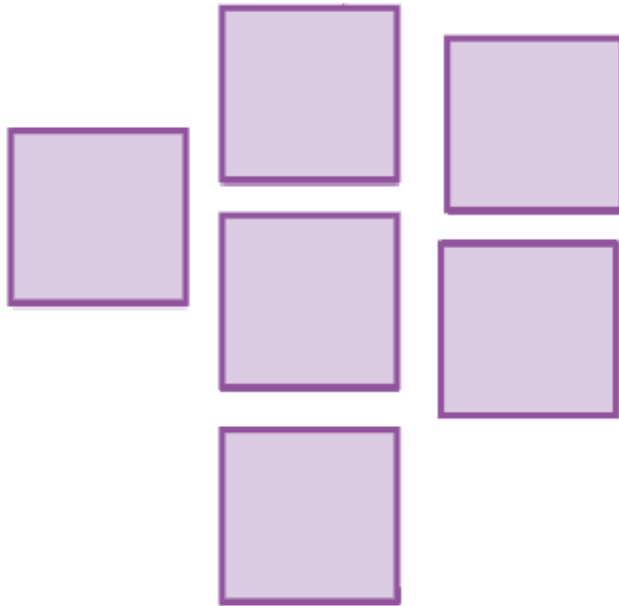
엘보우 방법

k-평균 알고리즘의 작동 방식을 이해하고 과일 사진을 자동으로 모으는 비지도 학습 모델을 만들어 봅니다.

대체 이게  
무슨 과일이지?



과일 매니아



← 이렇게 과일 사진을  
자동으로 모을  
수 있을까요?



## 1. 정의 (Definition)

- K-평균 알고리즘은 비지도 학습(unsupervised learning)의 대표적인 클러스터링(clustering) 기법
- 주어진 데이터셋을 사용자가 지정한 K개의 그룹(클러스터)으로 자동 분류
- 각 그룹은 중심점(centroid)를 기준으로 형성되며, 각 데이터 포인트는 가장 가까운 중심점에 할당

**활용 예시:** 고객 세분화, 이미지 압축, 이상치 탐지, 문서 군집화

## 2. 작동 원리 (How K-Means Works)

1. 중심점 초기화: 데이터에서 무작위로 K개의 중심점을 선택
2. 할당 단계 (Assignment step): 각 데이터를 가장 가까운 중심점에 할당 (유클리드 거리 사용)
3. 업데이트 단계 (Update step): 각 클러스터의 중심점을, 해당 클러스터에 속한 모든 데이터의 평균으로 재계산
4. 수렴 조건 확인: 중심점 변화가 거의 없거나, 정해진 반복 횟수에 도달하면 종료
5. 최종 결과: K개의 클러스터와 중심점

### 3. 수학적 정의

- 데이터 포인트 집합:  $X = \{x_1, x_2, \dots, x_n\}$
- 목표: 각  $x_i$ 를 클러스터  $C_k$ 에 할당하여 **클러스터 내 제곱 거리 합을 최소화**

$$\min \sum_{k=1}^K \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

- $\mu_k$ : 클러스터  $C_k$ 의 중심점 (centroid)

#### 4. 한계 및 주의점

- K를 사전에 지정해야 함 → 적절한 K 선택이 중요 (Elbow method 활용)
- 이상치(outlier)에 민감
- 원형 구조나 밀도 차이가 있는 데이터에는 부적합

**\*문제점 1: 클러스터가 원형이 아닐 때**

$$\text{distance}(A, B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- K-평균은 유클리드 거리(Euclidean distance)를 기준으로 데이터를 분할하고, 클러스터의 중심점에서 가까운 데이터를 할당.  
\*두 점 사이의 직선 (가장 짧은) 거리
- 이 때문에, 각 클러스터는 구형(spherical) 또는 원형(circular) 구조라고 가정하는 셈.

**잘 작동하지 않는 예**

- 데이터가 달 모양(Moon-shaped)이나 나선형(Spiral)으로 분포되어 있는 경우
- 클러스터의 경계가 복잡하거나 구불구불한 경우

대안: DBSCAN, Spectral Clustering 등 비구형 군집에 강한 알고리즘 사용

### \*문제점 2: 클러스터의 밀도 차이가 있을 때

- K-평균은 각 클러스터가 비슷한 크기와 분산(=데이터 밀도)을 가지고 있다고 가정
- 하지만 현실 데이터는 하나의 클러스터가 조밀하고, 다른 하나는 넓게 퍼져 있을 수 있음

### 잘 작동하지 않는 예

- 하나의 클러스터는 좁고 밀집돼 있고, 다른 클러스터는 넓고 희박한 경우
- 그 결과, K-평균이 작은 클러스터를 여러 개로 쪼개거나, 넓은 클러스터를 제대로 인식하지 못함

대안: Gaussian Mixture Model (GMM): 서로 다른 분산을 가진 클러스터를 가정할 수 있음, Density-Based 방법 (예: DBSCAN)

**Advanced**

$$\text{distance}(A, B) = |x_2 - x_1| + |y_2 - y_1|$$

**\*맨하탄 거리(Manhattan Distance)란?**

- 격자 형태의 도시(예: 뉴욕 맨해튼)에서 길을 따라 움직이는 거리에서 유래
- 직선 이동이 불가능하고, 가로와 세로로만 이동할 수 있을 때 사용하는 거리 개념
- 맨하탄 거리는 항상 유클리드 거리보다 크거나 같다

**언제 사용?**

- 데이터가 축을 따라 정렬되어 있을 때 (예: 체스판, 도시 도로)
- 모델이 축 기준으로 더 민감할 때 (예: LASSO 회귀)
- 로봇 경로 탐색, 강화학습 환경 등에서 경로가 grid 형태일 때



## Advanced

모델이 축 기준으로 더 민감하다는게 무슨 말?

-> 모델이 특성(feature) 하나하나의 값 변화에 민감하게 반응하고, 축을 따라 모델의 가중치를 조정하는 경향이 있다

### 배경 정리: L1 정규화와 LASSO

$$\min_{\beta} \left( \text{Loss} + \lambda \sum |\beta_j| \right)$$

- L1 정규화는 가중치 벡터의 절댓값 합(Manhattan Distance)을 최소화하려는 방식.
- LASSO (Least Absolute Shrinkage and Selection Operator)는 L1 정규화를 사용하는 선형 회귀.
- L1 정규화는 가중치 중 일부를 0으로 만들어서 변수 선택(Feature Selection) 효과.

# Types of Regularization Techniques (6주차 수업)

L1 penalty =  $\|\beta\|_1 = |\beta_1| + |\beta_2|$  맨하탄 거리

L1 constraint defines a region:  $|\beta_1| + |\beta_2| \leq t$

$$\beta_1 + \beta_2 = t$$

$$\beta_1 - \beta_2 = t$$

$$-\beta_1 + \beta_2 = t$$

$$-\beta_1 - \beta_2 = t$$

The corners of this diamond lie on the axes (where either  $\beta_1$  or  $\beta_2 = 0$ ).

Where  $t$  represents constraint  
budget — max allowed "size" of  $\beta$

L2 penalty =  $\|\beta\|_2^2 = \beta_1^2 + \beta_2^2$  유클리드 거리

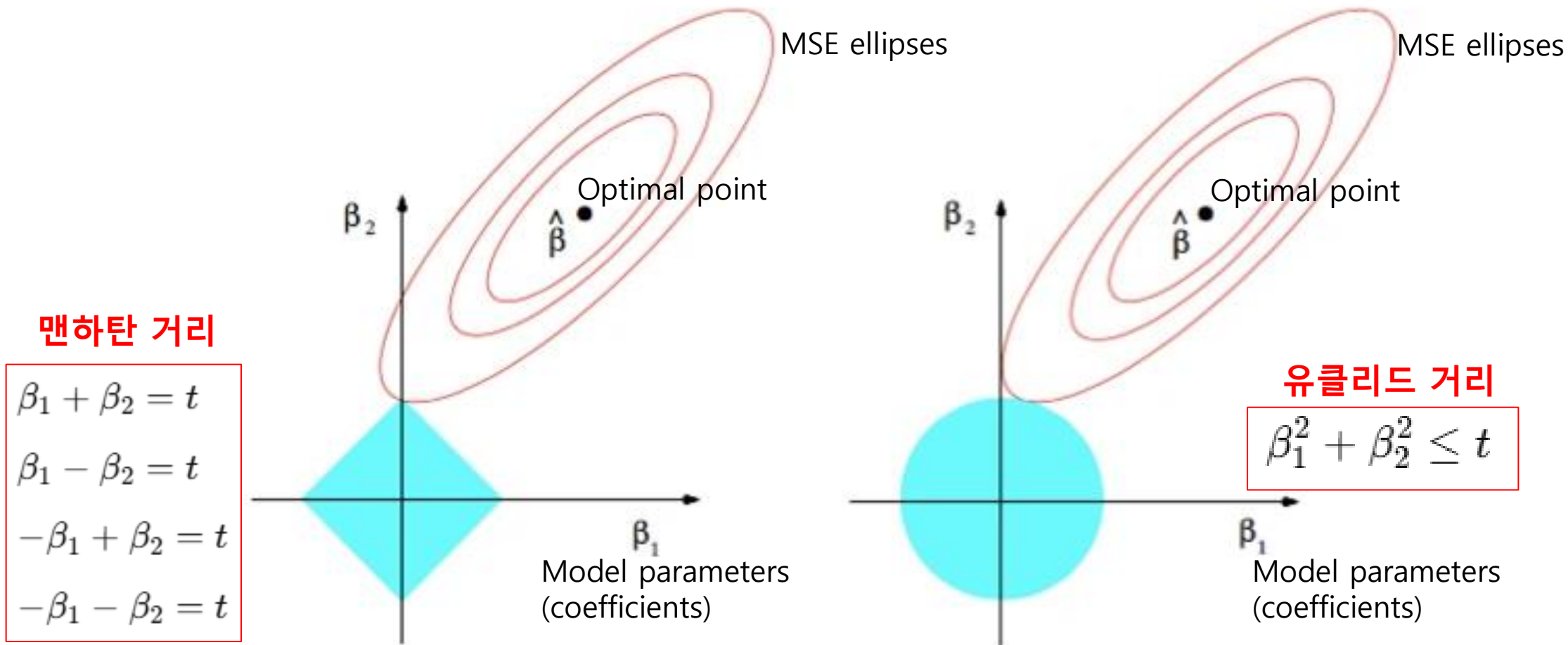
L2 constraint defines a region:  $\beta_1^2 + \beta_2^2 \leq t$

Where  $t$  represents constraint  
budget — max allowed "size" of  $\beta$

This is the equation of a circle (or a sphere in higher dimensions), centered at the origin

# Types of Regularization Techniques (6주차 수업)

This is visually equivalent to finding the first point where the MSE contour touches (is tangent to) the constraint region (blue shape).



교차점이 생기는 꼭짓점 덕분에, L1 제약은 축을 따라 정답이 나올 확률이 높아짐 → 즉, 가중치 중 일부가 정확히 0이 될 가능성 높음

## Advanced

모델이 축 기준으로 더 민감하다는게 그래서 무슨 말?

- L1 정규화/LASSO는 맨하탄 거리 기반으로, 특정 축(특성)에 맞춰 가중치를 줄이기 때문에 0으로 만들기 쉬움
- 이 때문에, 모델이 특정 feature 축을 따라 움직이며 민감하게 반응한다고 말함

# 유클리드 거리 vs 맨하탄 거리

항목	유클리드 거리	맨하탄 거리
별명	직선 거리	택시 거리
계산 방식	제곱합의 제곱근	절댓값의 합
이동 방향	모든 방향 가능	축을 따라 이동만 가능
사용 예시	K-평균, 이미지 압축	도시 길, 일부 딥러닝 모델

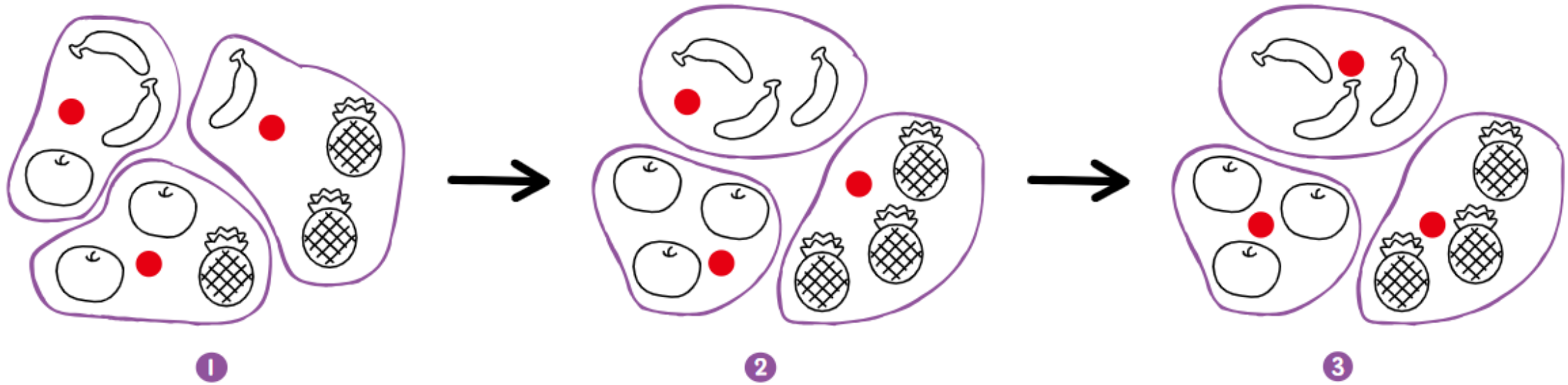
이런 경우 어떻게 평균값을 구할 수 있을까요? 바로 **k-평균**<sup>k-means</sup> 군집 알고리즘이 평균값을 자동으로 찾아줍니다. 이 평균값이 클러스터의 중심에 위치하기 때문에 **클러스터 중심**<sup>cluster center</sup> 또는 **센트로이드**<sup>centroid</sup>라고 부릅니다.

## k-평균 알고리즘 소개

k-평균 알고리즘의 작동 방식은 다음과 같습니다.


- 1 무작위로 k개의 클러스터 중심을 정합니다.
- 2 각 샘플에서 가장 가까운 클러스터 중심을 찾아 해당 클러스터의 샘플로 지정합니다.
- 3 클러스터에 속한 샘플의 평균값으로 클러스터 중심을 변경합니다.
- 4 클러스터 중심에 변화가 없을 때까지 2번으로 돌아가 반복합니다.

이를 그림으로 나타내면 다음과 같습니다.




## KMeans 클래스

1절에서 사용했던 데이터셋을 여기에서도 사용하겠습니다. 먼저 wget 명령으로 데이터를 다운로드 합니다.


손코딩

```
!wget https://bit.ly/fruits_300_data -O fruits_300.npy
```

손코딩

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



손코딩

```
from sklearn.cluster import KMeans  
km = KMeans(n_clusters=3, random_state=42)  
km.fit(fruits_2d)
```

군집된 결과는 KMeans 클래스 객체의 labels\_ 속성에 저장됩니다. labels\_ 배열의 길이는 샘플 개수와 같습니다. 이 배열은 각 샘플이 어떤 레이블에 해당되는지 나타냅니다. n\_clusters=3으로 지정했기 때문에 labels\_ 배열의 값은 0, 1, 2 중 하나입니다.

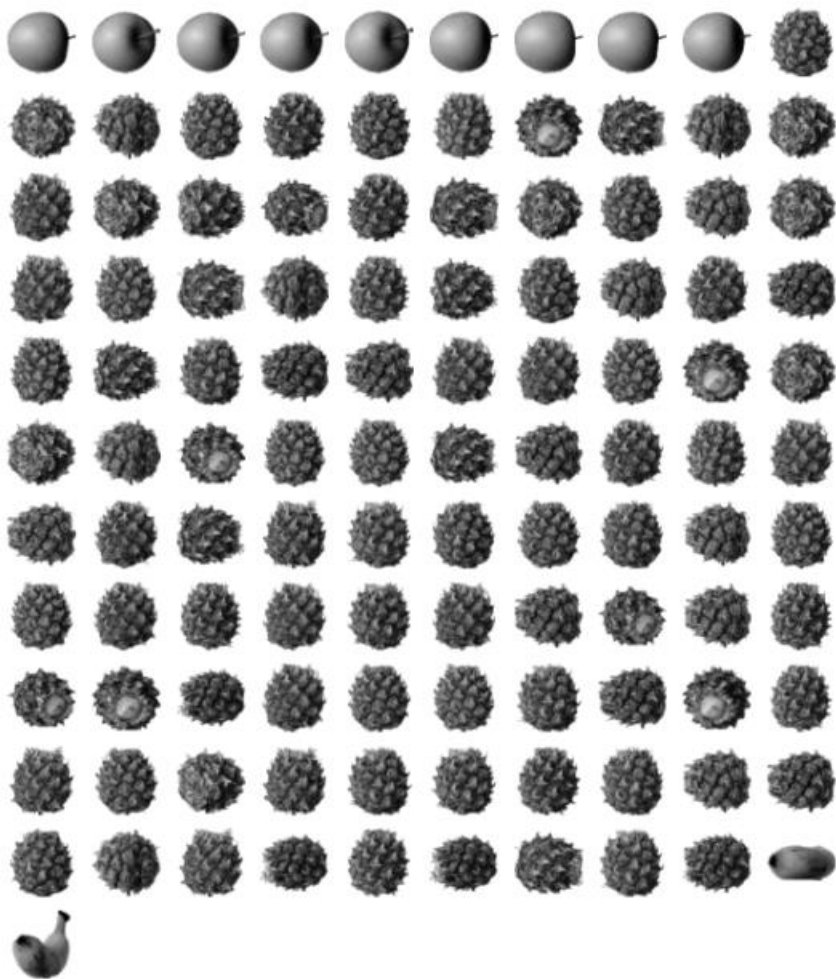


손코딩

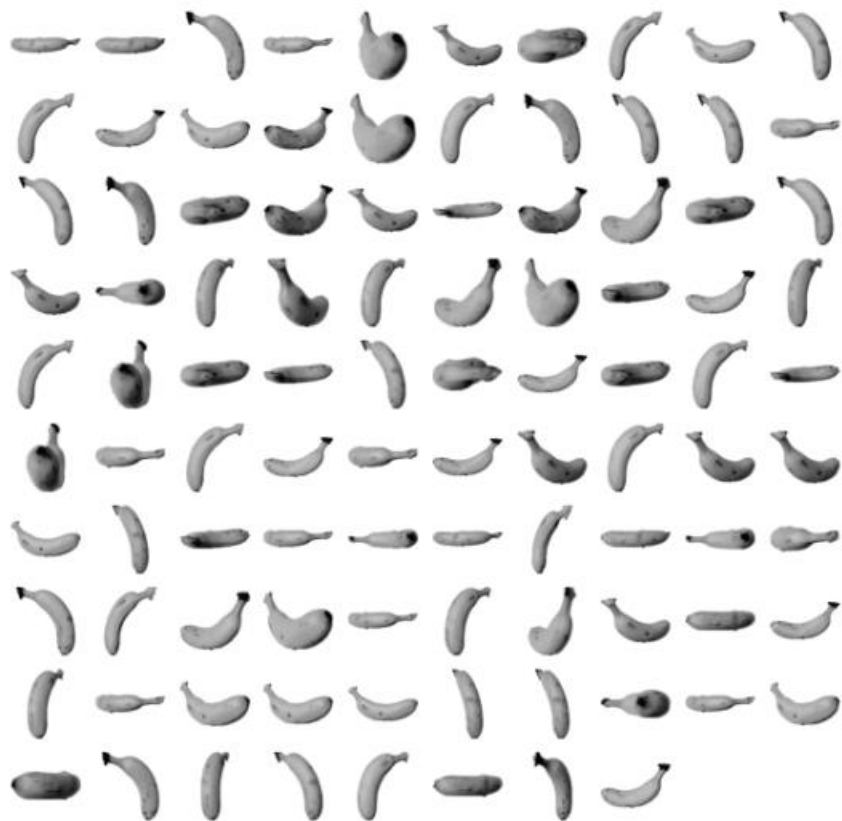
```
import matplotlib.pyplot as plt

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 1개이면 열의 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])
```



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





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

06-2

k-평균

