

# **비지도학습:** **Clustering, K-means, PCA**

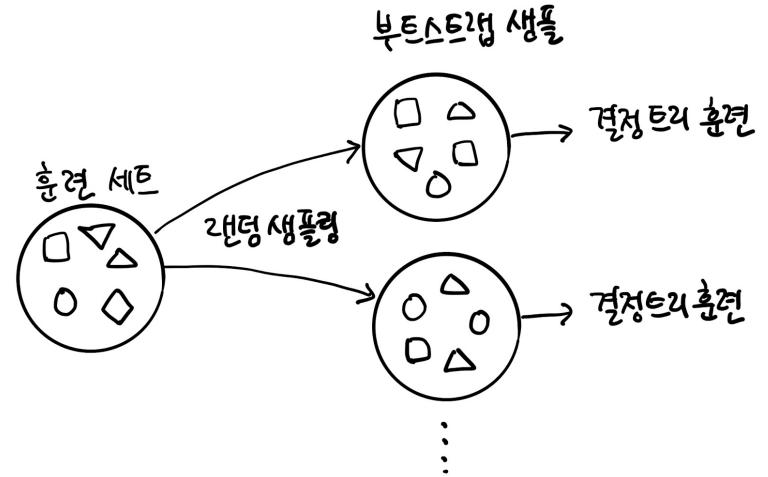
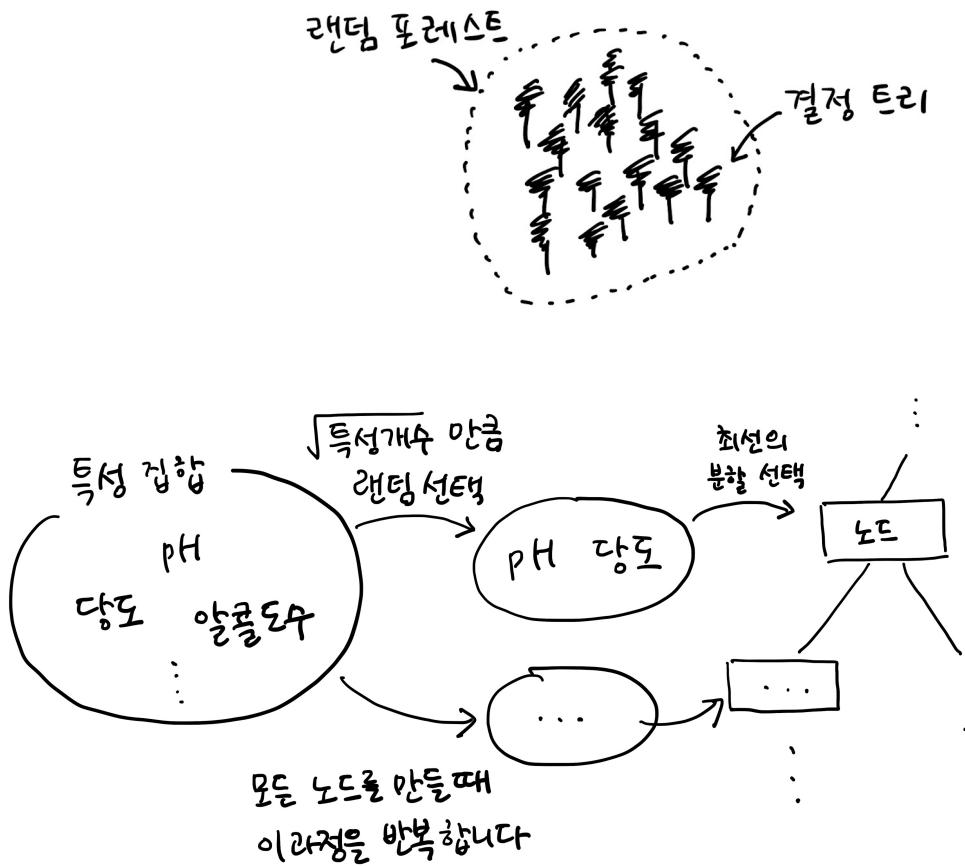
임 경 태

# CONTENTS

---

- 1      복습**
- 2      군집과 대푯값의 개념**
- 3      K-means 군집 알고리즘**
- 4      주성분 분석**

## Random Forest 복습



- 엑스트라 트리: 결정 트리의 노드를 랜덤하게 분할함.
- 그레이디언트 부스팅: 이전 트리의 손실을 보완하는 시스템으로  
이은 결정트리를 연속하여 추가함.
- 하이퍼파라미터 기반 그레이디언트 부스팅: 훈련 데이터를  
256개 정수 구간으로 나누어 빠르고 높은 성능을 냄.

# CONTENTS

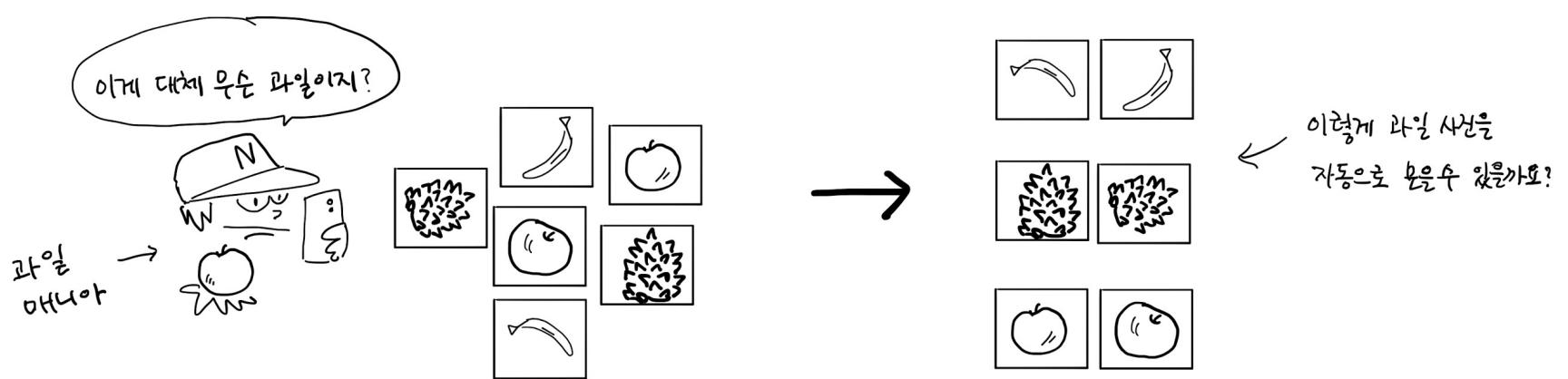
---

- 1 복습
- 2 군집과 대푯값의 개념
- 3 K-means 군집 알고리즘
- 4 주성분 분석

# 새로운 문제!

## ❶ 과일 분류하기

- 입력: 과일 사진 여러개
- Target: 없음..
- 출력: 종류별로 분류된 과일사진
- 모델?:



어라 label 정보가 없네??



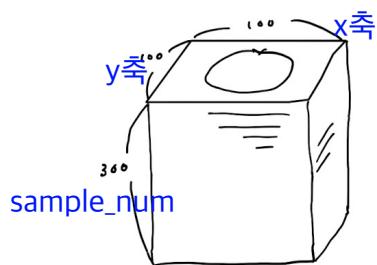
## 군집 (Clustering)

- 정의: 비지도학습 방법 중 하나로 개체를 분류하기 위한 명확한 분류기준이 존재하지 않을 때 주어진 데이터의 특성 정보를 이용해 같은 그룹을 정의하고 서로 유사한 개체가 되도록 대표성을 찾는 방법  
간단히 말해 Label (target) 정보가 없을 때 Label을 대략적으로 찾는 방법
- 어떻게 하면 주어진 데이터의 특성 정보를 이용해 같은 그룹(label)을 정의 할 수 있을까?
  - 데이터 특징의 대푯값 (평균, 분산, 표준편차, 분포 등) 을 활용해서 분류 하자!

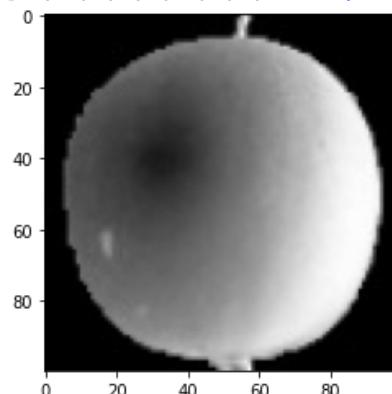
## 데이터 특성파악

## 데이터 구조 살피기

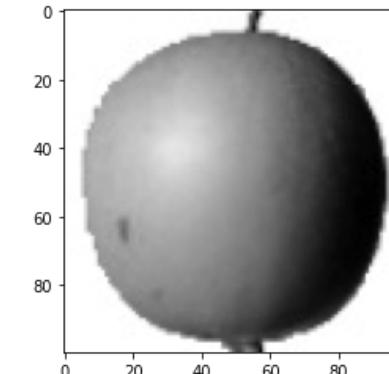
- sample\_num이 0번째인 사진의 x축 데이터를 확인해보자
    - 사진은 기본적으로 0~255의 정수값으로 이루어져있다. 높은 값은 밝은색, 낮은 값은 어두운색 표현



```
plt.imshow(fruits[0], cmap='gray')
plt.show()
레이터의 이미지를 출력 cmap: color map
```



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

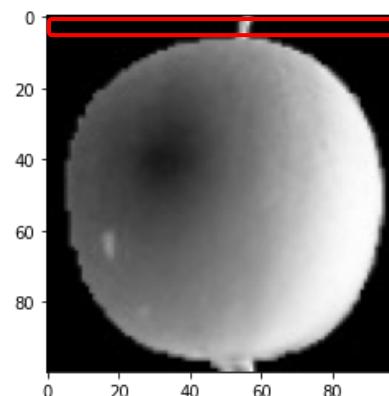


## 데이터 구조 살피기

- 바탕화면이 검정색인 이유?

- 바탕이 검정색 (255에 가까운)이면 모델이 학습할 때 과일보다 바탕에 영향을 많이 받음. 왜냐하면 값이 크니까!

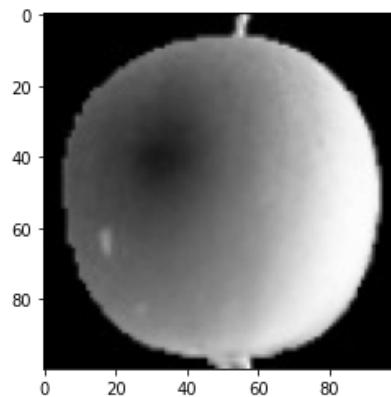
Tip:  $Wx + b$  를 생각해보자 픽셀값이 0이면 출력도 0이됨



```
print(fruits[0, 0, :])  
[ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  
 2  2  2  2  2  2  1  1  1  1  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  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  1 ]  
  
plt.imshow(fruits[0], cmap='gray')  
plt.show()
```

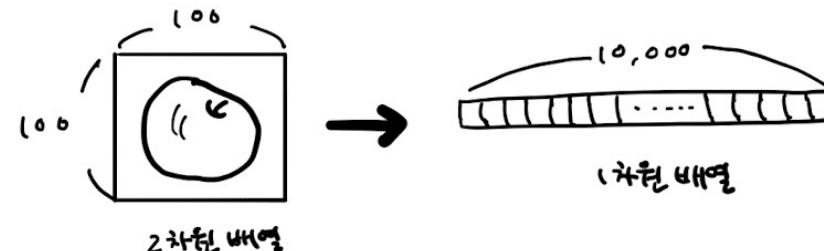
## 데이터 구조 변경하기

- 2차원 배열로 구성된 샘플을 1차원으로 변경하자
  - 시각화 하기엔 2차원 배열이 좋지만 데이터의 특성을 파악하기엔 1차원 vector가 유리하다.
  - 이때 실험을 위해 사과, 파인애플, 바나나를 각각 따로 저장해보자



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

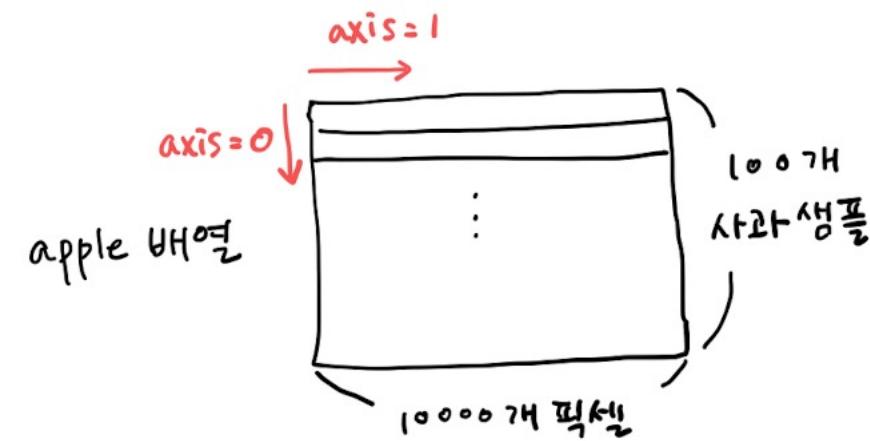
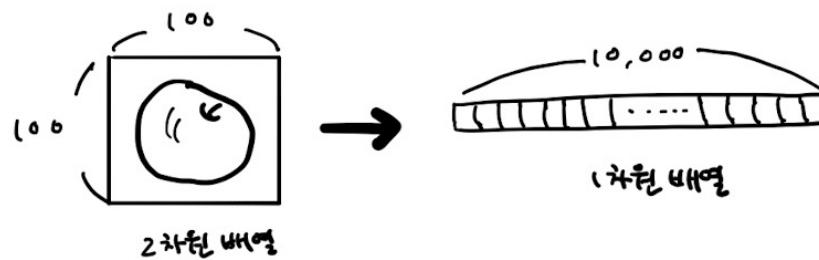
print(apple.shape)
(100, 10000)
```



# 데이터 특성파악

## 데이터 대푯값 파악하기

- 2차원 배열로 구성된 샘플을 1차원으로 변경하자
  - 시각화 하기엔 2차원 배열이 좋지만 데이터의 특성을 파악하기엔 1차원 vector가 유리하다

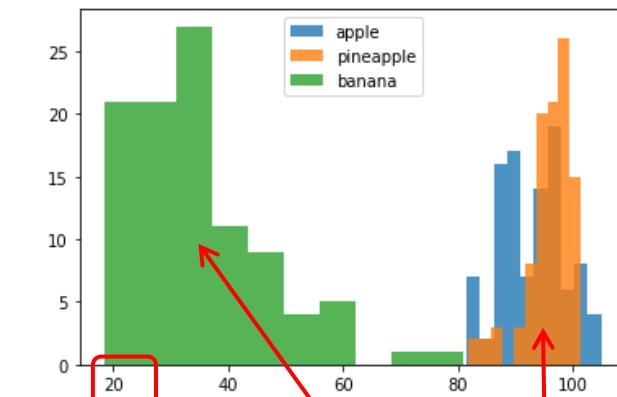
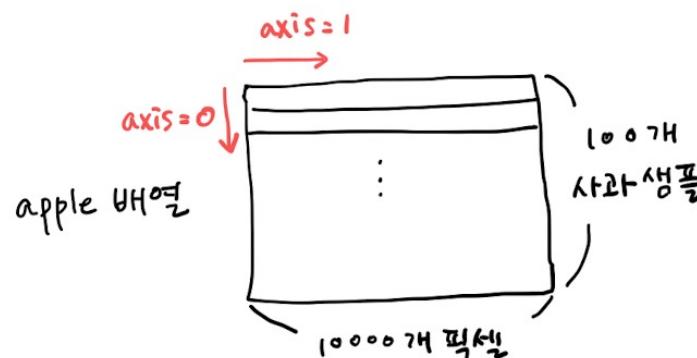


사과 사진을 다른 과일과 구분 짓는 **대푯값**은 어떻게 찾을까?  
- 사과는 동그랗고 바나나는 길지... 그러면 vector값의 분포도 다르겠네?

## 데이터 대푯값 파악하기

- 대푯값(평균, 표준편차, 중간값, 분산 등)을 시각화 해보자
  - 각 샘플을 대표할 수 있는 값은 평균이므로 **샘플을 평균값으로** 변환해보자
  - 변환된 평균값을 histogram을 이용해 각 과일별 평균의 빈도를 분석해보자

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



만개의 값의 평균값이  
20~30사이인 샘플의 수

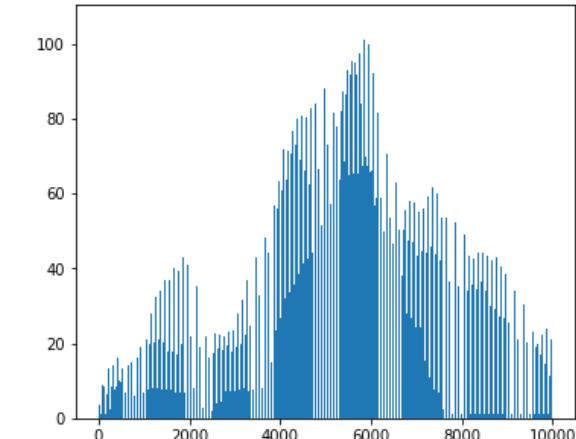
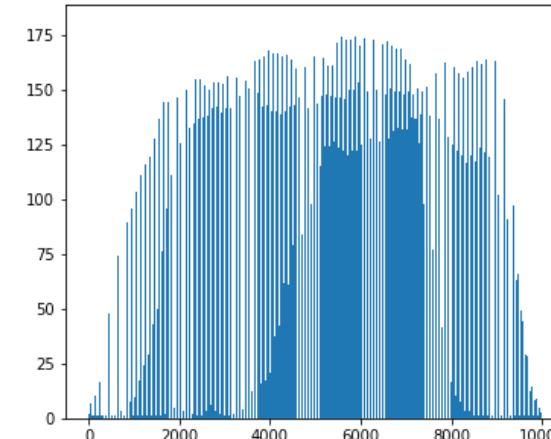
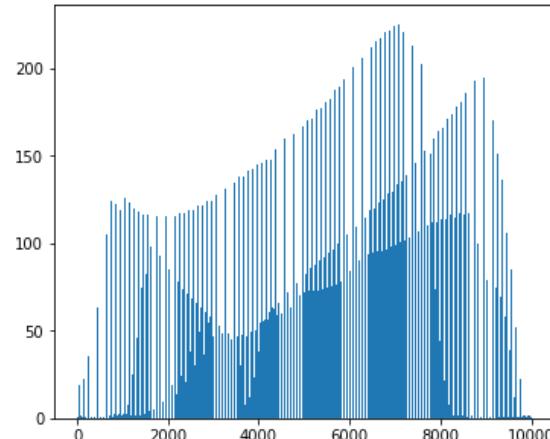
바나나는 사진에서 차지하는 영역이 작으니 평균값이 작네?  
사과와 파인애플은 평균값으로 구분하기 어렵다 어떡하지?



## 데이터 대푯값 파악하기

- 대푯값(평균, 표준편차, 중간값, 분산 등)을 시각화 해보자
  - 세 과일은 모양이 다르니 픽셀값이 높은 위치가 다르지 않을까? **픽셀의 평균값으로** 변환해보자
  - 변환된 평균값을 histogram을 이용해 각 과일별 평균의 빈도를 분석해보자

```
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()
```

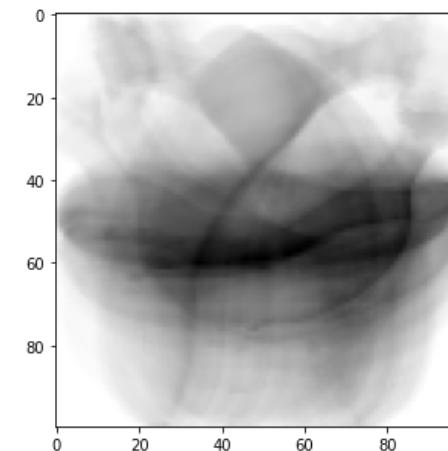
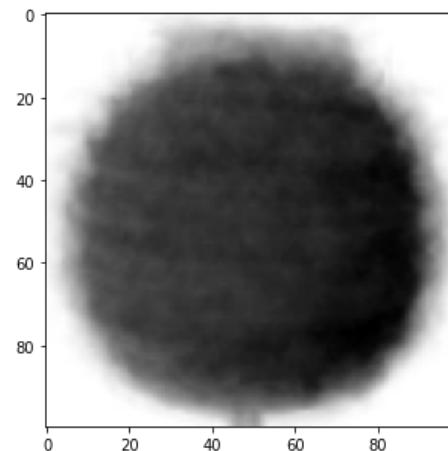
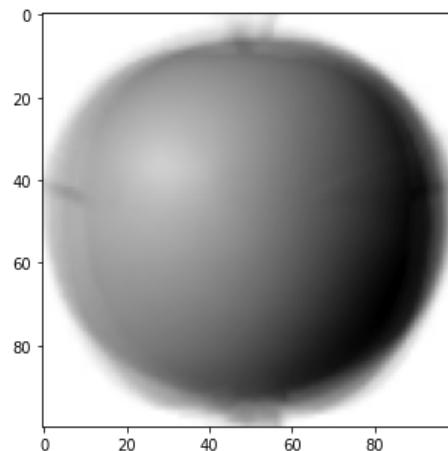


오! 세 과일의 특성값의 **분포**가 매우 다르구나 **분포**를 보고 분류하면 되겠다!

## 데이터 대푯값 시각화하기

- 세 과일은 모양이 픽셀 위치에 따라 값의 차이가 난다. 그렇다면 각 과일의 픽셀 평균으로 만든 이미지와

새로 입력 받는 파일을 비교해서 분류하면 되겠네?



```
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)

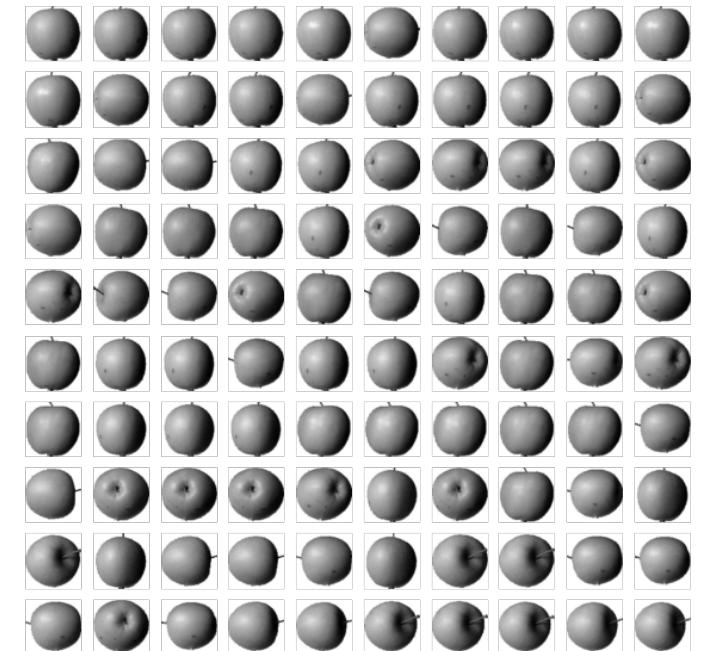
fig, axs = plt.subplots(1, 3, figsize=(20, 5))
axs[0].imshow(apple_mean, cmap='gray_r')
axs[1].imshow(pineapple_mean, cmap='gray_r')
axs[2].imshow(banana_mean, cmap='gray_r')
plt.show()
```

## 데이터 대푯값과 유사한 샘플 고르기

- 사과, 파인애플, 바나나가 포함된 전체 사진에서 사과의 대푯값(픽셀 평균)과 가장 유사한 샘플을 어떻게 고를 수 있을까?
  - 사과 대푯값과 각 샘플의 픽셀값의 오차를 이용해서 오차가 작은 샘플을 찾으면 된다.

```
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] 값이 작은것부터 큰것으로 sorting하고 index를 반환
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()
```



전체 300장 사진 중 사과 대푯값과 유사한 샘플 100개를 뽑았는데 모두 사과가 나왔다 == 분류 정확도 100% !!  
그런데 뭔가 이상한데? 우리는 데이터가 사과, 파인애플, 바나나 사진이고 각 파일의 인덱스를 이미 알고 있었음...

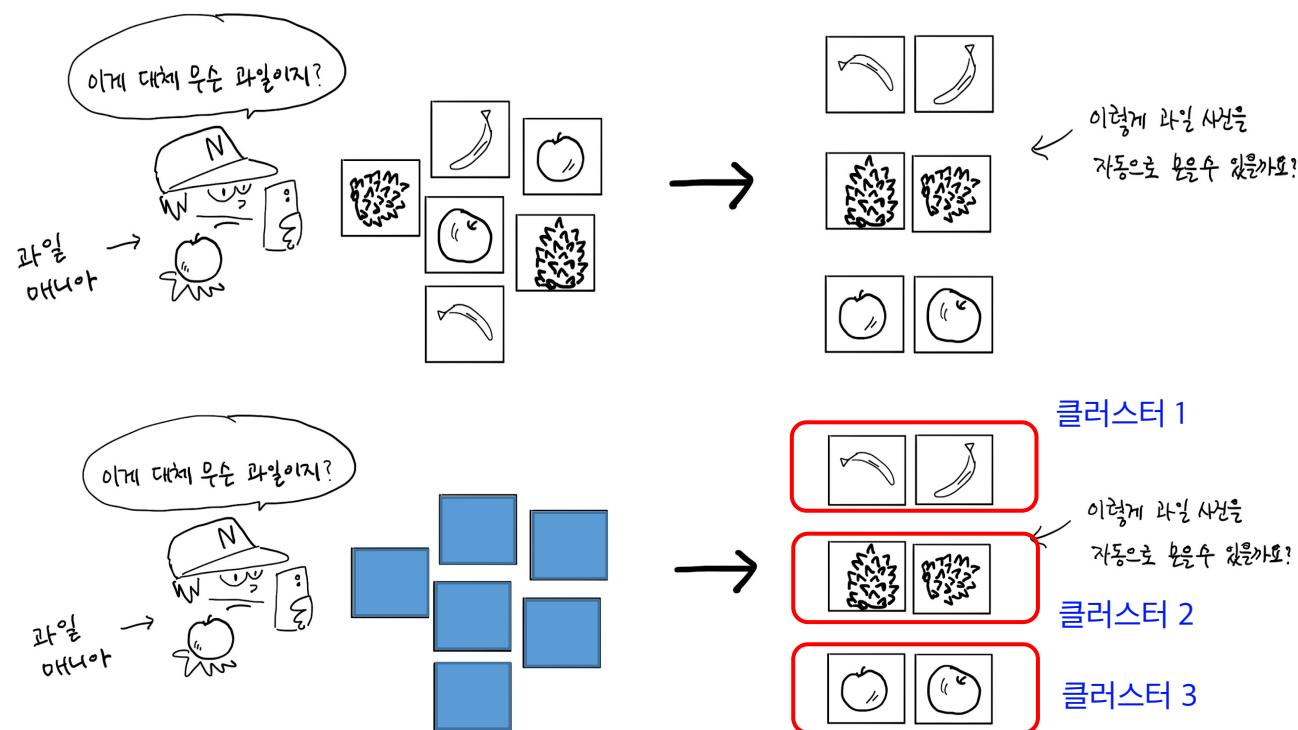
# CONTENTS

---

- 1 복습
- 2 군집과 대푯값의 개념
- 3 K-means 군집 알고리즘
- 4 주성분 분석

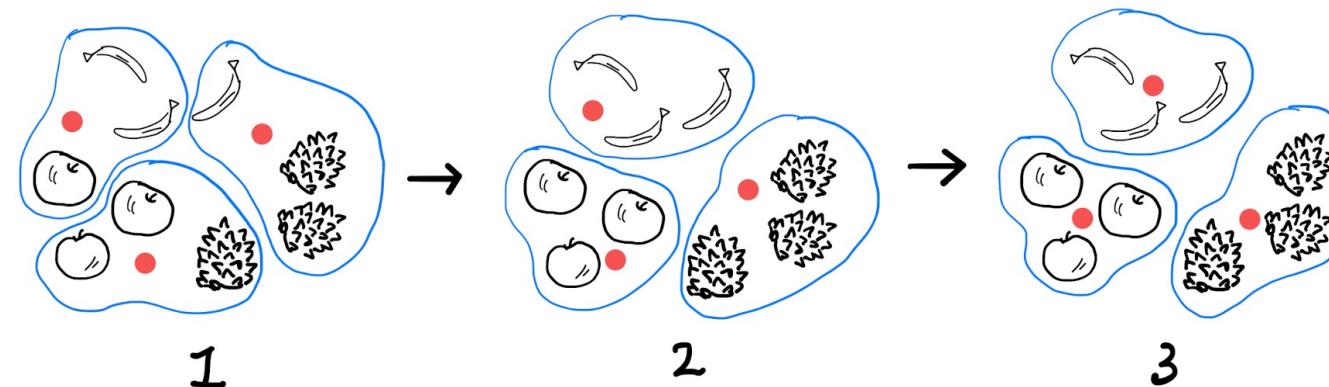
## ✎ Target 값을 알 수 없을 때

- 실제 target값을 전혀 모를 경우, 입력 데이터에 몇개의 다른 과일이 있는가 (=클러스터의 개수) 알 수 없음
  - 이때 군집 알고리즘에서 만든 그룹을 **클러스터**라고함.



### ❶ K-평균 군집 알고리즘이란?

- 클러스터(분류)의 대푯값(중심)을 랜덤으로 정하고 점차 가까운 샘플의 중심으로 이동하는 알고리즘
1. 일단 K개의 클러스터의 중심값을 정하자.
  2. 각 샘플에서 가장 가까운 클러스터 중심을 찾아 해당 클러스터의 샘플로 지정
  3. 클러스터에 속한 샘플의 평균값으로 클러스터 중심을 변경
  4. 클러스터 중심에 변화가 없을때 까지 2번 반복!



# K-means 군집 모델



# K-평균 군집 알고리즘의 구현

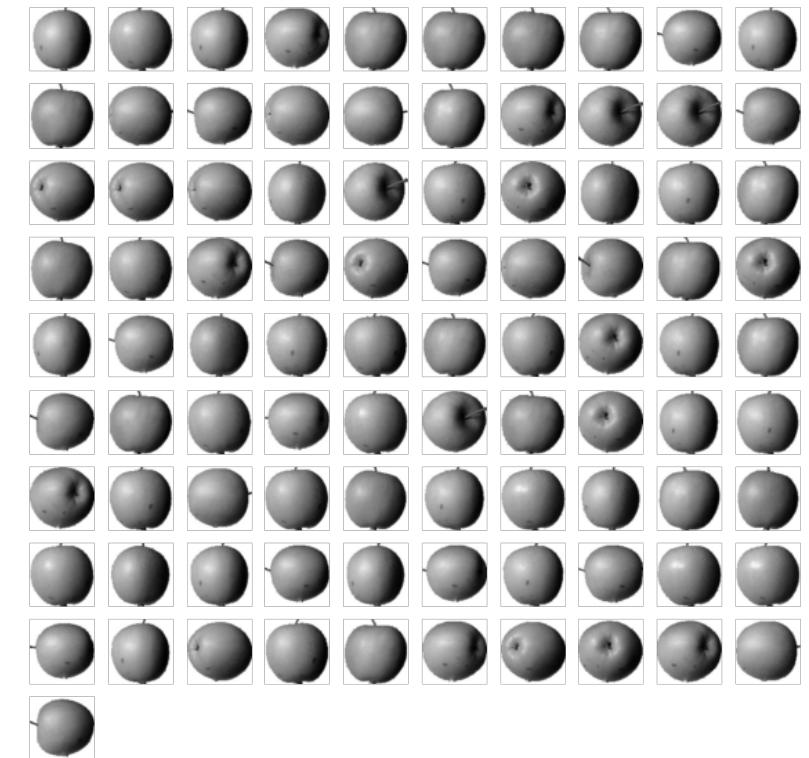
- 클러스터(분류)의 대푯값(중심)을 랜덤으로 정하고 점차 가까운 샘플의 중심으로 이동하는 알고리즘

## 📝 K-평균 군집 학습 결과 분석

- 0번째 클러스터 시각화

```
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])
```



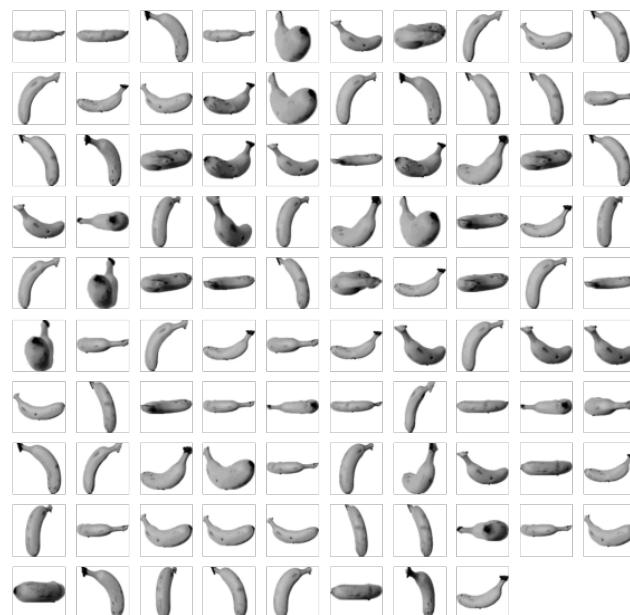
0번으로 클러스터링 된 총 91개의 샘플 시각화  
클러스터링이 잘 되었다!

# K-means 군집 모델

## ✎ K-평균 군집 학습 결과 분석

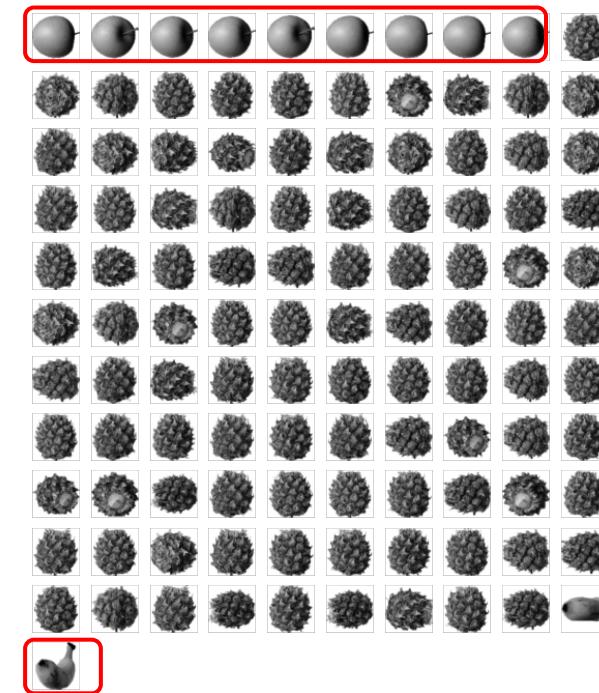
- 1, 2번째 클러스터 시각화

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



1번으로 클러스터링 된 총 98개의 샘플 시각화

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



2번 클러스터엔 사과 9개, 바나나2개, 나머지 파인애플로 클러스터링됨

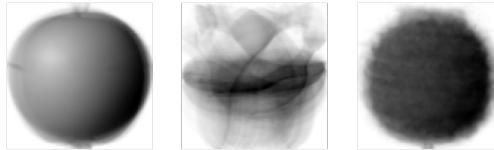
그렇다면 각 클러스터의 중심(**대푯값**)은 어떻게 생겼을까?

## ✎ K-평균 군집 학습 결과 분석

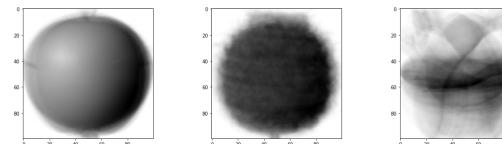
- 각 클러스터의 중심(대푯값)을 시각화 해보자

각 클러스터의 중심값을 저장한 변수

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



K-means로 구한 중심(대푯값)



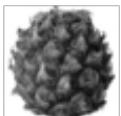
기존에 우리가 정답을 알고 있을 때 구한 대푯값

```
print(km.transform(fruits_2d[100:101]))  
[[5267.70439881 8837.37750892 3393.8136117 ]]
```

샘플 한 개를 각 클러스터 중심까지 거리로 변환해주는 함수

```
print(km.predict(fruits_2d[100:101]))  
[2]
```

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



```
print(km.n_iter_)  
3
```

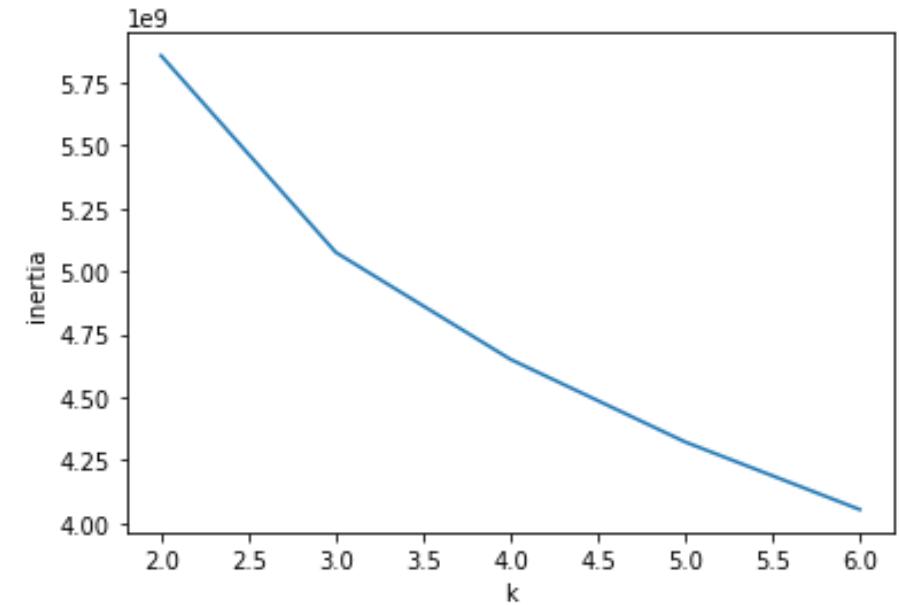
K-means 알고리즘이 대푯값을 잘 찾았구나!  
그런데 K가 3개인 걸 우리는 이미 알고 있었네?? 반칙 아닌가...  
최적의 K를 찾는 방법은 뭘까?

### 📝 K-평균 군집 학습의 hyperparameter 최적화

- 실전에선 K가 몇개인지 알 수 없다. 최적의 중심 개수 K는 어떻게 구할 수 있을까?
  - 정답은 없으나 이너셔(inertia)를 활용한 엘보우(elbow) 방법을 많이 활용함.
  - 이너셔: 클러스터 중심과 클러스터의 샘플들 사이의 거리의 제곱 합을 통해 클러스터에 속한 샘플이 얼마나 가깝게 모여 있는지 나타내는 값

```
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()
```



당연히 k값을 증가시킬 때마다 Inertia값이 작아짐  
따라서 엘보우(변곡점)이 생기는 부분을 최적값이라고 판단

### K-means를 마치며

여러분은 K-mans 알고리즘을 python과 numpy만 이용해 구현할 수 있을까요?

# CONTENTS

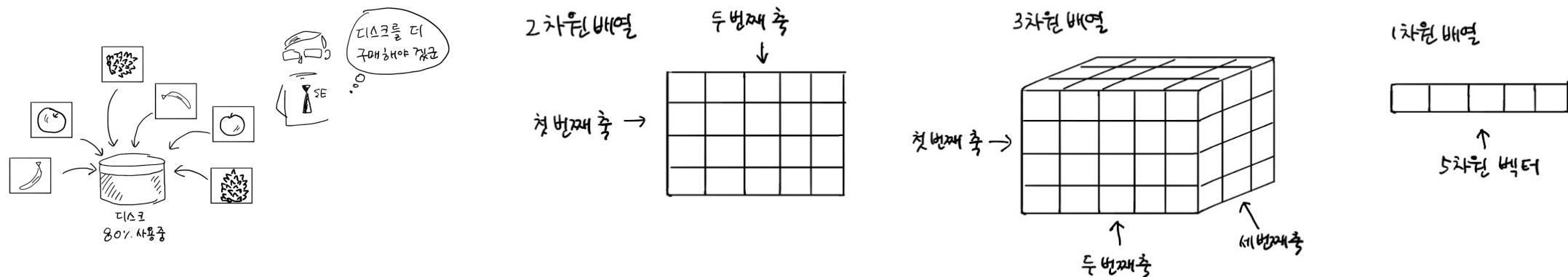
---

- 1 복습
- 2 군집과 대푯값의 개념
- 3 K-means 군집 알고리즘
- 4 주성분 분석

## 차원(dimension)의 개념과 차원 축소의 필요성

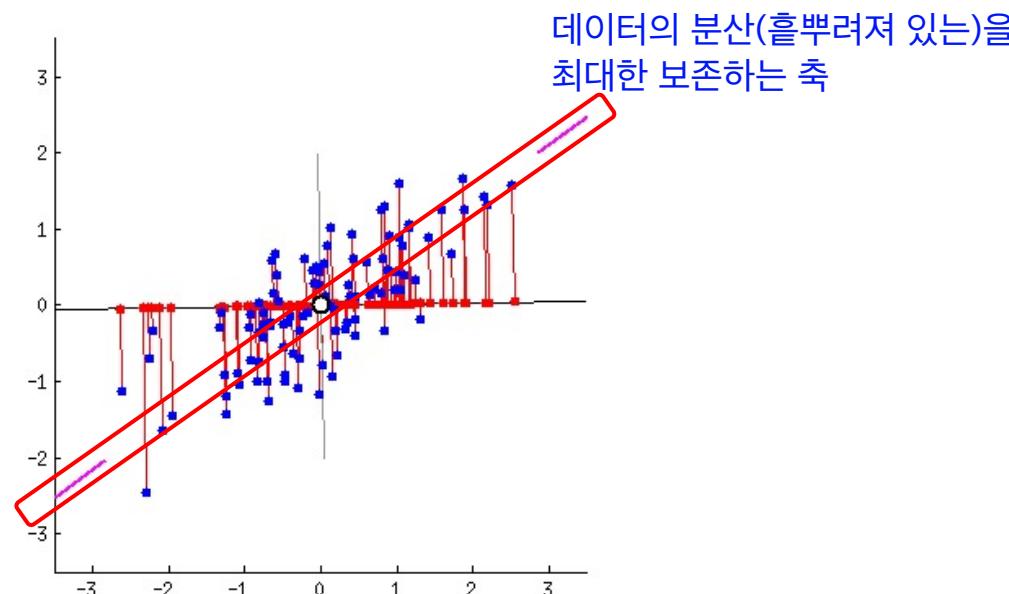
- 앞서 파일 사진의 경우 차원(dimension)이 10000인 vector로 표현했다. 너무 크지않나?
  - 데이터가 많아질 수록 기하 급수적으로 더 많은 디스크 용량과 연산 시간이 필요함, 특성이 많으면 overfit될 확률 up
  - 따라서, 효과적으로 데이터의 차원을 축소 할 방법이 연구됨

Linear regression의 feature engineering을 생각해보자!



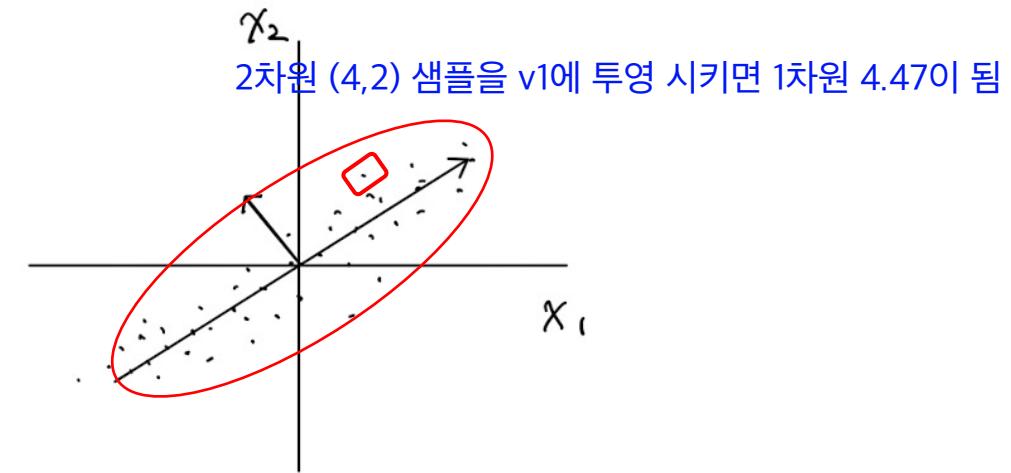
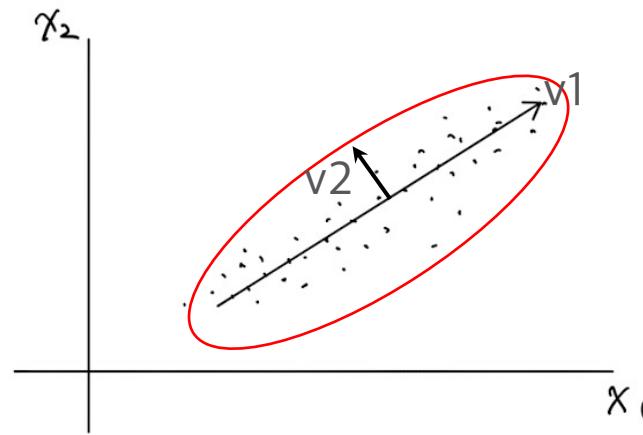
### 주성분 분석 (Principal Component Analysis) 의 개념

- 개념: 데이터 집합 A의 분포의 주성분(중요한 성분)을 찾아주는 방법 == 중요한 부분만 기억하겠다
- 방법: 데이터를 한개의 축으로 사상 시켰을 때 그 **분산**이 가장 커지는 축을 첫 번째 주성분 (벡터), 두 번째로 커지는 축을 두 번째 주성분으로 놓이도록 새로운 좌표계로 데이터를 선형 변환



## 주성분 분석 (Principal Component Analysis) 의 개념

- 개념: 데이터 집합 A의 분포의 주성분(중요한 성분)을 찾아주는 방법 == 중요한 부분만 기억하겠다



위와 같은 데이터의 분포 특성을 가장 잘 설명할 수 있는 방법은?

-  $v1, v2$  두 개의 벡터로 데이터 분포를 설명하는 것이다.  $v1$ 의 방향과 크기, 그리고  $v2$ 의 방향과 크기를 알면 이 데이터 분포가 어떤 형태인지 단순하면서도 효과적으로 파악할 수 있다. 이때 고유값(벡터의 길이)가 큰 순서대로 벡터를 정렬하면 중요한 순서대로 주성분을 구성하게 된다.

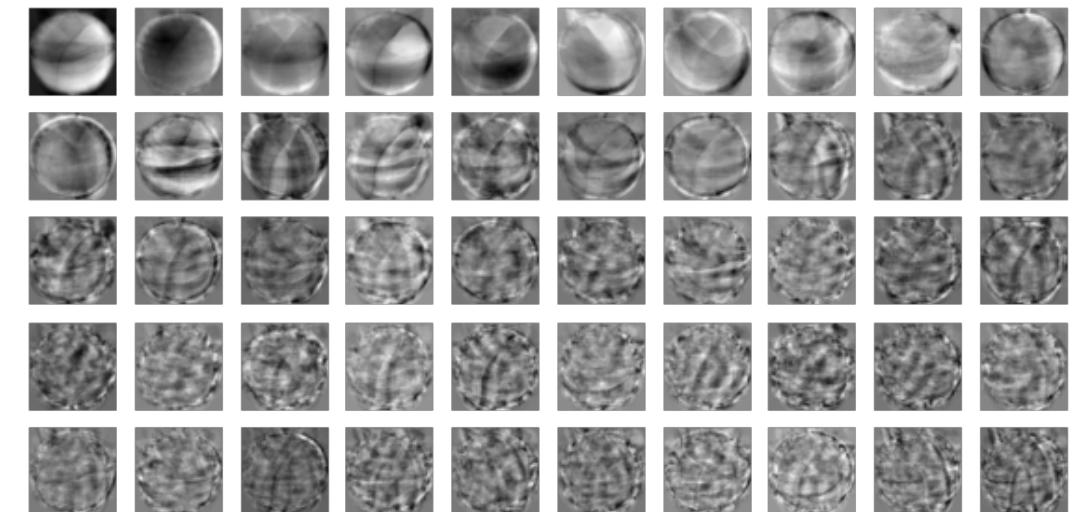
## 주성분 분석 (PCA)

# 주성분 분석 (Principal Component Analysis) 의 도입

- 과일 사진에 주성분 분석을 진행해보자.

```
from sklearn.decomposition import PCA  
  
pca = PCA(n_components=50)  
pca.fit(fruits_2d)      주성분 몇개를 추출 할래?  
  
print(pca.components_.shape)  
(50, 10000) 추출된 50개의 주성분  
  
draw_fruits(pca.components_.reshape(-1, 100, 100))  
  
print(fruits_2d.shape)  
(300, 10000)  
  
fruits_pca = pca.transform(fruits_2d)  
print(fruits_pca.shape) 실제 샘플을 추출된 주성분에 투영시킴  
(300, 50)
```

추출된 50개의 주성분을 시각화

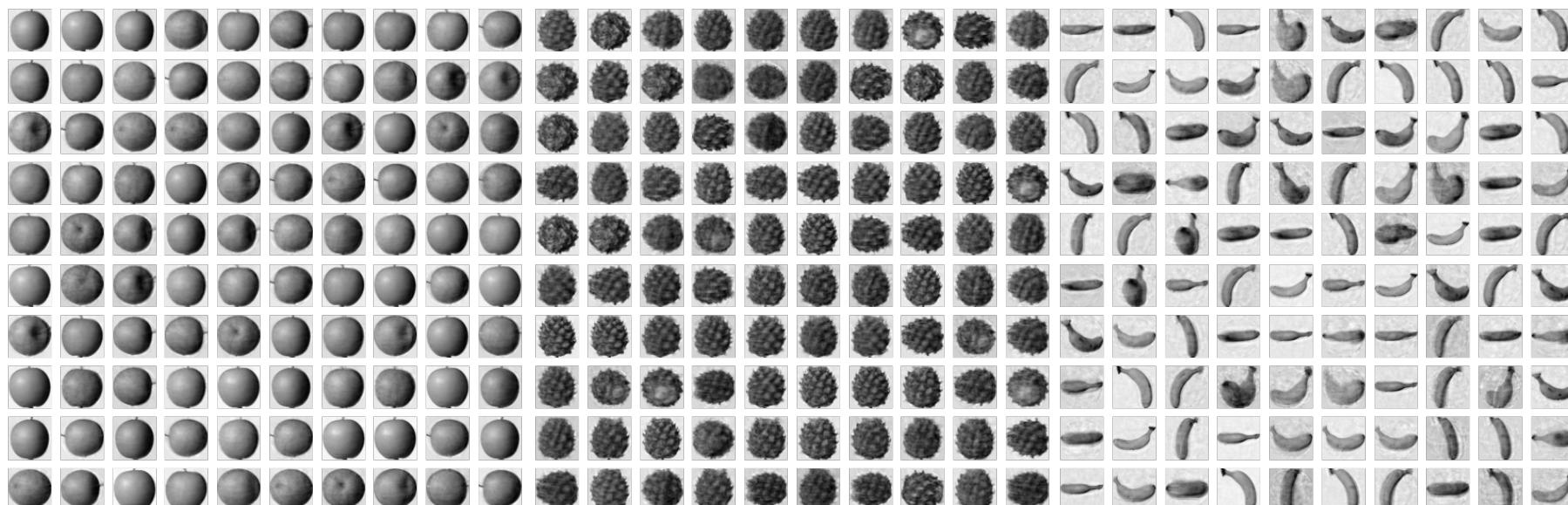


## 주성분 분석 복원

- 차원이 10000 --> 50으로 변환된 파일 사진을 다시 10000 차원으로 복원할 수 없을까?

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

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

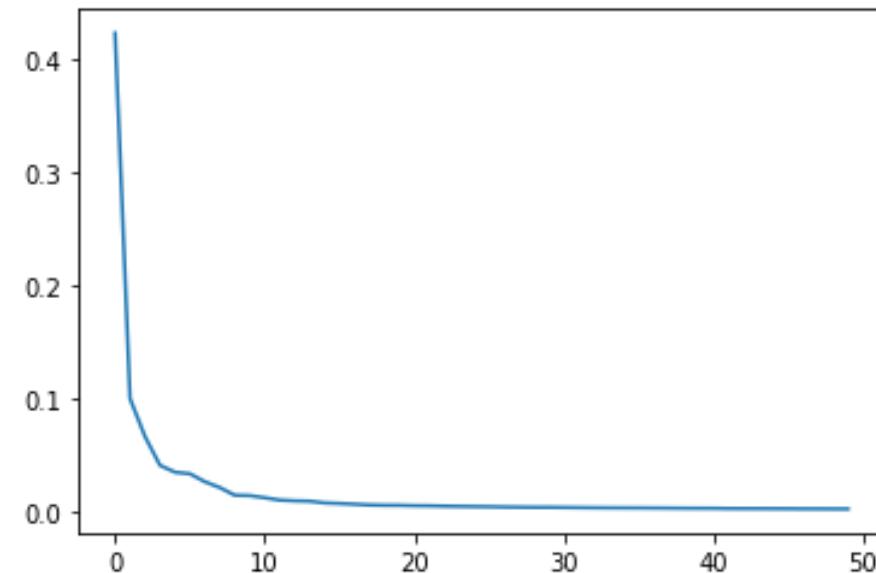


## ✎ 주성분 분석의 설명된 분산

- 설명된 분산: 추출된 주성분이 분산을 얼마나 잘 나타내는지 기록한 값
  - 추출한 50개의 주성분으로 표현하고 있는 총 분산의 비율

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

plt.plot(pca.explained_variance_ratio_)
```



## ✎ 주성분 분석을 활용한 Logistic Regression

- 주성분 분석을 도입해서 차원을 축소한 후 학습하면 성능이 나빠질까?
  - Logistic Regression으로 두 경우를 비교 평가해보자

```
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'])) 50차원으로 줄인 데이터
1.0
print(np.mean(scores['fit_time']))
0.03938336372375488
```

```
pca = PCA(n_components=0.5)
pca.fit(fruits_2d)
print(pca.n_components_)
2 2차원으로 줄인 데이터

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

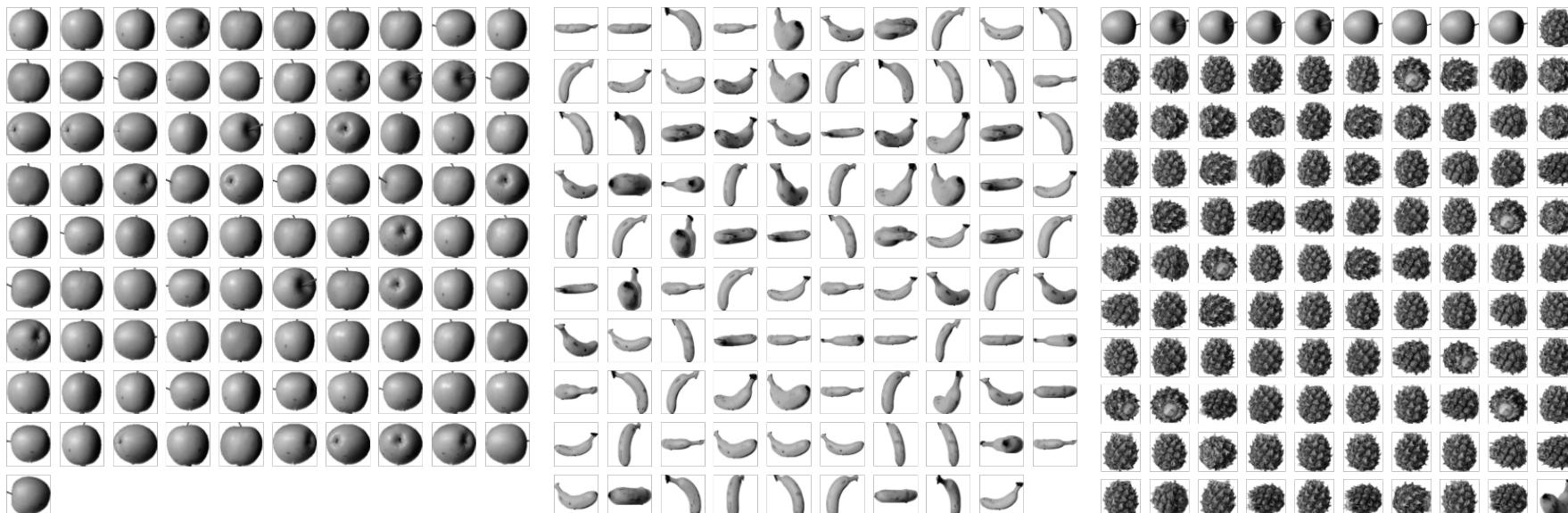
scores = cross_validate(lr, fruits_pca, target)
print(np.mean(scores['test_score']))
0.9933333333333334
print(np.mean(scores['fit_time']))
0.048157548904418944
```

## 주성분 분석을 활용한 K-means 클러스터링

- 주성분 분석을 도입해서 차원을 축소한 후 클러스터링을 하면 성능이 나빠질까?

```
km = KMeans(n_clusters=3, random_state=42)
km.fit(fruits_pca) 2차원으로 줄인 데이터

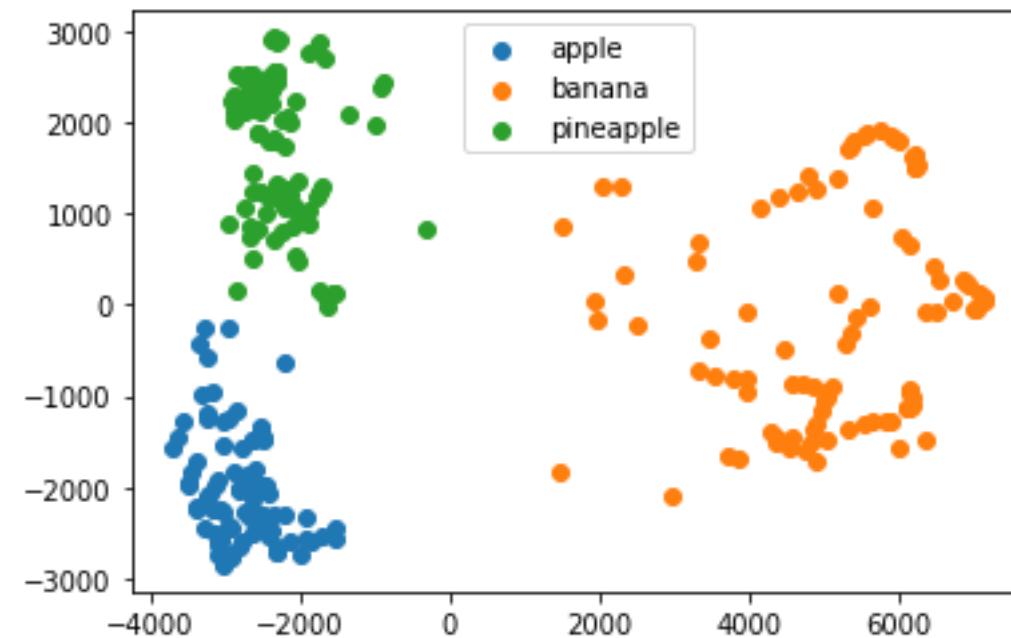
print(np.unique(km.labels_, return_counts=True))
(array([0, 1, 2], dtype=int32), array([ 91,  99, 110]))
```



## 주성분 분석을 활용한 시각화

- 주성분 분석을 도입하면 차원이 줄어들기 때문에 시각화에 많이 사용된다.

```
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()
```



---

감사합니다.