주성분 분석

구글 코랩에서 실행하기

PCA 클래스

In [1]:
```
!wget https://bit.ly/fruits_300_data -O fruits_300.npy
```
```
--2023-10-29 12:47:40--  https://bit.ly/fruits_300_data
Resolving bit.ly (bit.ly)... 67.199.248.11, 67.199.248.10
Connecting to bit.ly (bit.ly)|67.199.248.11|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://github.com/rickiepark/hg-mldl/raw/master/fruits_300.npy [following]
--2023-10-29 12:47:40--  https://github.com/rickiepark/hg-mldl/raw/master/fruits_300.npy
Resolving github.com (github.com)... 140.82.113.3
Connecting to github.com (github.com)|140.82.113.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/rickiepark/hg-mldl/master/fruits_300.npy [following]
--2023-10-29 12:47:41--  https://raw.githubusercontent.com/rickiepark/hg-mldl/master/fruits_300.npy
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.109.133, 185.199.108.133, 185.199.111.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.109.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3000128 (2.9M) [application/octet-stream]
Saving to: ‘fruits_300.npy’

fruits_300.npy      100%[===================>]   2.86M  --.-KB/s    in 0.09s

2023-10-29 12:47:41 (31.5 MB/s) - ‘fruits_300.npy’ saved [3000128/3000128]
```

In [2]:
```
import numpy as np

fruits = np.load('fruits_300.npy')
fruits_2d = fruits.reshape(-1, 100*100) # 2차원 배열로 변경
```

In [3]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components=50)
# PCA 객체 pca 생성 후, 주성분 개수인 n_components을 50으로 설정
pca.fit(fruits_2d)
# 모델 학습
```

Out[3]:
```
PCA
PCA(n_components=50)
```

In [4]:
```
print(pca.components_.shape)
```
```
(50, 10000)
```

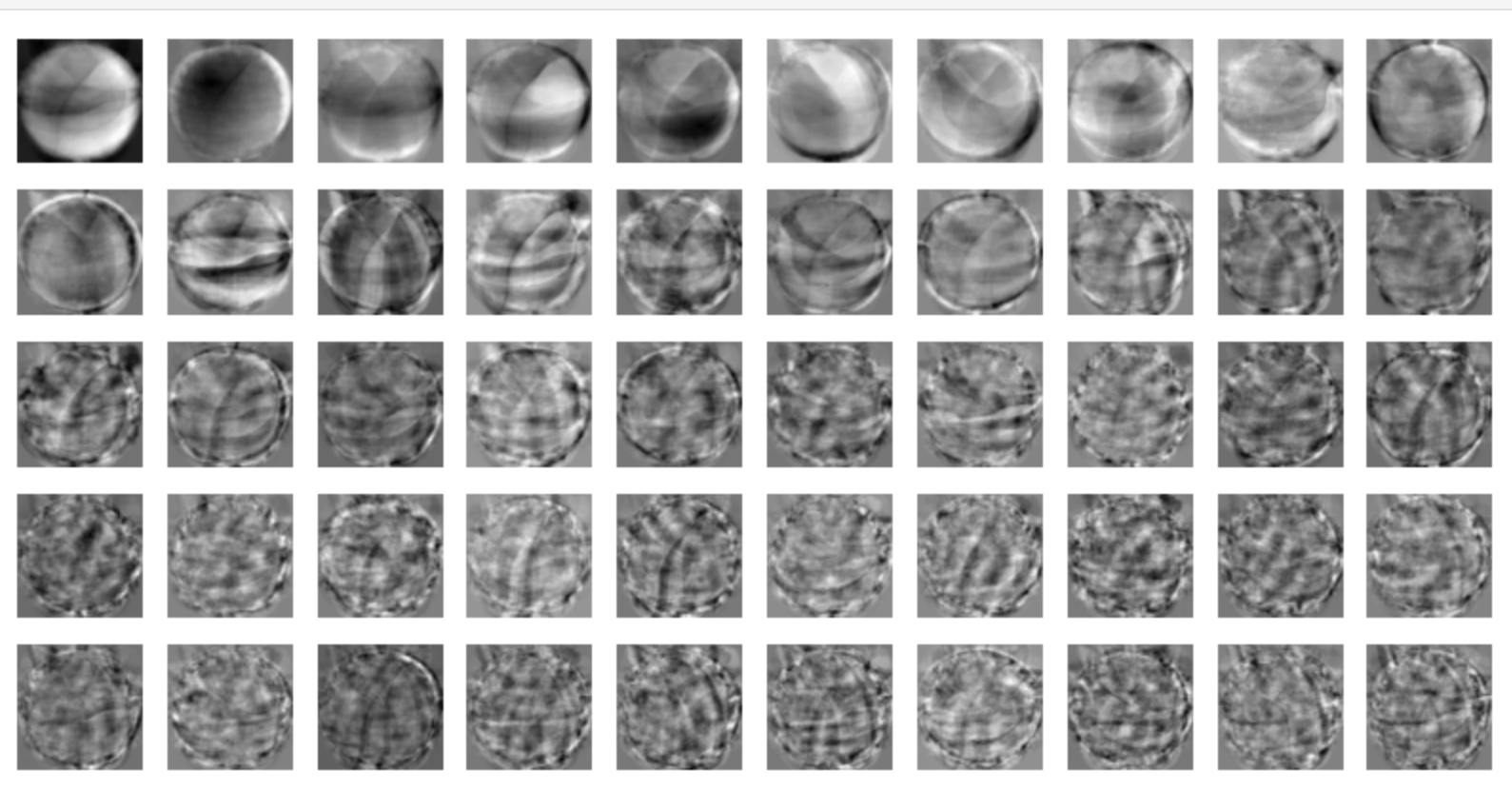
In [5]:
```
import matplotlib.pyplot as plt

def draw_fruits(arr, ratio=1):# 각 클러스터가 어떤 그림을 나타내는지 그림으로 출력하는 draw_fruits() 함수
    n = len(arr)

    rows = int(np.ceil(n/10))

    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:
                axs[i, j].imshow(arr[i*10 + j], cmap='gray_r')
            axs[i, j].axis('off')
    plt.show()
```

In [6]:
```
draw_fruits(pca.components_.reshape(-1, 100, 100)) # 주성분을 100 x 100 크기의 이미지로 출력
```



In [9]:
```
print(fruits_2d.shape)
```
```
(300, 10000)
```

In [10]:
```
fruits_pca = pca.transform(fruits_2d) # transform()을 통해서 원본 데이터의 차원을 50으로 줄임
```

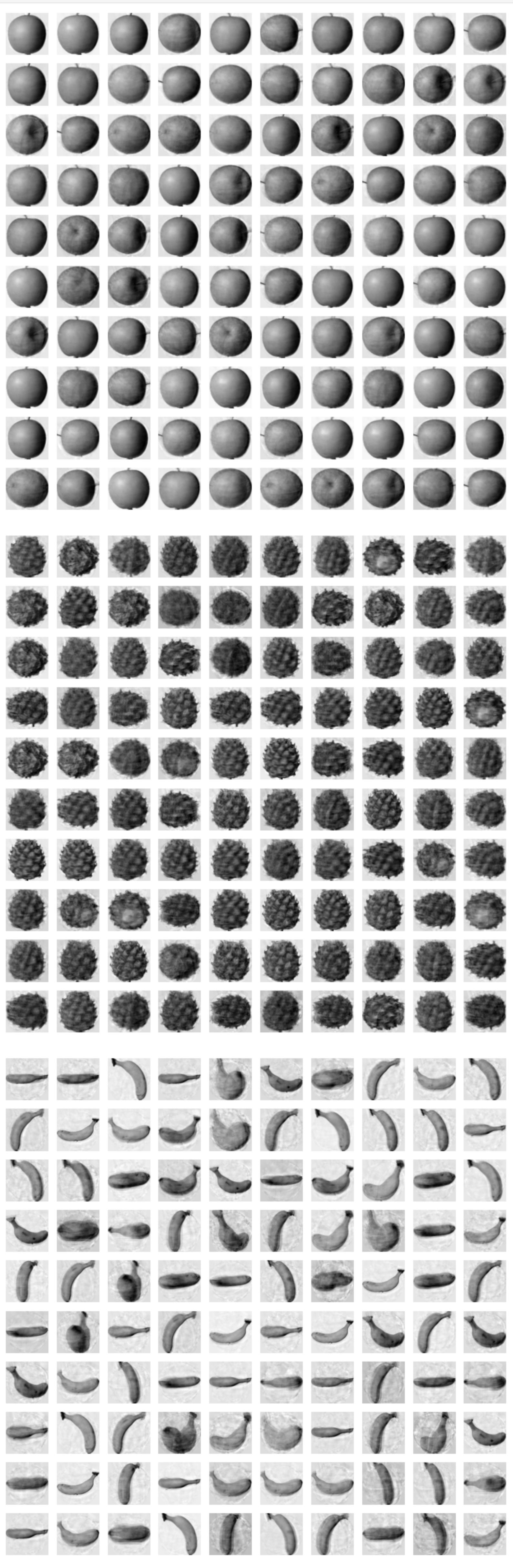
In [11]:
```
print(fruits_pca.shape)
```
```
(300, 50)
```

원본 데이터 재구성

In [12]:
```
fruits_inverse = pca.inverse_transform(fruits_pca) # inverse_transform() 메서드를 통해 원본 데이터를 복원
print(fruits_inverse.shape)
```
```
(300, 10000)
```

In [13]:
```
fruits_reconstruct = fruits_inverse.reshape(-1, 100, 100)
```

In [14]:
```
for start in [0, 100, 200]:
    draw_fruits(fruits_reconstruct[start:start+100])
    print("\n")
    # 사과, 파인애플, 바나나를 각 100개씩 출력
```

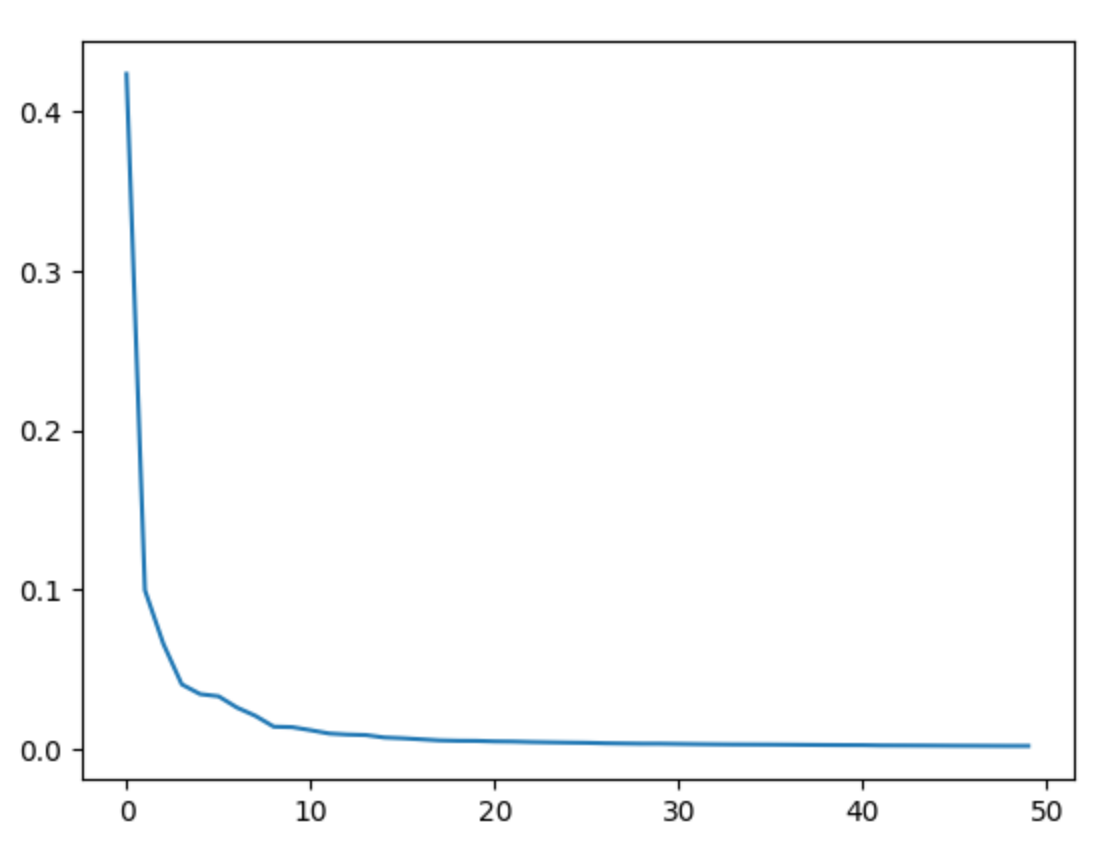


설명된 분산

In [15]:
```
print(np.sum(pca.explained_variance_ratio_))
# explained_variance_ratio_을 통해서 각 주성분의 설명된 분산 비율을 기록
# np.sum()을 통해서 분산 비율을 모두 더해 50개의 주성분으로 표현하고 있는 총 분산 비율을 얻음
```
```
0.9215273005864528
```

In [16]:
```
plt.plot(pca.explained_variance_ratio_)
# 주성분이 원본 데이터의 분산을 얼마나 잘 나타내는지 기록한 값을 설명된 분산이라고 함
# 첫 번째 주성분의 설명된 분산이 가장 큼
```

Out[16]:
```
[<matplotlib.lines.Line2D at 0x7ad187899180>]
```



다른 알고리즘과 함께 사용하기

In [17]:
```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
```

In [18]:
```
target = np.array([0] * 100 + [1] * 100 + [2] * 100)
# 지도 학습 모델이므로 타겟값을 생성
# 0은 사과, 1은 파인애플, 2는 바나나로 지정
# 각 과일마다 100개의 데이터가 있기 때문에 [0], [1], [2]에 100을 곱해줘서 타겟값을 생성
```

In [19]:
```
from sklearn.model_selection import cross_validate

scores = cross_validate(lr, fruits_2d, target)
print(np.mean(scores['test_score'])) # 교차 검증 점수 출력
print(np.mean(scores['fit_time']))  # 훈련 시간 출력
```
```
0.9966666666666667
2.102800893783569
```

In [20]:
```
scores = cross_validate(lr, fruits_pca, target)
print(np.mean(scores['test_score']))
print(np.mean(scores['fit_time']))
```
```
1.0
0.03332781791687012
```

In [21]:
```
pca = PCA(n_components=0.5)  # 0~1 사이의 값을 ㅁ넣어서 분산의 비율을 입력 가능
pca.fit(fruits_2d)
```

Out[21]:
```
PCA
PCA(n_components=0.5)
```

In [22]:
```
print(pca.n_components_) # 주성분 개수 출력
```
```
2
```

In [24]:
```
fruits_pca = pca.transform(fruits_2d)
# 주성분을 2개 갖는 모델의 원본 데이터를 변환
print(fruits_pca.shape)
```
```
(300, 2)
```

In [25]:
```
scores = cross_validate(lr, fruits_pca, target)
print(np.mean(scores['test_score']))
print(np.mean(scores['fit_time']))
```
```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
0.9933333333333334
0.06181974411010742
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

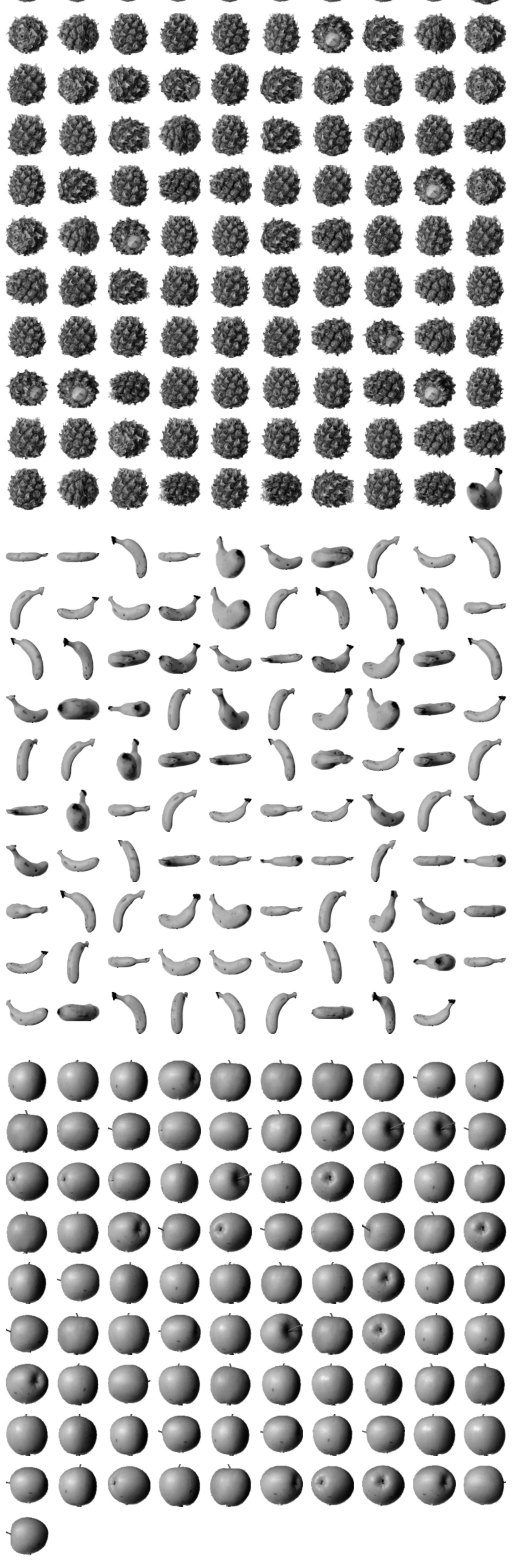
In [26]:
```
from sklearn.cluster import KMeans

km = KMeans(n_clusters=3, random_state=42)
km.fit(fruits_pca)
```
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

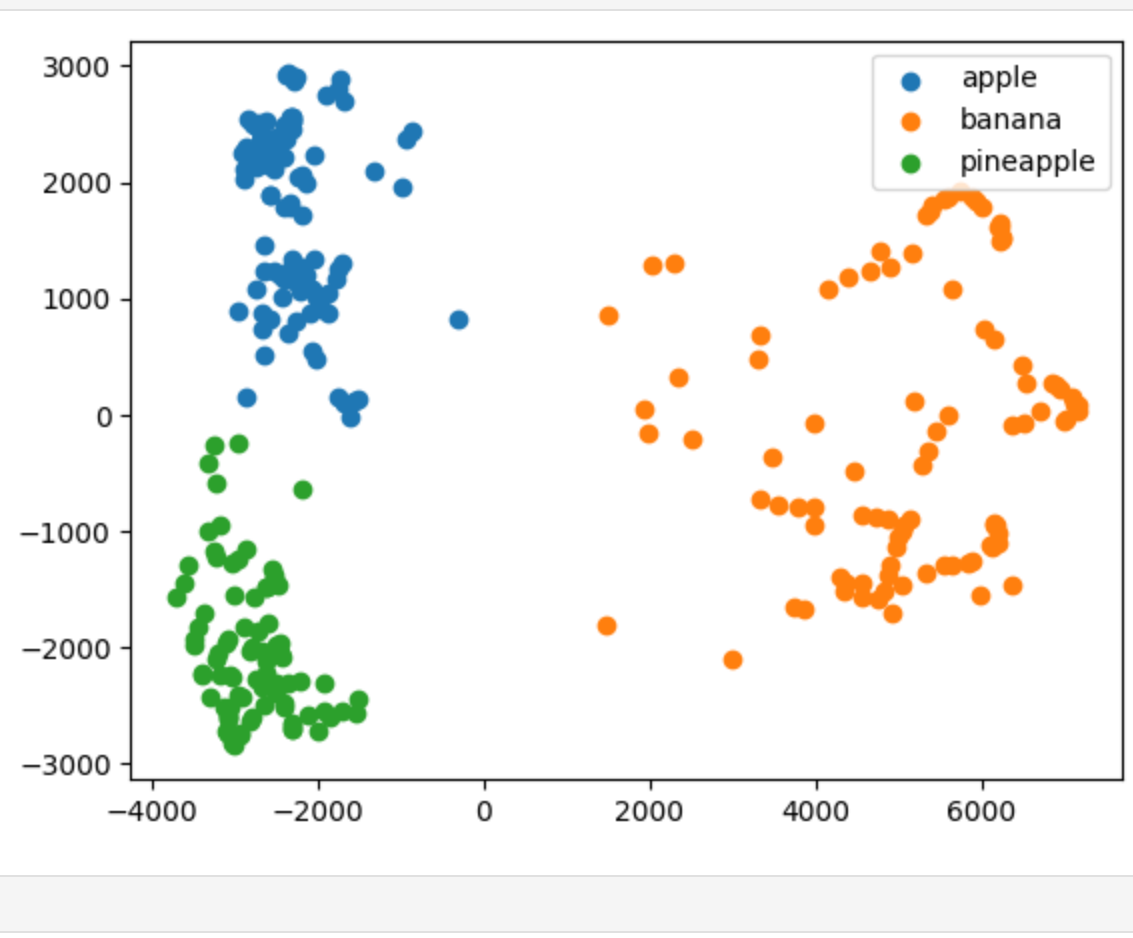
Out[26]:
```
KMeans
KMeans(n_clusters=3, random_state=42)
```

In [27]:
```
print(np.unique(km.labels_, return_counts=True))
# fruits_pca로 찾은 클러스터는 각각 91, 99, 110개의 샘플을 포함
```
```
(array([0, 1, 2], dtype=int32), array([110,  99,  91]))
```

In [28]:
```
for label in range(0, 3):
    draw_fruits(fruits[km.labels_ == label])
    print("\n")
```



In [29]:
```
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()
```



In [29]: