

차원축소하기 전 원본 digits 데이터이다.

```
digits = datasets.load_digits()

X = digits.data
Y = digits.target
feature_names = digits.feature_names
target_names = digits.target_names
n_samples = X.shape[0]
images = X.reshape((n_samples, -1))

_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
for ax, image in zip(axes, images):
    ax.set_axis_off()
    image = image.reshape(8, 8)
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
```



32차원으로 축소한 X_reduced 데이터를 다시 64차원으로 복구하고 이미지를 출력하였다.


```
X_reduced=student_pca(X,32)

X_centered = X - X.mean(axis=0)
U,s,Vt = np.linalg.svd(X_centered)
w2 = Vt.T[:, :32]
X_recovered = X_reduced.dot(w2.T)
X_recovered = X_recovered + X.mean(axis=0)

n_samples = X_recovered.shape[0]
images = X_recovered.reshape((n_samples, -1))

_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
for ax, image in zip(axes, images):
    ax.set_axis_off()
    image = image.reshape(8, 8)
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
loss = np.sum((X-X_recovered)**2,axis=1).mean()
print(loss)

40.42470493509364
```



4차원으로 축소한 X_reduced 데이터를 다시 64차원으로 복구하고 이미지를 출력하였다.

```
X_reduced=student_pca(X,4)

X_centered = X - X.mean(axis=0)
U,s,Vt = np.linalg.svd(X_centered)
w2 = Vt.T[:, :4]
X_recovered = X_reduced.dot(w2.T)
X_recovered = X_recovered + X.mean(axis=0)

n_samples = X_recovered.shape[0]
images = X_recovered.reshape((n_samples, -1))

_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
for ax, image in zip(axes, images):
    ax.set_axis_off()
    image = image.reshape(8, 8)
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
loss = np.sum((X-X_recovered)**2,axis=1).mean()
print(loss)

616.1911300562696
```



3차원으로 축소한 X_{reduced} 데이터를 다시 64차원으로 복구하고 이미지를 출력하였다.

```
X_reduced=student_pca(X,3)
```

```
X_centered = X - X.mean(axis=0)
U,s,Vt = np.linalg.svd(X_centered)
w2 = Vt.T[:, :3]
X_recovered = X_reduced.dot(w2.T)
X_recovered = X_recovered + X.mean(axis=0)
```

```
n_samples = X_recovered.shape[0]
images = X_recovered.reshape((n_samples, -1))
```

```
_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
for ax, image in zip(axes, images):
    ax.set_axis_off()
    image = image.reshape(8, 8)
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
loss = np.sum((X-X_recovered)**2,axis=1).mean()
print(loss)
```

```
717.2362446162666
```



2차원으로 축소한 X_{reduced} 데이터를 다시 64차원으로 복구하고 이미지를 출력하였다.

```
X_reduced=student_pca(X,2)
```

```
X_centered = X - X.mean(axis=0)
U,s,Vt = np.linalg.svd(X_centered)
w2 = Vt.T[:, :2]
X_recovered = X_reduced.dot(w2.T)
X_recovered = X_recovered + X.mean(axis=0)
```

```
n_samples = X_recovered.shape[0]
images = X_recovered.reshape((n_samples, -1))
```

```
_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
for ax, image in zip(axes, images):
    ax.set_axis_off()
    image = image.reshape(8, 8)
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
loss = np.sum((X-X_recovered)**2,axis=1).mean()
print(loss)
```

```
858.9447808487329
```



$X_{\text{reconstruction}} = X_{\text{reduced}} \cdot \text{Eigenvectors.T} + \text{Mean}$ 공식을 기반으로 reconstruct 하였다.

차원을 더 많이 축소하였다가 복구할수록 MSE 오차가 더 커진다. 32차원일때는 대략 40, 4차원일 때는 616, 3차원일때는 717, 2차원일때는 858 정도가 나온다. 차원을 줄일 때 원래 데이터의 분산 정보를 잃었기 때문에 완벽하게 복구할 수는 없다. 차원을 더 많이 줄일수록 더 많은 분산정보를 잃기 때문에 2차원으로 줄이고 복구하면 오차가 제일 클 수 밖에 없다.