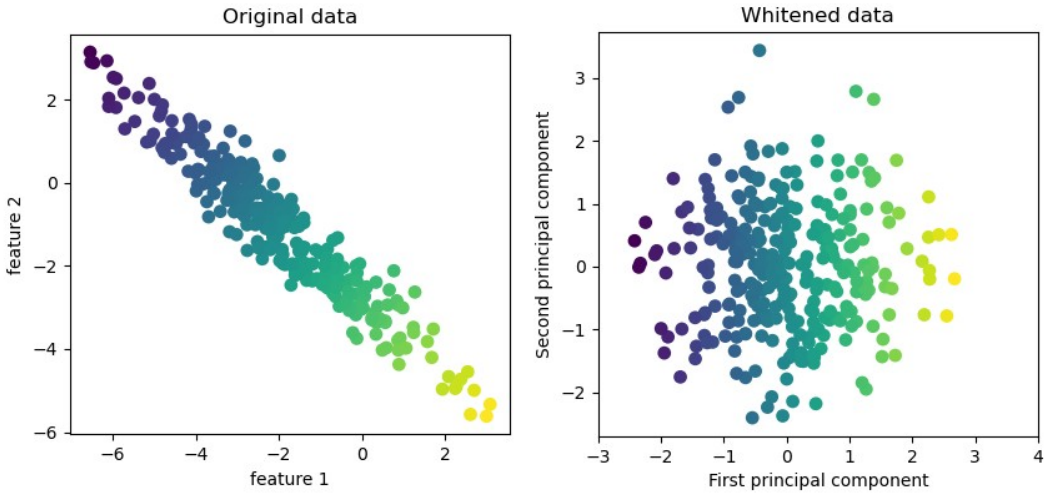
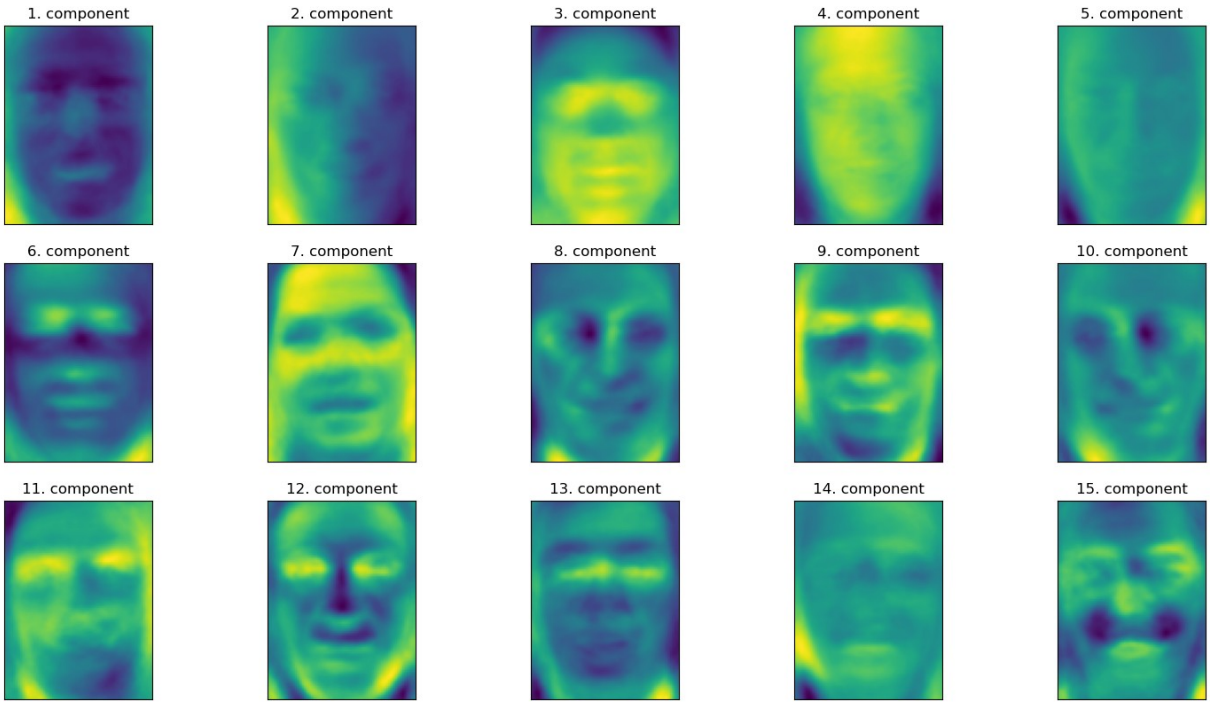
Q2-2. Eigenface Feature Extraction

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

from sklearn.datasets import fetch_lfw_people

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

##########################################################################################################################################
### Dataset Preparation and Analysis #####################################################################################################
##########################################################################################################################################
# Labeled Faces in the Wild (LFW) people 데이터셋 준비
people = fetch_lfw_people(min_faces_per_person=20, resize=0.7)

image_shape = people.images[0].shape    # 데이터셋 이미지 데이터 형태

# 데이터셋에서 10 개의 이미지 데이터 전시
fix, axes = plt.subplots(2, 5, figsize=(15, 8), subplot_kw={'xticks' : (), 'yticks' : ()})
for target, image, ax in zip(people.target, people.images, axes.ravel()):
    ax.imshow(image)
    ax.set_title(people.target_names[target])

plt.show()

print('people.images.shape : {}'.format(people.images.shape))       # 데이터셋의 이미지 형태 출력
print('Number of classes : {}'.format(len(people.target_names)))    # 데이터셋의 Class 개수 출력

# 각 클래스별 이미지 개수 출력
counts = np.bincount(people.target)
for i, (count, name) in enumerate(zip(counts, people.target_names)):
    print('{0:25} {1:3}'.format(name, count), end=' ')
    if (i + 1) % 3 == 0:
        print()

# 데이터셋 내에서 이미지의 내용과 이미지 이름이 동일한 요소만 모음
# 이미지의 내용과 이미지 이름이 불일치하는 경우를 제외시킴
mask = np.zeros(people.target.shape, dtype=np.bool)
for target in np.unique(people.target):
    mask[np.where(people.target == target)[0][:50]] = 1

X_people = people.data[mask]        # 이미지 이름이 내용과 일치하는 이미지 데이터를 사용함
y_people = people.target[mask]      # 이미지 이름이 내용과 일치하는 Target 데이터를 사용함

X_people = X_people / 255   # Grayscale 이미지 데이터를 0 과 1 사이로 구성함

# 데이터셋을 Training Set 과 Test Set 으로 분리함
X_train, X_test, y_train, y_test = train_test_split(X_people, y_people, stratify=y_people, random_state=0)

##########################################################################################################################################
### KNN Classification with Original Dataset #############################################################################################
##########################################################################################################################################
knn = KNeighborsClassifier(n_neighbors=1)   # Neighbor 1 개로 구성된 KNN Classifier 를 준비함
knn.fit(X_train, y_train)                   # Training 데이터셋으로 KNN 을 학습시킴

print()
print('Test set score of 1-nn : {:.2f}'.format(knn.score(X_test, y_test)))  # KNN Classifier 의 정확도를 출력함

# PCA Whitening 과정을 출력함
mglearn.plots.plot_pca_whitening()
plt.show()

##########################################################################################################################################
### Principle Componeny Analysis of LFW dataset ##########################################################################################
##########################################################################################################################################
pca = PCA(n_components=100, whiten=True, random_state=0).fit(X_train)   # LFW 데이터셋에 대해 100 개의 Principle Component 를 생성하는 PCA 객체 준비
X_train_pca = pca.transform(X_train)    # Principle Component 를 이용하여 Training 데이터셋을 재구성함
X_test_pca = pca.transform(X_test)      # Principle Component 를 이용하여 Test 데이터셋을 재구성함

print('X_train_pca.shape : {}'.format(X_train_pca.shape))   # PCA 기반으로 재구성된 데이터의 형태를 출력함

##########################################################################################################################################
### KNN Classification of PCA-based Dataset ##############################################################################################
##########################################################################################################################################
knn = KNeighborsClassifier(n_neighbors=1)   # Neighbor 1 개로 구성된 KNN Classifier 를 준비함
knn.fit(X_train_pca, y_train)               # PCA 기반으로 재구성된 데이터셋으로 KNN 을 학습시킴

print('Test set accuracy : {:.2f}'.format(knn.score(X_test_pca, y_test)))   # PCA 기반 Test 데이터셋으로 KNN Classifier 의 정확도를 출력함

print('pca.components_.shape : {}'.format(pca.components_.shape))   # Principle Component 의 형태를 출력함

# 사용하는 Principle Component 의 개수에 따른 결과 이미지를 출력함
fix, axes = plt.subplots(3, 5, figsize=(15, 12), subplot_kw={'xticks': (), 'yticks': ()})
for i, (component, ax) in enumerate(zip(pca.components_, axes.ravel())):
    ax.imshow(component.reshape(image_shape), cmap='viridis')
    ax.set_title("{}. component".format((i + 1)))

plt.show()

# 사용하는 Principle Component 의 개수에 따른 이미지 복구 결과를 출력함
mglearn.plots.plot_pca_faces(X_train, X_test, image_shape)

plt.show()

# PCA 기반으로 재구성된 데이터셋을 Scatter Plot 으로 출력함
mglearn.discrete_scatter(X_train_pca[:, 0], X_train_pca[:, 1], y_train)
plt.xlabel("First principal component")
plt.ylabel("Second principal component")

plt.show()
```

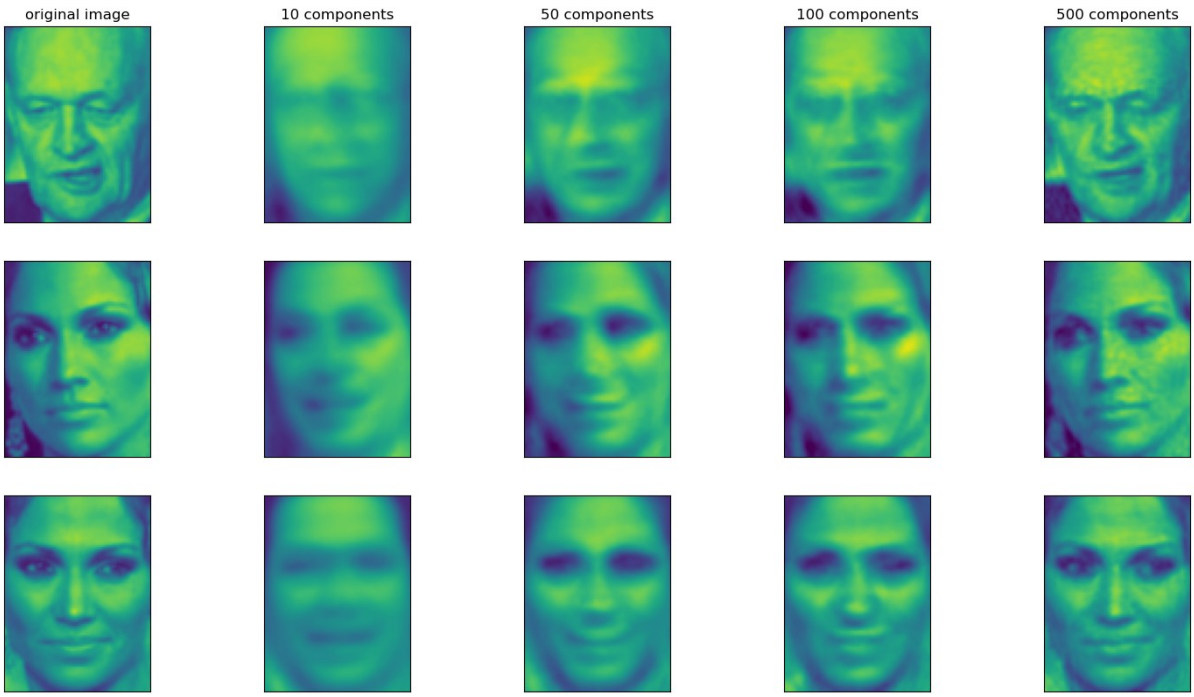
LFW 의 데이터셋 예제 10 개 출력



PCA Whitening 과정 전시



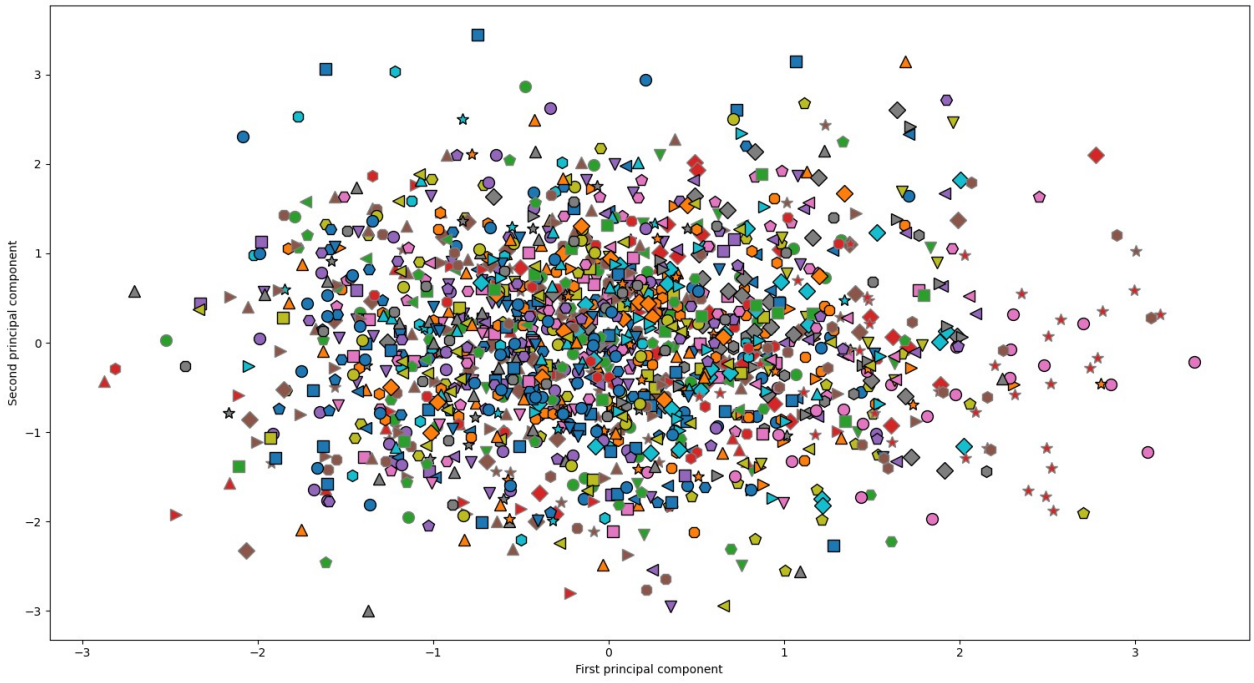
LFW 데이터셋 이미지에 대한 Principle Component 10 개 출력



Principle Component 사용 개수에 따른 이미지 복구 결과

더 많은 Principle Component 를 사용할수록 Variance 가 높아지면서

설명력이 높아지기 때문에 데이터 복구 결과가 원본과 더 유사해짐



Principle Component 재구성된 데이터셋의 분포