분해 (Decomposition)

- 큰 하나의 행렬을 여러개의 작은 행렬로 분해
- 분해 과정에서 중요한 정보만 남게됨

데이터 불러오기 및 시각화

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

from sklearn.datasets import load_iris, fetch_olivetti_faces
from sklearn.decomposition import PCA, IncrementalPCA, KernelPCA, SparsePCA
from sklearn.decomposition import TruncatedSVD, DictionaryLearning, FactorAnalysis
from sklearn.decomposition import FastICA, NMF, LatentDirichletAllocation
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

```
iris, labels = load_iris(return_X_y=True)
faces, _ = fetch_olivetti_faces(return_X_y=True, shuffle=True)
```

```
downloading Olivetti faces from https://ndownloader.figshare.com/files/5976027 to /root/sciki
```

```
def plot_iris(iris, labels):
  plt.figure()
  colors = ['navy', 'purple', 'red']
  for xy, label in zip(iris, labels):
    plt.scatter(xy[0], xy[1], color=colors[label])
```

```
def show_faces(faces):
  plt.figure()
```

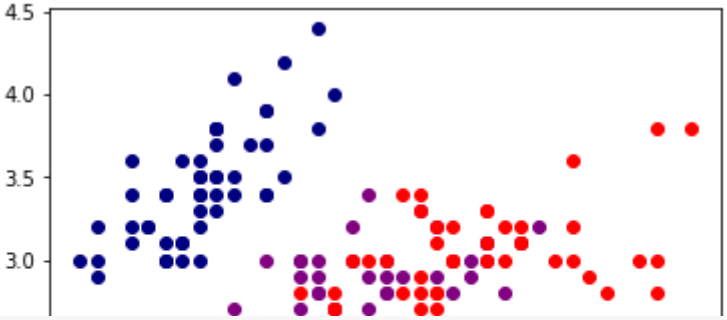
셀 삭제를 실행취소하려면 Ctrl+M Z 또는 수정 메뉴의 실행취소 옵션을 사용하세요.

```
    plt.subplot(num_rows, num_cols, i + 1)
    plt.imshow(np.reshape(faces[i], (64, 64)), cmap=plt.cm.gray)
```

```
iris.shape
```

```
(150, 4)
```

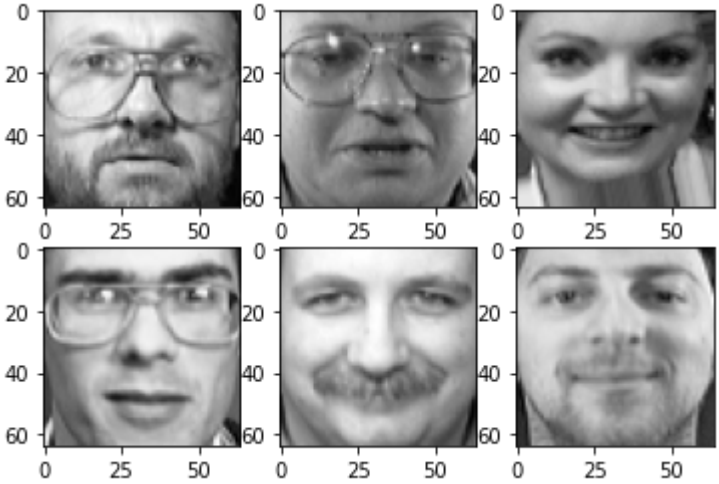
```
plot_iris(iris[:, :2], labels)
```




```
faces.shape
```

```
(400, 4096)
```

```
show_faces(faces)
```



Principal Component Analysis (PCA)

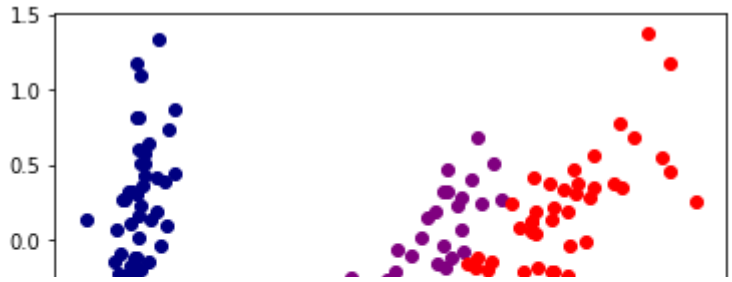
- PCA를 사용해 iris 데이터 변환
- 150×4 크기의 데이터를 150×2 크기의 행렬로 압축

셀 삭제를 실행취소하려면 Ctrl+M Z 또는 수정 메뉴의 실행취소 옵션을 사용하세요.

```
model.fit(iris)
transformed_iris = model.transform(iris)
transformed_iris.shape
```

```
(150, 2)
```

```
plot_iris(transformed_iris, labels)
```

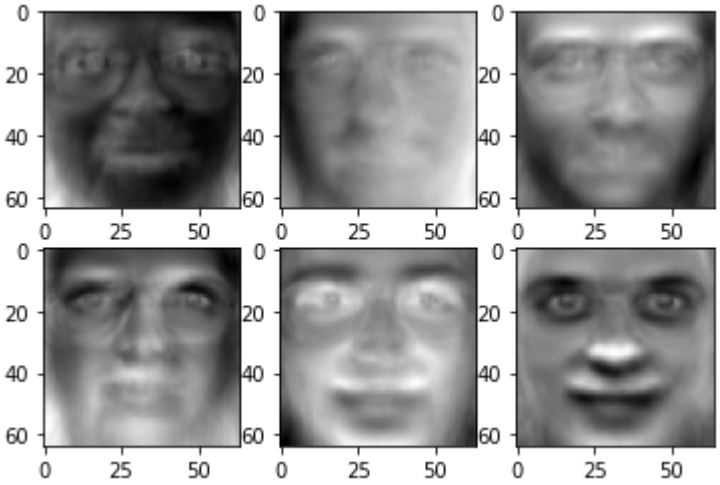


- PCA를 통해 학습된 각 컴포넌트 (6개)
- 각 컴포넌트는 얼굴의 주요 특징을 나타냄

```
model = PCA(n_components=2*3, random_state=0)
model.fit(faces)
faces_components = model.components_
faces_components.shape
```

```
(6, 4096)
```

```
show_faces(faces_components)
```



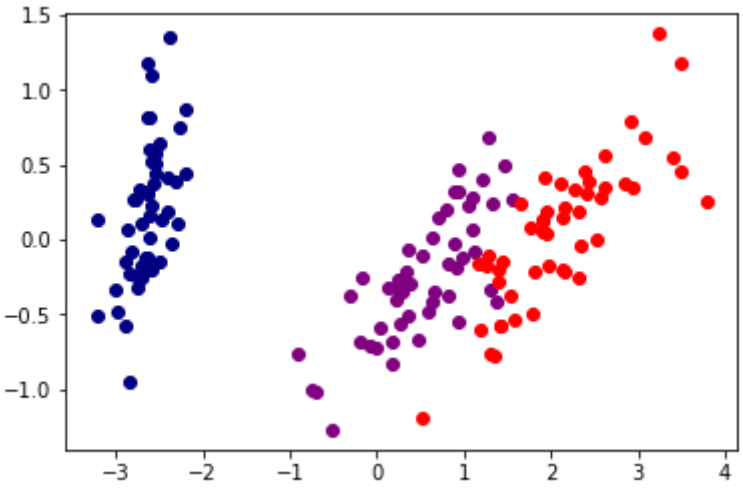
셀 삭제를 실행취소하려면 Ctrl+M Z 또는 수정 메뉴의 실행취소 옵션을 사용하세요.

- PCA는 SVD 알고리즘 실행을 위해 전체 학습용 데이터 셋을 메모리에 올려야 함
- Incremental PCA는 학습 데이터를 미니 배치 단위로 나누어 사용
- 학습 데이터가 크거나 온라인으로 PCA 적용이 필요할 때 유용

```
model = IncrementalPCA(n_components=2)
model.fit(iris)
transformed_iris = model.transform(iris)
transformed_iris.shape
```

```
(150, 2)
```

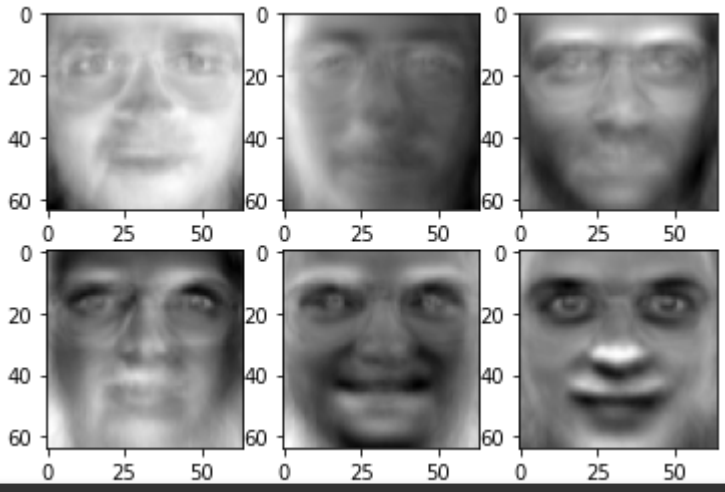
```
plot_iris(transformed_iris, labels)
```




```
model = IncrementalPCA(n_components=2*3)
model.fit(faces)
faces_components = model.components_
faces_components.shape
```

```
(6, 4096)
```

```
show_faces(faces_components)
```



셀 삭제를 실행취소하려면 Ctrl+M Z 또는 수정 메뉴의 실행취소 옵션을 사용하세요.

Kernel PCA

- 차원 축소를 위한 복잡한 비선형 투형

```
model = KernelPCA(n_components=2, kernel='rbf', random_state=0)
model.fit(iris)
transformed_iris = model.transform(iris)
plot_iris(transformed_iris, labels)
```


```
model = KernelPCA(n_components=2*3, kernel='rbf', random_state=0)
model.fit(faces)
faces_components = model.components_
faces_components.shape
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-21-031b9b70f2e7> in <module>()
      1 model = KernelPCA(n_components=2*3, kernel='rbf', random_state=0)
      2 model.fit(faces)
----> 3 faces_components = model.components_
      4 faces_components.shape

AttributeError: 'KernelPCA' object has no attribute 'components_'
```

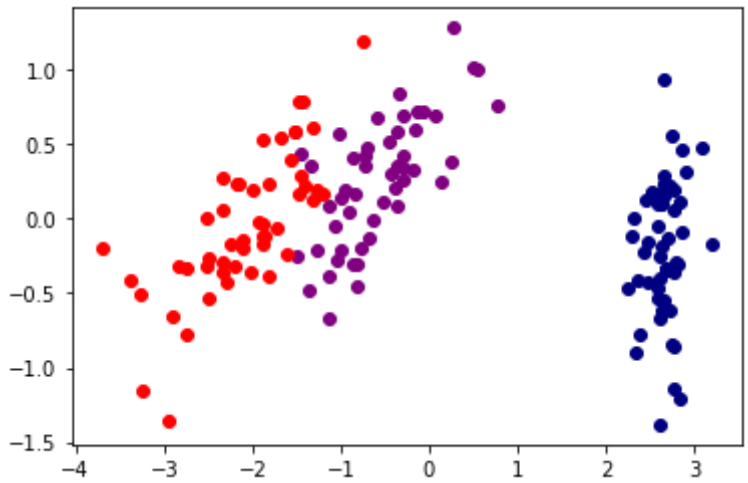
SEARCH STACK OVERFLOW

Sparse PCA

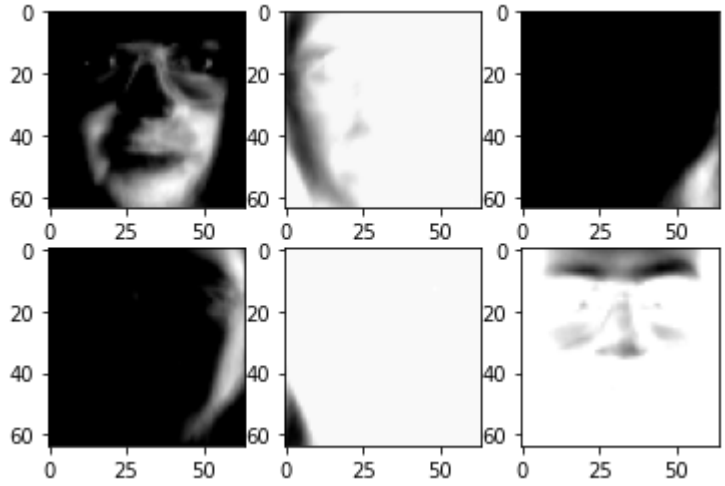
- PCA의 주요 단점 중 하나는 주성분들이 보통 모든 입력 변수들의 선형결합으로 나타난다는 점
- 희소 주성분분석(Sparse PCA)는 몇 개 변수들만의 선형결합으로 주성분을 나타냄으로써 이러한 단점을 극복

```
model = SparsePCA(n_components=2, random_state=0)
```

셀 삭제를 실행취소하려면 Ctrl+M Z 또는 수정 메뉴의 실행취소 옵션을 사용하세요.



```
model = SparsePCA(n_components=2*3, random_state=0)
model.fit(faces)
faces_components = model.components_
show_faces(faces_components)
```



Truncated Singular Value Decomposition (Truncated SVD)

- PCA는 정방행렬에 대해서만 행렬 분해 가능
- SVD는 정방행렬 뿐만 아니라 행과 열이 다른 행렬도 분해 가능
- PCA는 밀집 행렬(Dense Matrix)에 대한 변환만 가능하지만, SVD는 희소 행렬(Sparse Matrix)에 대한 변환도 가능
- 전체 행렬 크기에 대해 Full SVD를 사용하는 경우는 적음
- 특이값이 0인 부분을 모두 제거하고 차원을 줄인 Truncated SVD를 주로 사용

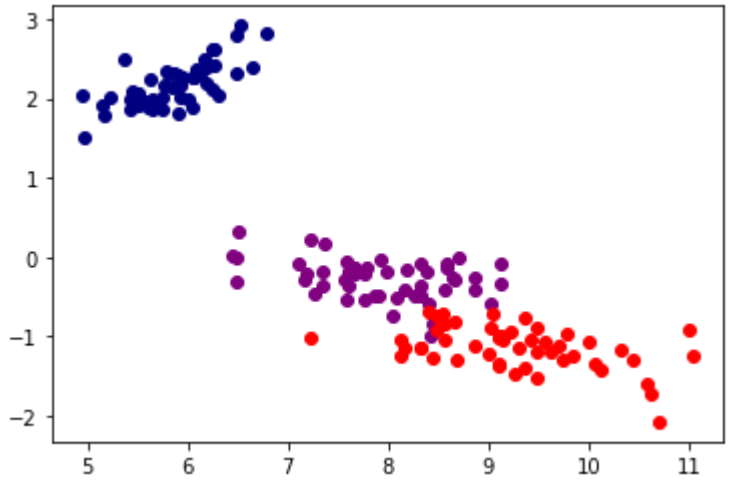
```
X => A * B * C
A => 좌 특이 행렬
B => 특이값 행렬 (대각 행렬)
C => 우 특이 행렬
```



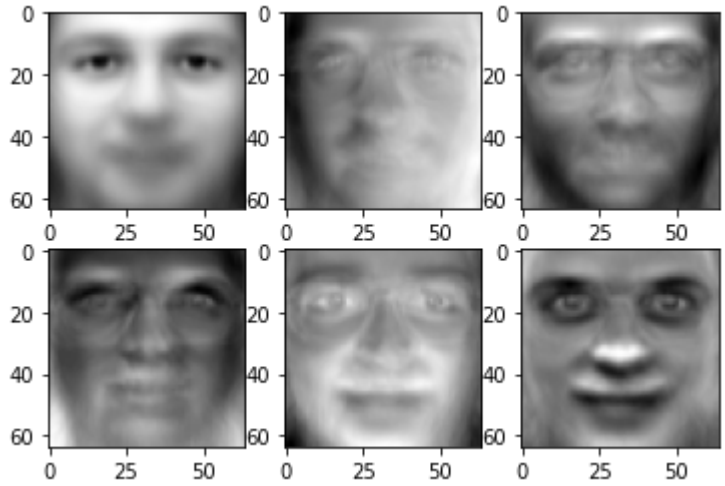
```
B => (N, M)
C => (M, M)

[[2, 1, 0, 0],
 [0, 5, 0, 0],
 [0, 0, 4, 4],
 [0, 0, 0, 3]]

A => (N, D)
B => (D, D)
C => (D, M)
```

```
model = TruncatedSVD(n_components=2, random_state=0)
model.fit(iris)
transformed_iris = model.transform(iris)
```

```
plot_iris(transformed_iris, labels)
```



```
model = TruncatedSVD(n_components=2*3, random_state=0)
model.fit(faces)
faces_components = model.components_
show_faces(faces_components)
```

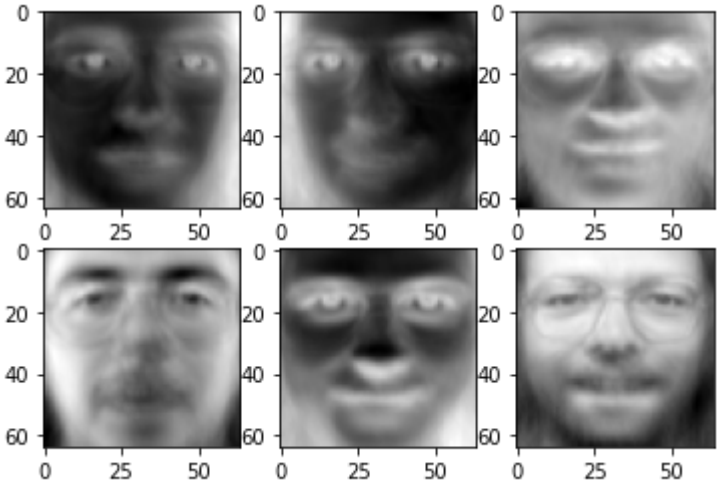


셀 삭제를 실행취소하려면 Ctrl+M Z 또는 수정 메뉴의 실행취소 옵션을 사용하세요.

- Sparse code를 사용하여 데이터를 가장 잘 나타내는 사전 찾기
- Sparse coding은 overcomplete 기저벡터(basis vector)를 기반으로 데이터를 효율적으로 표현하기 위해 개발
- 기저 벡터는 벡터 공간에 속하는 벡터의 집합이 선형 독립이고, 다른 모든 벡터 공간의 벡터들이 그 벡터 집합의 선형 조합으로 나타남

```
model = DictionaryLearning(n_components=2, random_state=0)
model.fit(iris)
transformed_iris = model.transform(iris)
plot_iris(transformed_iris, labels)
```


```
model = DictionaryLearning(n_components=2*3, random_state=0)
model.fit(faces)
faces_components = model.components_
show_faces(faces_components)
```



Factor Analysis

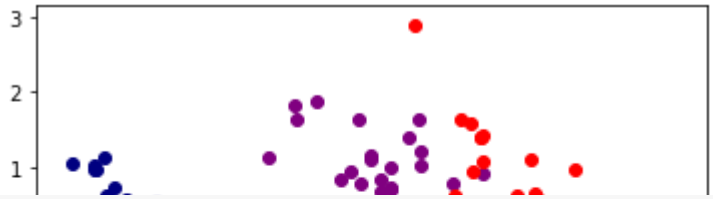
- 요인 분석(Factor Analysis)은 변수들 간의 상관관계를 고려하여 저변에 내재된 개념인 요인들을 추출해내는 분석방법

셀 삭제를 실행취소하려면 Ctrl+M Z 또는 수정 메뉴의 실행취소 옵션을 사용하세요.

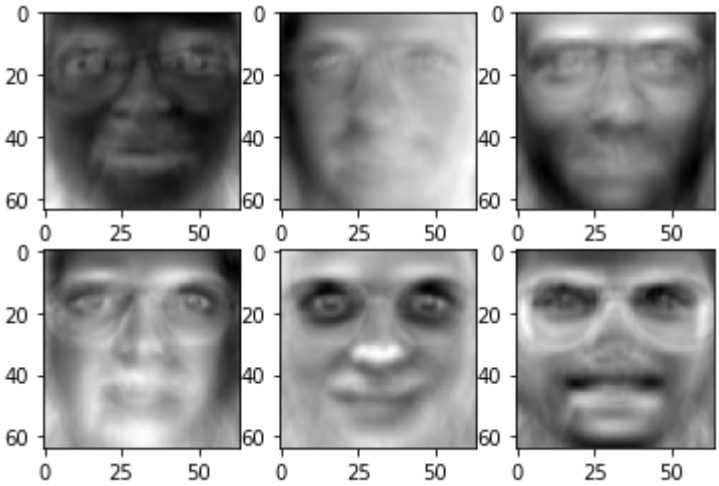
어주는 방법

려

```
model = FactorAnalysis(n_components=2, random_state=0)
model.fit(iris)
transformed_iris = model.transform(iris)
plot_iris(transformed_iris, labels)
```



```
model = FactorAnalysis(n_components=2*3, random_state=0)
model.fit(faces)
faces_components = model.components_
show_faces(faces_components)
```

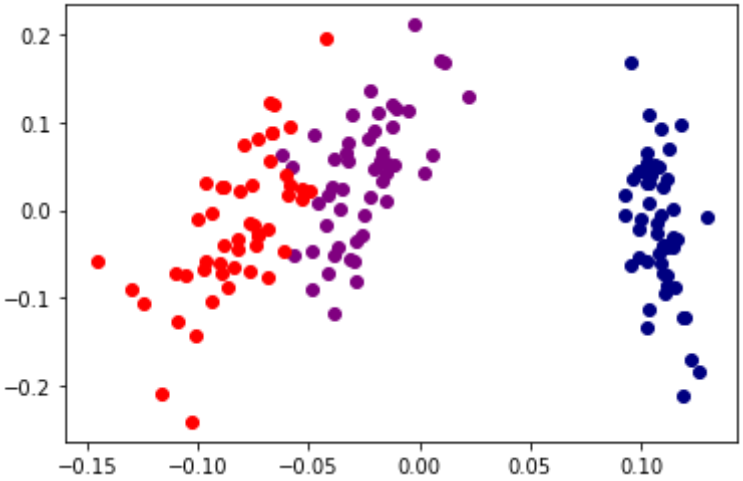


Independent Component Analysis(ICA)

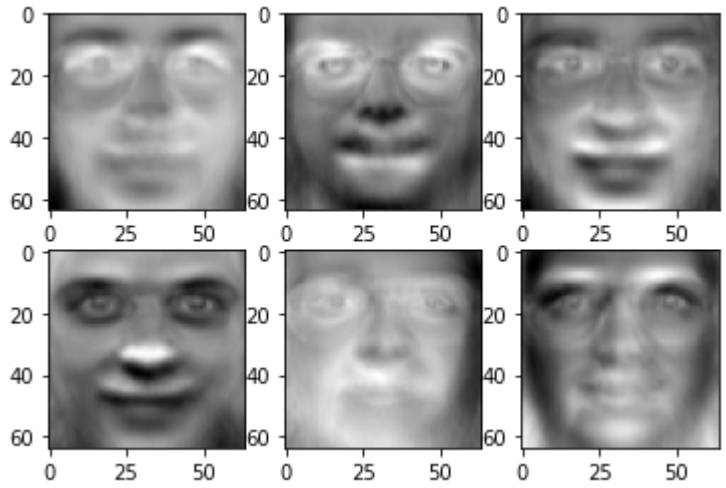
- 독립 성분 분석(Independent Component Analysis, ICA)은 다변량의 신호를 통계적으로 독립적인 하부 성분으로 분리하는 계산 방법
- ICA는 주성분을 이용하는 점은 PCA와 유사하지만, 데이터를 가장 잘 설명하는 축을 찾는 PCA와 달리 가장 독립적인 축, 독립성이 최대가 되는 벡터를 찾음

```
model = FastICA(n_components=2, random_state=0)
```

셀 삭제를 실행취소하려면 Ctrl+M Z 또는 수정 메뉴의 실행취소 옵션을 사용하세요.



```
model = FastICA(n_components=2*3, random_state=0)
model.fit(faces)
faces_components = model.components_
show_faces(faces_components)
```



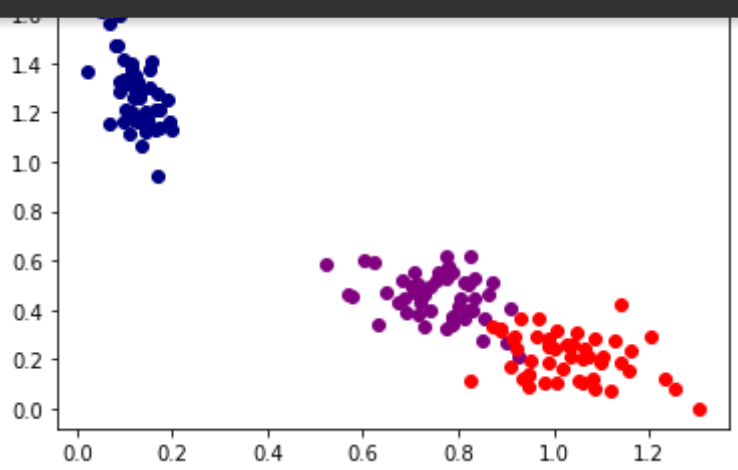
Non-negative Matrix Factorization

- 음수 미포함 행렬 분해(Non-negative matrix factorization, NMF)는 음수를 포함하지 않은 행렬 V를 음수를 포함하지 않은 행렬 W와 H의 곱으로 분해하는 알고리즘

```
X => W * H

X => (N, M)
W => (N, D)
H => (D, M)
```

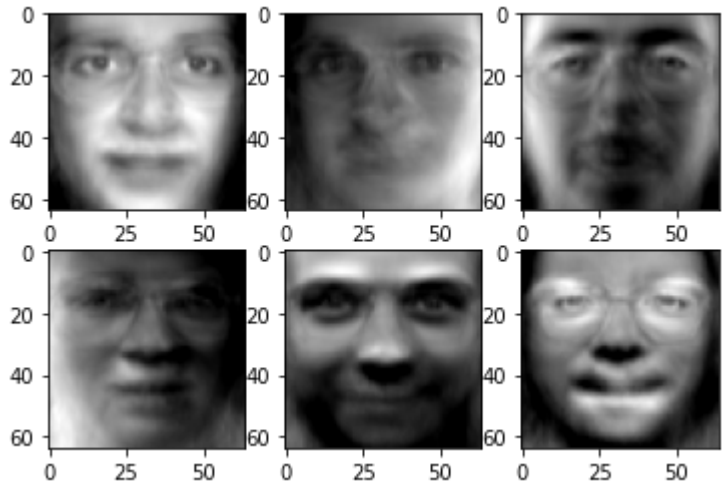
```
model = NMF(n_components=2, random_state=0)
model.fit(iris)
transformed_iris = model.transform(iris)
```

셀 삭제를 실행취소하려면 Ctrl+M Z 또는 수정 메뉴의 실행취소 옵션을 사용하세요.



```
model = NMF(n_components=2*3, random_state=0)
model.fit(faces)
```

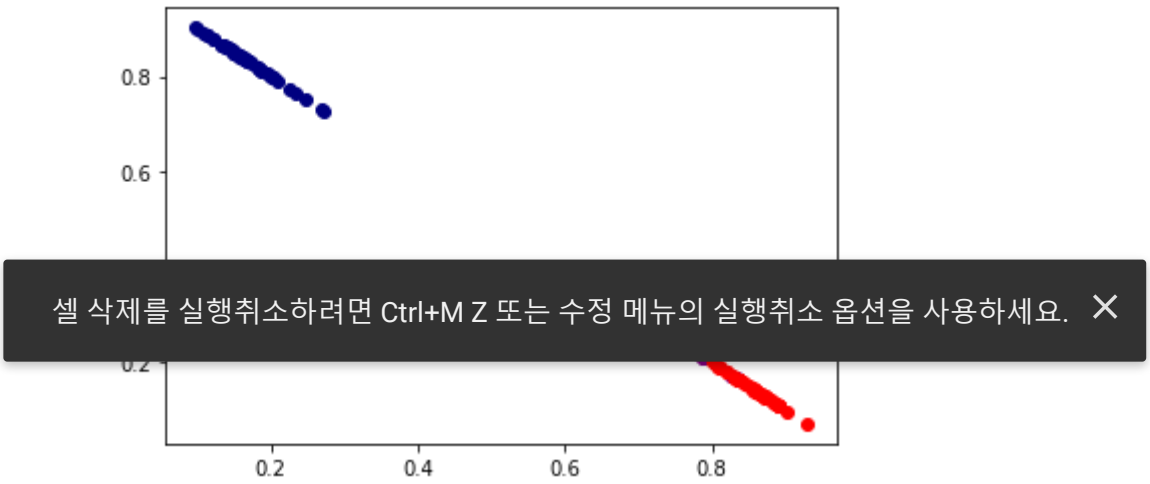
```
faces_components = model.components_
show_faces(faces_components)
```



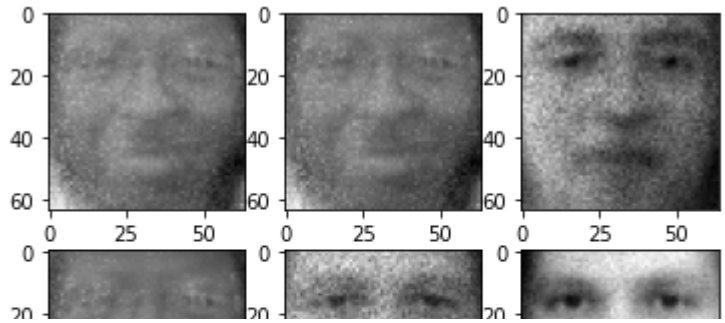
Latent Dirichlet Allocation (LDA)

- 잠재 디리클레 할당은 이산 자료들에 대한 확률적 생성 모형
- 디리클레 분포에 따라 잠재적인 의미 구조를 파악

```
model = LatentDirichletAllocation(n_components=2, random_state=0)
model.fit(iris)
transformed_iris = model.transform(iris)
plot_iris(transformed_iris, labels)
```



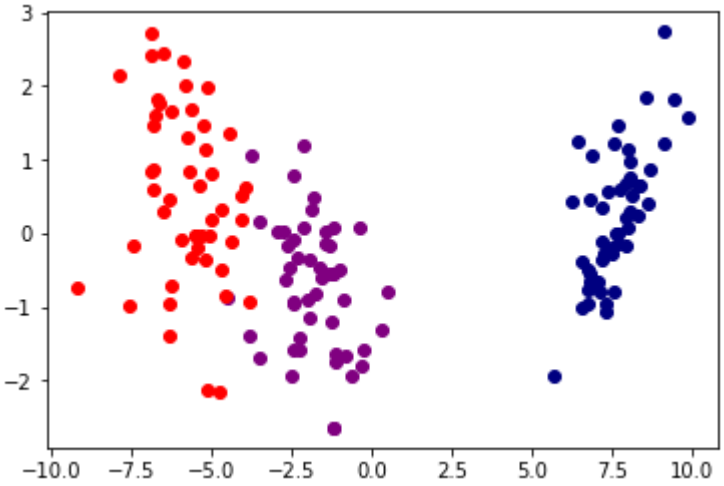

```
model = LatentDirichletAllocation(n_components=2*3, random_state=0)
model.fit(faces)
faces_components = model.components_
show_faces(faces_components)
```



Linear Discriminant Analysis (LDA)

- LDA는 PCA와 유사하게 입력 데이터 세트를 저차원 공간에 투영해 차원을 축소
- LDA는 지도학습 분류에서 사용하기 쉽도록 개별 클래스르 분별할 수 있는 기준을 최대한 유지하면서 차원 축소

```
model = LinearDiscriminantAnalysis(n_components=2)
model.fit(iris, labels)
transformed_iris = model.transform(iris)
plot_iris(transformed_iris, labels)
```



셀 삭제를 실행취소하려면 Ctrl+M Z 또는 수정 메뉴의 실행취소 옵션을 사용하세요.

- 행렬 분해를 통해 압축된 데이터를 사용해 학습

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_digits
from sklearn.model_selection import cross_val_score
```

```
def min_max_scale(x):
  min_value, max_value = np.min(x, 0), np.max(x, 0)
  x = (x - min_value) / (max_value - min_value)
  return x
```

```
def plot_digits(digits, labels):
  digits = min_max_scale(digits)
  ax = plt.subplot(111, projection='3d')
  for i in range(digits.shape[0]):
    ax.text(digits[i, 0], digits[i, 1], digits[i, 2],
            str(labels[i]), color=plt.cm.Set1(labels[i] / 10.),
            fontdict={'weight': 'bold', 'size': 9})
  ax.view_init(4, -72)
```

```
digits = load_digits()
```

```
nmf = NMF(n_components=3)
nmf.fit(digits.data)
decomposed_digits = nmf.transform(digits.data)
```

```
print(digits.data.shape)
print(decomposed_digits.shape)
print(decomposed_digits)
```

```
(1797, 64)
(1797, 3)
[[0.48392621 0.         1.24523913]
 [0.5829615  1.46767565 0.07150889]
 [0.61515882 1.10963211 0.387782  ]
 ...
 [0.55272665 1.26056523 0.72094739]
 [0.7872562  0.27898731 1.04952029]
 [0.78507412 0.67250886 0.92677983]]
```

```
plt.figure(figsize=(20, 10))
plot_digits(decomposed_digits, digits.target)
```

KNN

```
knn = KNeighborsClassifier()
```

```
score = cross_val_score(
    estimator=knn,
    X=digits.data, y=digits.target,
    cv=5
)
score
```

```
array([0.94722222, 0.95555556, 0.96657382, 0.98050139, 0.9637883 ])
```

```
print('mean cross val score: {} (+/- {})'.format(score.mean(), score.std()))
```

```
mean cross val score: 0.9627282575054161 (+/- 0.011168537355954218)
```

```
score = cross_val_score(
    estimator=knn,
    X=decomposed_digits, y=digits.target,
    cv=5
)
score
```

```
array([0.54722222, 0.58055556, 0.64066852, 0.59610028, 0.56267409])
```

```
print('mean cross val score: {} (+/- {})'.format(score.mean(), score.std()))
```

```
mean cross val score: 0.5854441349427422 (+/- 0.03214521445075084)
```

셀 삭제를 실행취소하려면 Ctrl+M Z 또는 수정 메뉴의 실행취소 옵션을 사용하세요.

```
svm = SVC()
```

```
score = cross_val_score(
    estimator=svm,
    X=digits.data, y=digits.target,
    cv=5
)
score
```

```
array([0.96111111, 0.94444444, 0.98328691, 0.98885794, 0.93871866])
```

```
print('mean cross val score: {} (+/- {})'.format(score.mean(), score.std()))
```

```
mean cross val score: 0.9632838130609718 (+/- 0.02008605863225686)
```

```
score = cross_val_score(
    estimator=svm,
    X=decomposed_digits, y=digits.target,
    cv=5
)
score
```

```
array([0.61388889, 0.62222222, 0.66016713, 0.60167131, 0.59888579])
```

```
print('mean cross val score: {} (+/- {})'.format(score.mean(), score.std()))
```

```
mean cross val score: 0.6193670690188796 (+/- 0.022070024720937543)
```

Decision Tree

```
decision_tree = DecisionTreeClassifier()
```

```
score = cross_val_score(
    estimator=decision_tree,
    X=digits.data, y=digits.target,
    cv=5
)
score
```

```
array([0.775     , 0.725     , 0.78551532, 0.83844011, 0.81615599])
```

```
print('mean cross val score: {} (+/- {})'.format(score.mean(), score.std()))
```

```
mean cross val score: 0.7880222841225628 (+/- 0.03868109537958548)
```

셀 삭제를 실행취소하려면 Ctrl+M Z 또는 수정 메뉴의 실행취소 옵션을 사용하세요.

```
    X=decomposed_digits, y=digits.target,
    cv=5
)
score
```

```
array([0.58888889, 0.50277778, 0.57381616, 0.58495822, 0.50417827])
```

```
print('mean cross val score: {} (+/- {})'.format(score.mean(), score.std()))
```

```
mean cross val score: 0.5509238625812443 (+/- 0.03905619860692382)
```

Random Forest

```
random_forest = RandomForestClassifier()
```

```
score = cross_val_score(
    estimator=random_forest,
    X=digits.data, y=digits.target,
    cv=5
)
score
```

```
array([0.93055556, 0.90833333, 0.96100279, 0.96100279, 0.93593315])
```

```
print('mean cross val score: {} (+/- {})'.format(score.mean(), score.std()))
```

```
mean cross val score: 0.9393655215103683 (+/- 0.01994359937854058)
```

```
score = cross_val_score(
    estimator=random_forest,
    X=decomposed_digits, y=digits.target,
    cv=5
)
score
```

```
array([0.57777778, 0.59166667, 0.66852368, 0.61559889, 0.545961  ])
```

```
print('mean cross val score: {} (+/- {})'.format(score.mean(), score.std()))
```

```
mean cross val score: 0.5999056019808109 (+/- 0.04104207548638512)
```

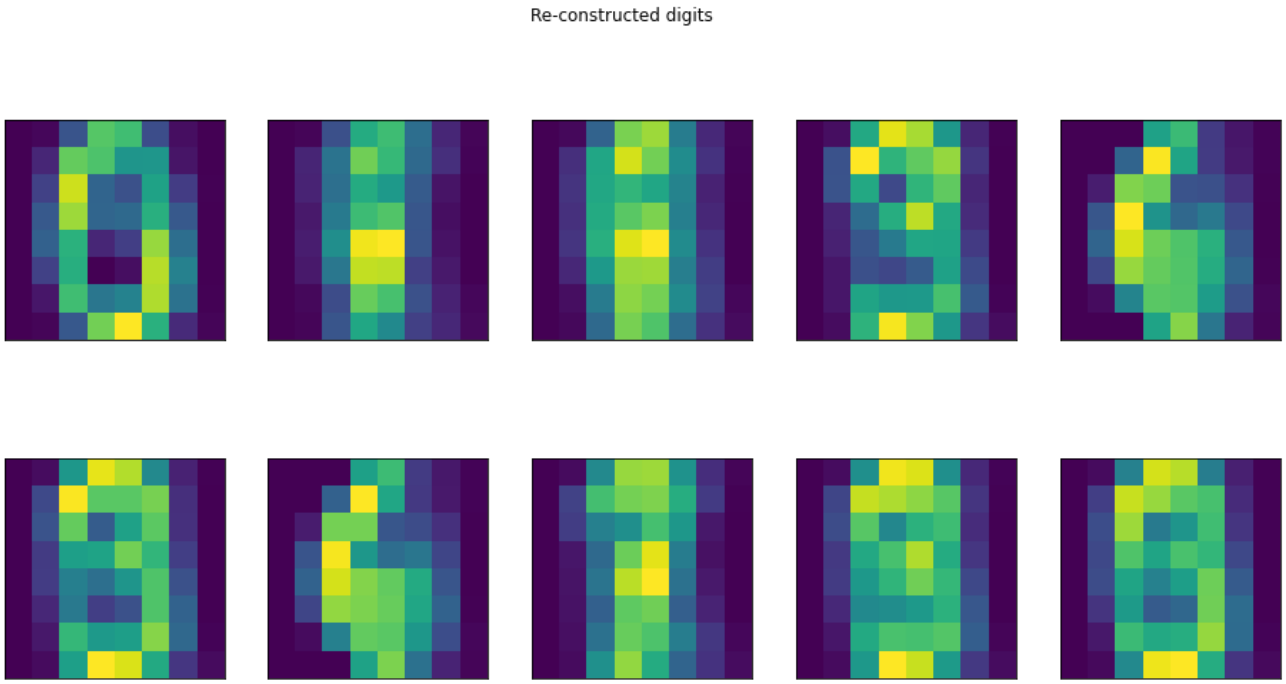
복원된 표현을 사용한 학습

- 분해 후 복원된 행렬을 사용해 학습

셀 삭제를 실행취소하려면 Ctrl+M Z 또는 수정 메뉴의 실행취소 옵션을 사용하세요. ×

```
(1797, 64)
```

```
plt.figure(figsize=(16, 8))
plt.suptitle('Re-constructed digits')
for i in range(10):
  plt.subplot(2, 5, i+1)
  plt.xticks([])
  plt.yticks([])
  plt.imshow(reconstructed_digits[i].reshape(8, 8))
```



KNN

```
score = cross_val_score(
    estimator=knn,
    X=reconstructed_digits, y=digits.target,
    cv=5
)
score
```

```
array([0.54166667, 0.59444444, 0.66295265, 0.57660167, 0.57381616])
```

```
print('mean cross val score: {} (+/- {})'.format(score.mean(), score.std()))
```

셀 삭제를 실행취소하려면 Ctrl+M Z 또는 수정 메뉴의 실행취소 옵션을 사용하세요.

SVM

```
score = cross_val_score(
    estimator=svm,
    X=reconstructed_digits, y=digits.target,
    cv=5
)
score
```

```
array([0.62777778, 0.60555556, 0.66016713, 0.61002786, 0.5821727 ])
```

```
print('mean cross val score: {} (+/- {})'.format(score.mean(), score.std()))
```

```
mean cross val score: 0.6171402042711235 (+/- 0.025969174809053776)
```

Decision Tree

```
score = cross_val_score(
    estimator=decision_tree,
    X=reconstructed_digits, y=digits.target,
    cv=5
)
score
```

```
array([0.57222222, 0.52777778, 0.55988858, 0.55988858, 0.54038997])
```

```
print('mean cross val score: {} (+/- {})'.format(score.mean(), score.std()))
```

```
mean cross val score: 0.552033426183844 (+/- 0.015842353680764393)
```

Random Forest

```
score = cross_val_score(
    estimator=random_forest,
    X=reconstructed_digits, y=digits.target,
    cv=5
)
score
```

```
array([0.58055556, 0.58888889, 0.63788301, 0.61002786, 0.57103064])
```

```
print('mean cross val score: {} (+/- {})'.format(score.mean(), score.std()))
```

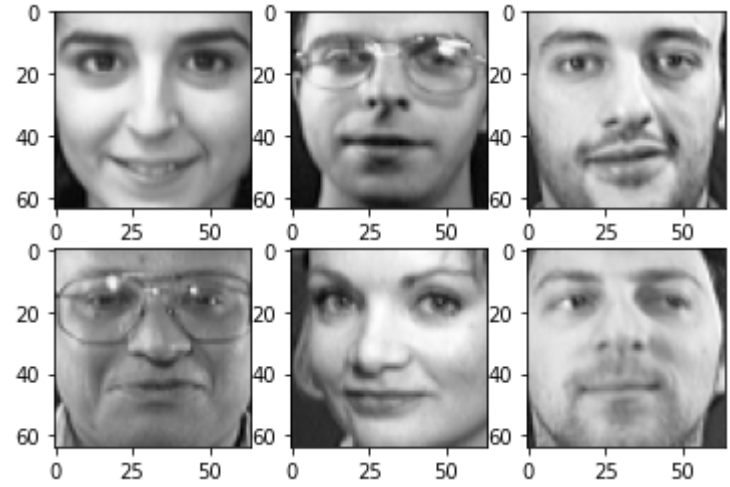
```
mean cross val score: 0.5976771897245434 (+/- 0.023872452547610862)
```

셀 삭제를 실행취소하려면 Ctrl+M Z 또는 수정 메뉴의 실행취소 옵션을 사용하세요.

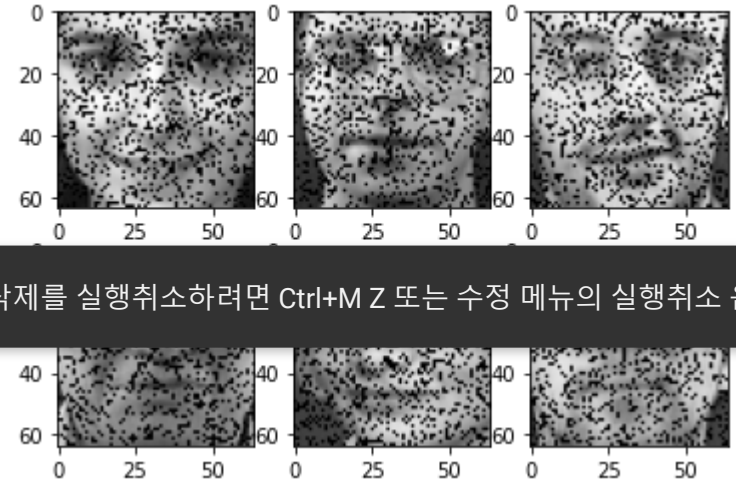
```
train_faces, test_faces = train_test_split(faces, test_size=0.1)
```

```
show_faces(train_faces)
```

```
show_faces(test_faces)
```



```
damaged_faces = []
for face in test_faces:
  idx = np.random.choice(range(64 * 64), size=1024)
  damaged_face = face.copy()
  damaged_face[idx] = 0.
  damaged_faces.append(damaged_face)
show_faces(damaged_faces)
```



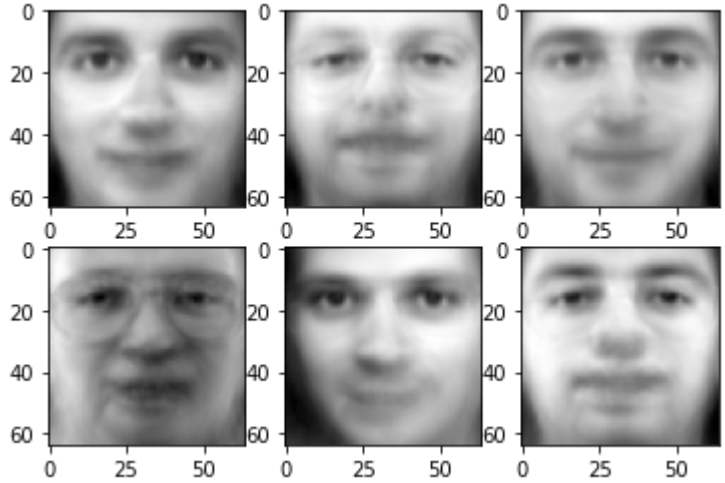
셀 삭제를 실행취소하려면 Ctrl+M Z 또는 수정 메뉴의 실행취소 옵션을 사용하세요. ×

```
nmf = NMF(n_components=10)
nmf.fit(train_faces)
```

```
NMF(alpha=0.0, beta_loss='frobenius', init=None, l1_ratio=0.0, max_iter=200,
    n_components=10, random_state=None, shuffle=False, solver='cd', tol=0.0001,
    verbose=0)
```

```
matrix1 = nmf.transform(damaged_faces)
matrix2 = nmf.components_
show_faces(matrix1 @ matrix2)
```

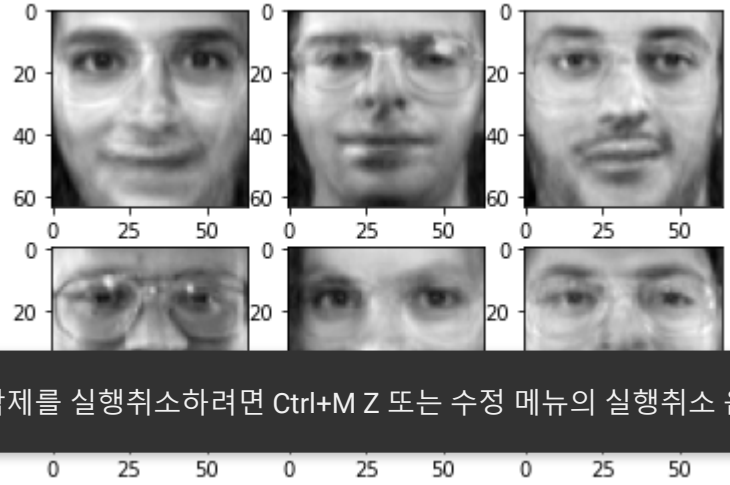
```
show_faces(matrix1 @ matrix2)
```



```
nmf = NMF(n_components=100)
nmf.fit(train_faces)
```

```
NMF(alpha=0.0, beta_loss='frobenius', init=None, l1_ratio=0.0, max_iter=200,
    n_components=100, random_state=None, shuffle=False, solver='cd', tol=0.0001,
    verbose=0)
```

```
matrix1 = nmf.transform(damaged_faces)
matrix2 = nmf.components_
show_faces(matrix1 @ matrix2)
```

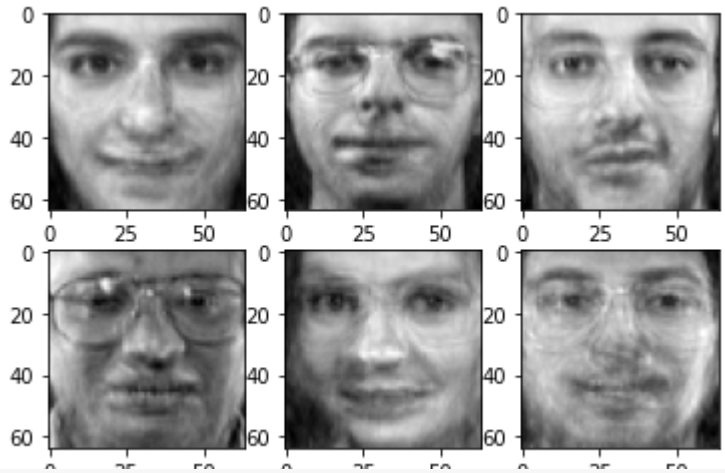


셀 삭제를 실행취소하려면 Ctrl+M Z 또는 수정 메뉴의 실행취소 옵션을 사용하세요.

```
nmf = NMF(n_components=300)
nmf.fit(train_faces)
```

```
NMF(alpha=0.0, beta_loss='frobenius', init=None, l1_ratio=0.0, max_iter=200,
    n_components=300, random_state=None, shuffle=False, solver='cd', tol=0.0001,
    verbose=0)
```

```
matrix1 = nmf.transform(damaged_faces)
matrix2 = nmf.components_
show_faces(matrix1 @ matrix2)
```



```
model = NMF(n_components=10)
```

```
{'alpha': 0.0,
 'beta_loss': 'frobenius',
 'init': None,
 'l1_ratio': 0.0,
 'max_iter': 200,
 'n_components': 10,
 'random_state': None,
 'shuffle': False,
 'solver': 'cd',
 'tol': 0.0001,
 'verbose': 0}
```