1.1 a) Eigengesichter

- Plotte die ersten 20 Hauptachsen (Eigenvektoren der Kovarianzmatrix).

[11]:
```
from sklearn.decomposition import PCA
from sklearn import preprocessing   # To try different scaling methods
import matplotlib.pyplot as plt
import pandas as pd
```

"Mean Face":

[12]:
```
plot_faces(faces.mean(axis=0))
```

[12]:
```
(<Figure size 1200x300 with 4 Axes>,
 array([[<Axes: >, <Axes: >, <Axes: >, <Axes: >]], dtype=object))
```

Zentrierere die Beobachtungen (vielleicht macht das das PCA-Model automatisch?)

[13]:
```
faces_centered = faces - faces.mean(axis=0)
faces_centered -= faces_centered.mean(axis=1).reshape(len(faces), -1)
```

[14]:
```
# Wähle die ersten 20 Hauptachsen
n_comp = 20
pca_model = PCA(n_components=n_comp)
transformed_faces = pca_model.fit_transform(faces)
pca_model.explained_variance_.cumsum()  # Es wird ~60% Varianz erklärt

# Macht der zentrierte Array einen Unterschied?
pca_model_centered = PCA(n_components=n_comp)
transformed_faces_centered = pca_model_centered.fit_transform(faces_centered)

# Und wie wäre es, wenn wir die Daten selbst skalieren und ohne transformation␣
 ↪fitten?
scaled_faces = preprocessing.scale(faces)
```

```
pca_model_prescaled = PCA(n_components=n_comp)
pca_model_prescaled.fit(scaled_faces)

# Mit StandardScaler
std_scaled_faces = StandardScaler().fit_transform(faces)
pca_model_std_prescaled = PCA(n_components=n_comp)
pca_model_std_prescaled.fit(std_scaled_faces)
```

```
/Users/ubd/miniforge3/envs/data_science/lib/python3.9/site-
packages/sklearn/preprocessing/_data.py:240: UserWarning: Numerical issues were
encountered when centering the data and might not be solved. Dataset may contain
too large values. You may need to prescale your features.
  warnings.warn(
/Users/ubd/miniforge3/envs/data_science/lib/python3.9/site-
packages/sklearn/preprocessing/_data.py:259: UserWarning: Numerical issues were
encountered when scaling the data and might not be solved. The standard
deviation of the data is probably very close to 0.
  warnings.warn(
```

[14]:
```
PCA(n_components=20)
```

Visualisierungen der ersten 20 Hauptachsen der 3 Modellen:

[42]:
```
plot_faces(pca_model.components_, cols=5, title="Basic Model")
plot_faces(pca_model_centered.components_, cols=5, title="Pre-centered Model")
plot_faces(pca_model_prescaled.components_, cols=5, title="Pre-scaled Model")
plot_faces(pca_model_std_prescaled.components_, cols=5, title="StandardScaler()␣
 ↪pre-scaled Model")
fig, _ = plot_faces(pca_model.components_, cols=5)
fig.savefig("faces_pca_with_20-main-components.png", dpi=900)
```

- Wie groß sind die zugehörigen Eigenwerte (Varianzen) dieser Eigengesichter?

Also, wir haben die obigen Eigenvektoren von der Kovarianzmatrix $X^T X$. Sei v_k die Eigenvektoren und λ_k die Eigenwerte für $k = 1, \ldots,$ Anzahl der Komponenten. Dann;

$$X^T X v_k = \lambda_k v_k \implies \lambda_k = < X^T X v_k, v_k > \text{ mit } ||v_k||^2 = 1$$

[16]:
```
m = faces_centered.shape[0]      # Da wir hier manuell berechenen,
                                 # benutze ich einfach die zentrierten Daten.␣
 ↪Sonst hätte ich wieder zentrieren müssen.
cov_matrix = np.dot(faces_centered.T, faces_centered) / m
for eigenvector in pca_model_centered.components_:
    print(np.dot(eigenvector.T, np.dot(cov_matrix, eigenvector)))
```

```
11.265934
6.3150907
4.4085064
3.4729922
2.5249925
```

```
2.0739794
1.6218457
1.6049399
1.3355033
1.2621926
1.1349277
1.0094956
0.9488584
0.84323895
0.79162353
0.7420134
0.66886693
0.5949468
0.591858
0.56110865
```

Oder wir können mit der folgenden Methoder des PCA-Objekts dasselbe berechnen, weil die Eigenwerte genau der erklärten Varianz auf einem (durch einen?) Eigenvektor.

[17]:
```
pca_model_centered.explained_variance_
```

[17]:
```
array([11.29417   ,  6.3309135 ,  4.419554  ,  3.4816983 ,  2.5313213 ,
        2.079178  ,  1.6259085 ,  1.6089658 ,  1.3388513 ,  1.2653569 ,
        1.1377738 ,  1.0120245 ,  0.95123667,  0.84535146,  0.79360664,
        0.7438736 ,  0.6705439 ,  0.5964383 ,  0.59334135,  0.56251556],
      dtype=float32)
```

- Plotte den Anteil der erklärten Varianz in Abhängigkeit der verwendeten Hauptkomponenten. Dazu kannst du das Attribut `explained_variance_ratio_` verwenden.

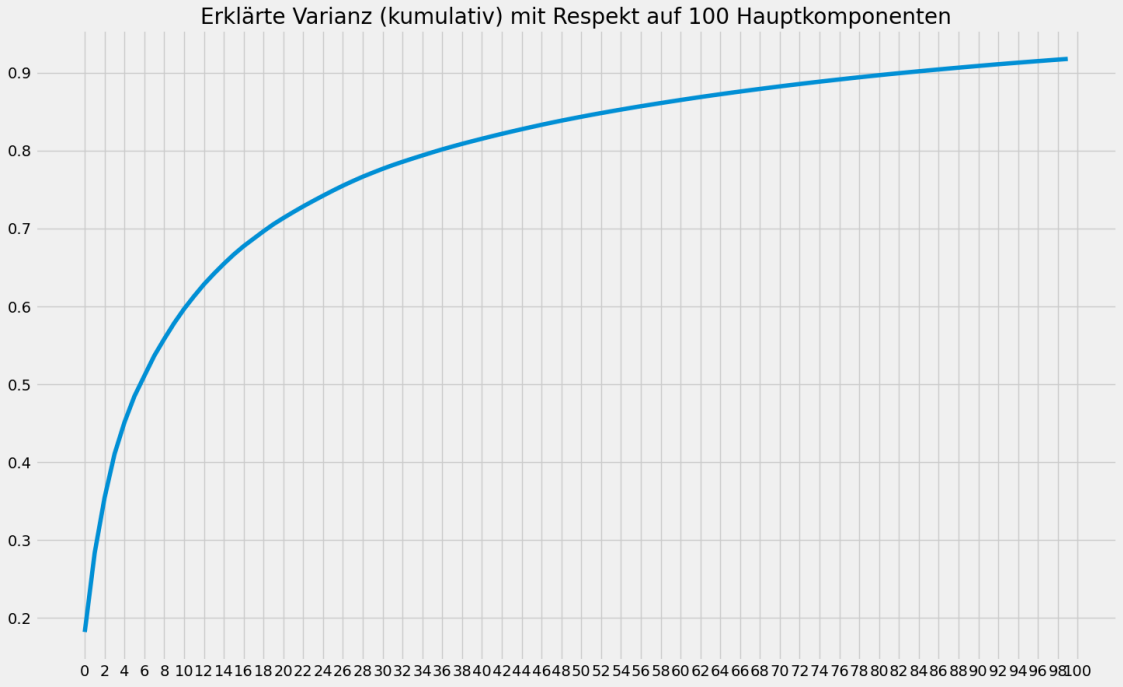
[18]:
```
pca_model_centered.explained_variance_ratio_
```

[18]:
```
array([0.18154666, 0.10176544, 0.07104155, 0.0559661 , 0.04068939,
       0.03342148, 0.02613545, 0.02586311, 0.02152119, 0.02033981,
       0.01828899, 0.01626766, 0.01529053, 0.01358849, 0.01275673,
       0.0119573 , 0.01077857, 0.00958737, 0.00953759, 0.00904208],
      dtype=float32)
```

[19]:
```
plt.figure(figsize=(16, 10))
plt.title('Erklärte Varianz mit Respekt auf die verwendeten Hauptkomponenten')
plt.plot(pca_model_centered.explained_variance_ratio_)
plt.xticks(np.arange(0, 20, step=2))
```

[19]:
```
([<matplotlib.axis.XTick at 0x315478130>,
  <matplotlib.axis.XTick at 0x3154789d0>,
  <matplotlib.axis.XTick at 0x315478550>,
  <matplotlib.axis.XTick at 0x3194a0fa0>,
  <matplotlib.axis.XTick at 0x3194b6a90>,
  <matplotlib.axis.XTick at 0x3194bd580>,
```

```
Text(82, 0, '82'),
Text(84, 0, '84'),
Text(86, 0, '86'),
Text(88, 0, '88'),
Text(90, 0, '90'),
Text(92, 0, '92'),
Text(94, 0, '94'),
Text(96, 0, '96'),
Text(98, 0, '98'),
Text(100, 0, '100')])
```



1.2 b) Inverse Transformation

Berechne eine PCA und plotte die Rekonstruktion von 5 Gesichter basierend auf $5, 10, 20, 50, 100, 200, 300$ und $ 400$ Hauptkomponenten. Dazu kannst du die Methode `inverse_transform` benutzen.

[43]:
```
import random
random.seed(42)
rnd_ind = idx[random.sample(range(len(idx)), 5)]

n_comp_list = [5, 10, 20, 50, 100, 200, 300, 400]
for n_comp in n_comp_list:
    pca_temp = PCA(n_components=n_comp)
    faces_transformed_temp = pca_temp.fit_transform(faces)
```

```
    faces_recovered_temp = pca_temp.inverse_transform(faces_transformed_temp)
    plot_faces(faces_recovered_temp[rnd_ind], cols=5, title=f"Rekonstruktion␣
 ↪mit {str(n_comp)} Hauptkomponenten")

pca_temp = PCA(n_components=20)
faces_transformed_temp = pca_temp.fit_transform(faces)
faces_recovered_temp = pca_temp.inverse_transform(faces_transformed_temp)
fig, _ = plot_faces(faces_recovered_temp[rnd_ind], cols=5)
fig.savefig("faces_reconstructed_pca_with_20-main-components.png", dpi=900)
```

1.3 c) Feature Importance

- Berechne die *Gini Feature Importance*. Du kannst `imshow` für die Visualisierung verwenden.

[23]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(faces, labels, test_size=0.
 ↪2, stratify=labels, random_state=42)
```

[ ]:
```
rf_clf = RandomForestClassifier( n_estimators =100 , random_state=42)
rf_clf.fit(X_train, y_train)
feature_imps = []
for name, score in zip(list(range(X_train.shape[1])), rf_clf.
 ↪feature_importances_):
    feature_imps.append(score)
    print(name, score)
```

[25]:
```
feature_imps = np.array(feature_imps)
plt.figure(figsize=(4,4))
plt.imshow(feature_imps.reshape(64,64), cmap="gray")
plt.axis("off")
```

[25]:
```
(-0.5, 63.5, 63.5, -0.5)
```

[40]:
```
def plot_digit(data):
    plt.figure(figsize=(8,8))
    image = data.reshape(64, 64)
    plt.imshow(image, interpolation="nearest", cmap="copper")
    plt.axis("off")

plot_digit(feature_imps)

cbar = plt.colorbar(ticks=[feature_imps.min(), feature_imps.max()])
cbar.ax.set_yticklabels(['Not important', 'Very important'])

plt.savefig("feature_importance.png", dpi=900)
plt.show()
```