@author: Carlos Andrés Tejada Ramírez
Point 4 of Homework 1 from statistics III

Data Exploration AND PCA (Point 3)

Use this data from FACES to do the following tasks

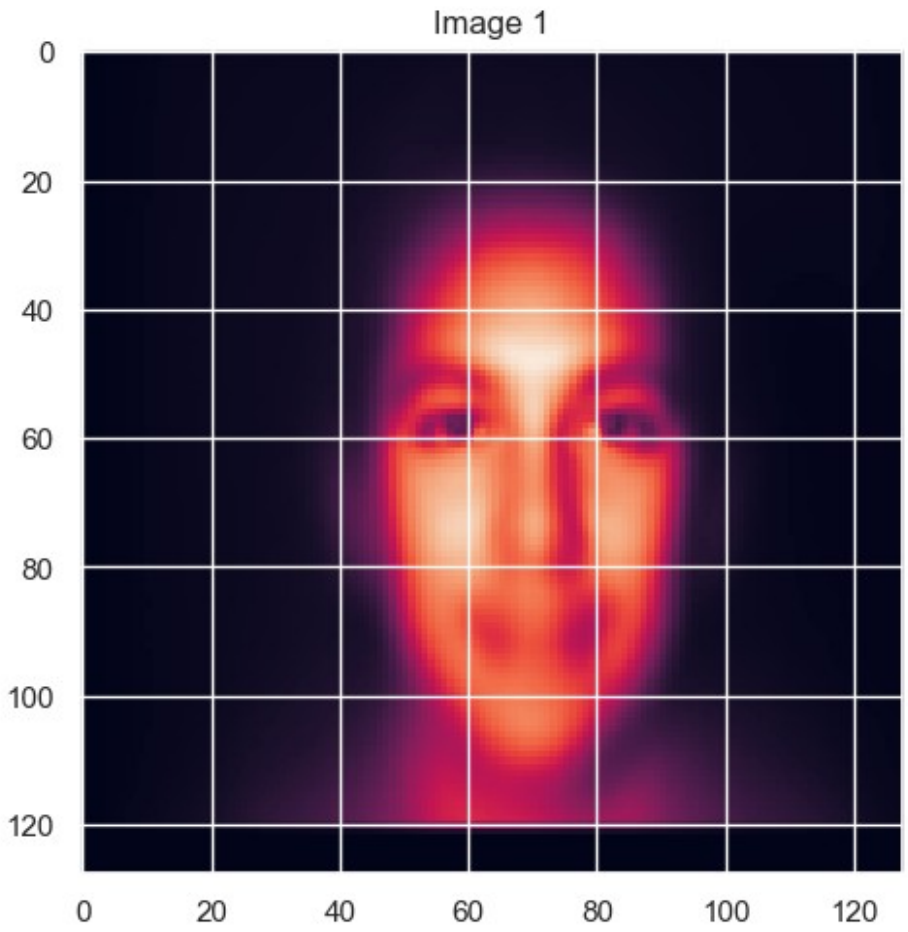
```
import numpy as np
import os
import pandas as pd
import matplotlib.pyplot as plt
import tarfile
import zipfile
from scipy.io import loadmat
from urllib.request import urlretrieve
from os.path import isfile, isdir
import seaborn as sns; sns.set()
%matplotlib inline

#
https://github.com/jdramirez/UCO_ML_AI/blob/master/src/notebook/PCA.ip
ynb

def download_files():
    """
    Este método descarga los archivos de imágenes si no existen
    """
    dest_path = "../Docs/FACES"
    os.makedirs(dest_path, exist_ok=True)  # Crear el directorio si no
existe
    path_tar = os.path.join(dest_path, 'faces.zip')

    if not isfile(path_tar):
        urlretrieve(

'http://courses.media.mit.edu/2002fall/mas622j/proj/faces/rawdata.zip'
,
            path_tar)

    with zipfile.ZipFile(path_tar) as tar:
        tar.extractall(dest_path)
        tar.close()

# Llamar a la función para descargar los archivos
download_files()

directory = '../Docs/FACES/rawdata'
images = []
```

```
for nfile in os.listdir(directory):
    bytesRead = open(os.path.join(directory, nfile), "rb").read()
    imgData = Image.frombytes('L', (128, 128), bytesRead)
    images.append(np.array(imgData).flatten())

images = np.array(images)

def plotImage(data, label="Image 1", ax=None):
    fn_shape = lambda X: X.reshape(128,128)
    fig = None
    if ax is None:
        fig, ax = plt.subplots(1,1, constrained_layout=True)
    ax.imshow(fn_shape(data))
    ax.set_title(label=label)
    return fig,ax

def plot5Images(data, ix_1, ix_2, ix_3, ix_4, ix_5):
    fn_shape = lambda X: X.reshape(128, 128)
    fig, ax = plt.subplots(1, 5, constrained_layout=True)
    ax[0].imshow(fn_shape(data[ix_1]))
    ax[0].set_title(label="Image %s" % ix_1)
    ax[1].imshow(fn_shape(data[ix_2]))
    ax[1].set_title(label="Image %s" % ix_2)
    ax[2].imshow(fn_shape(data[ix_3]))
    ax[2].set_title(label="Image %s" % ix_3)
    ax[3].imshow(fn_shape(data[ix_4]))
    ax[3].set_title(label="Image %s" % ix_4)
    ax[4].imshow(fn_shape(data[ix_5]))
    ax[4].set_title(label="Image %s" % ix_5)
    plt.show()
```

1. Calculate the mean face. What is the face with the average of the pixels and visualize it.

```
averageImage = images.mean(axis = 0)
plotImage(averageImage)
```

```
(<Figure size 640x480 with 1 Axes>, <Axes: title={'center': 'Image
1'}>)
```



Center the data, use PCA. How many components must be used to maintain 90% of the characteristics? Create a table to show the first 5 faces using the mean face + the reconstructed data using the first component, then with 3 components, then with the first 20 components, then with the components that explain 95% of the variance and finally with the number of components that has 99% of the variance. It can conclude from the results?

```
centeredImages = images - averageImage
```

In the link at the beginning of the code, there is a repository where it was shown that 200 eigenvectors contain the necessary amount of information (The most important characteristics are the shapes of the face. Then the eyebrows and the eyes. And finally the details such as the hair , mouth and skin color) to reconstruct the image with quality in this database, therefore, we are going to reuse that data.

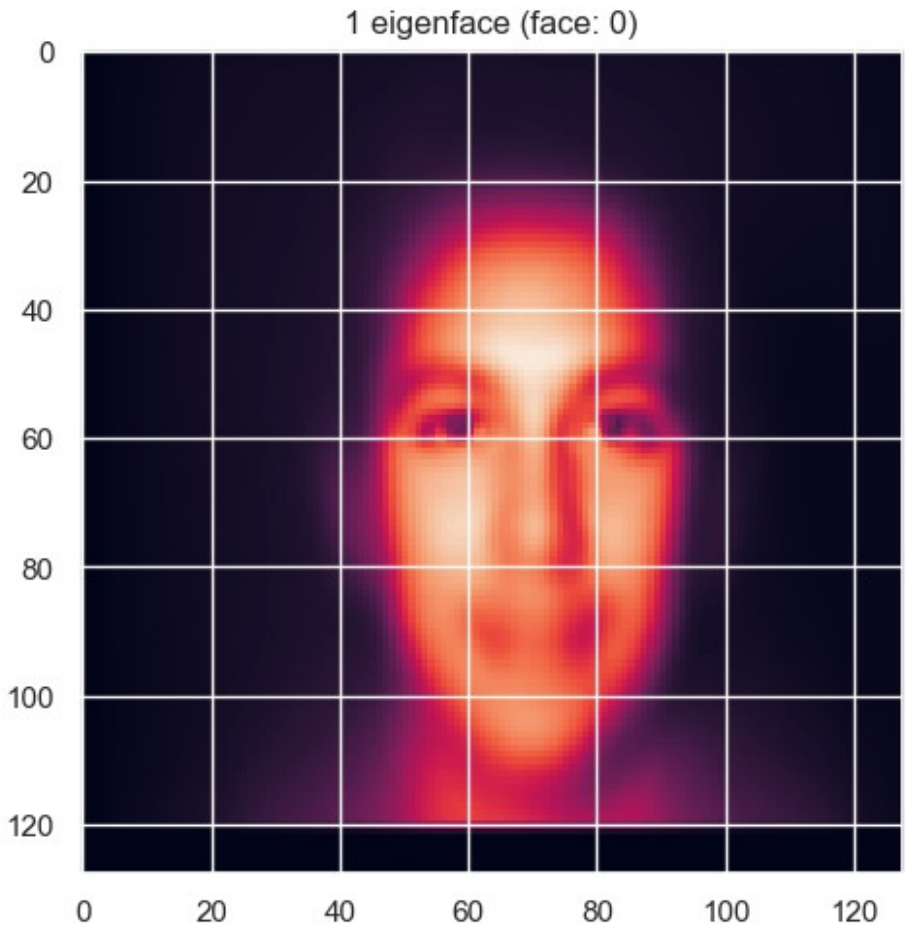
However, if it were necessary to calculate

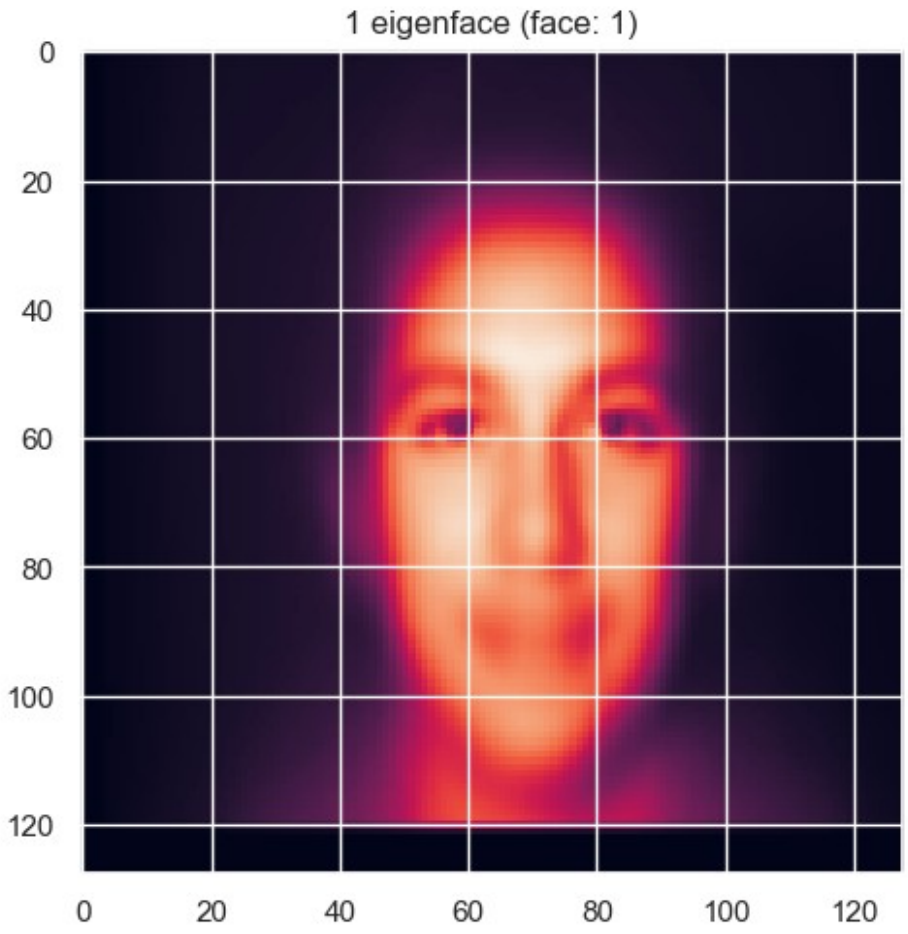
1. **Center the data**
2. **Calculate the covariance matrix**
3. **Calculate the eigenvalues and eigenvectors of the covariance matrix**
4. **Order the eigenvalues from highest to lowest**
5. **Calculate the cumulative proportion of explained variance**
6. **Select the number of main components**
7. **Reduce dimensionality using selected number of principal components**

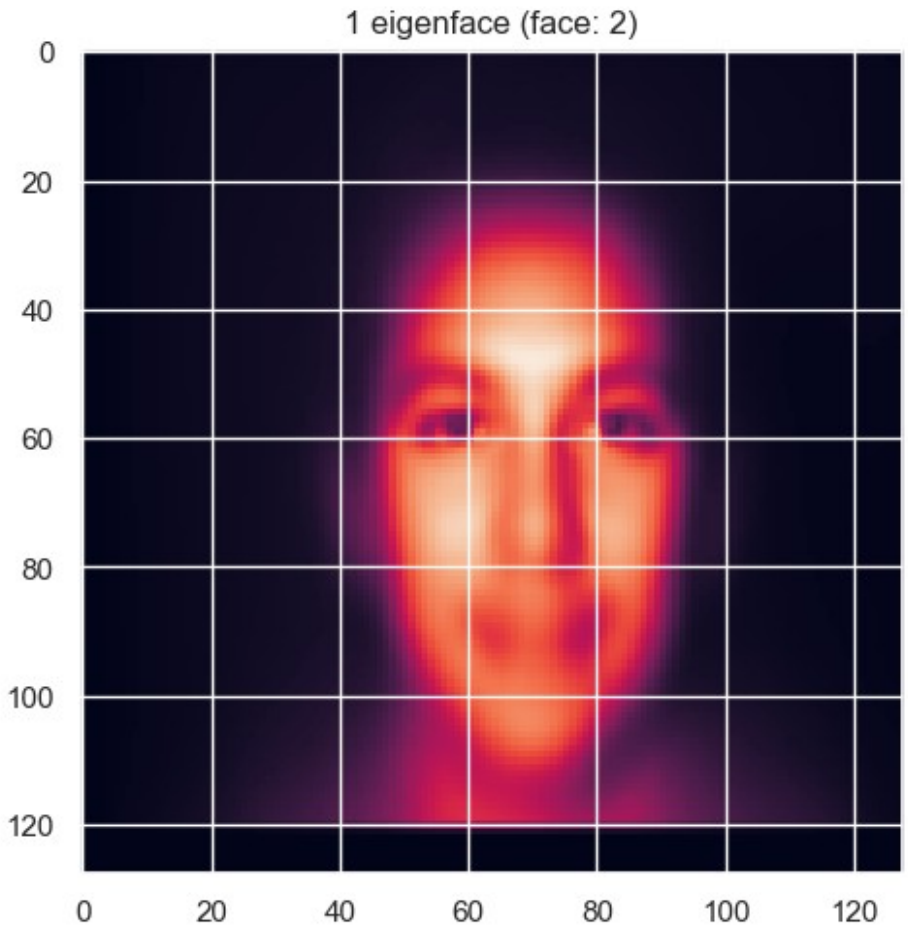
```
pca = PCA(n_components = 200)
imagesReduced = pca.fit_transform(centeredImages)

eigenImageValues = pca.explained_variance_
eigenImagenFaces = pca.components_
```

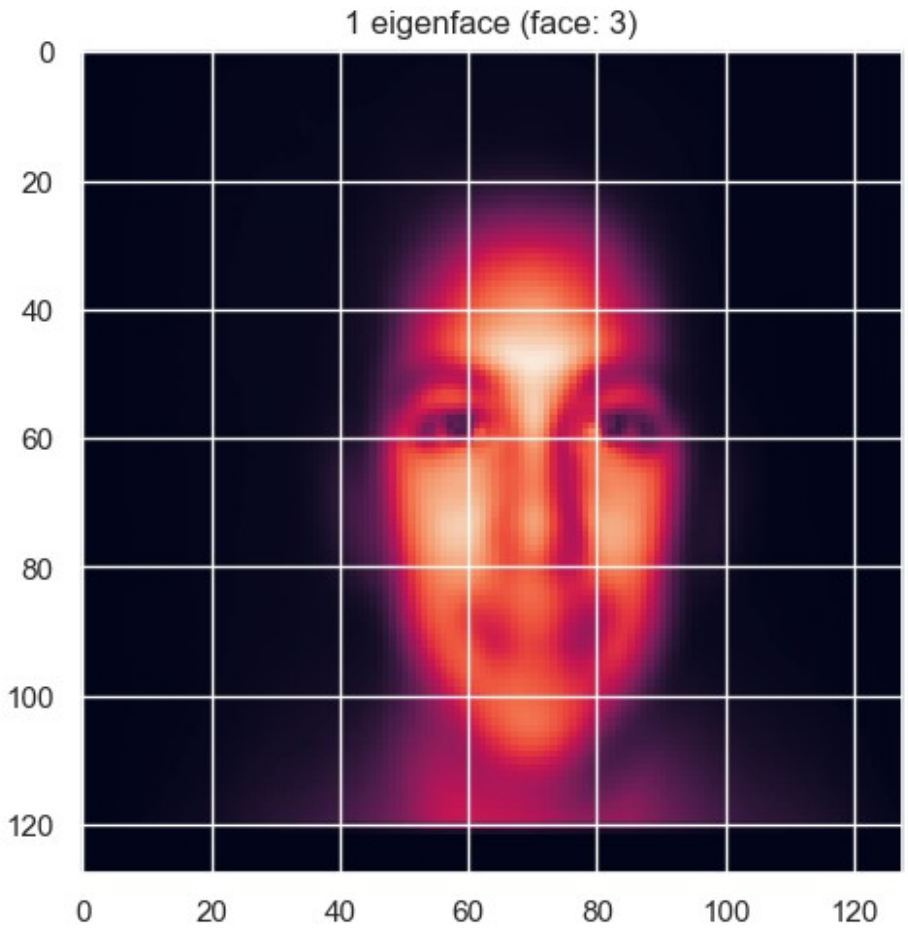
1 Component

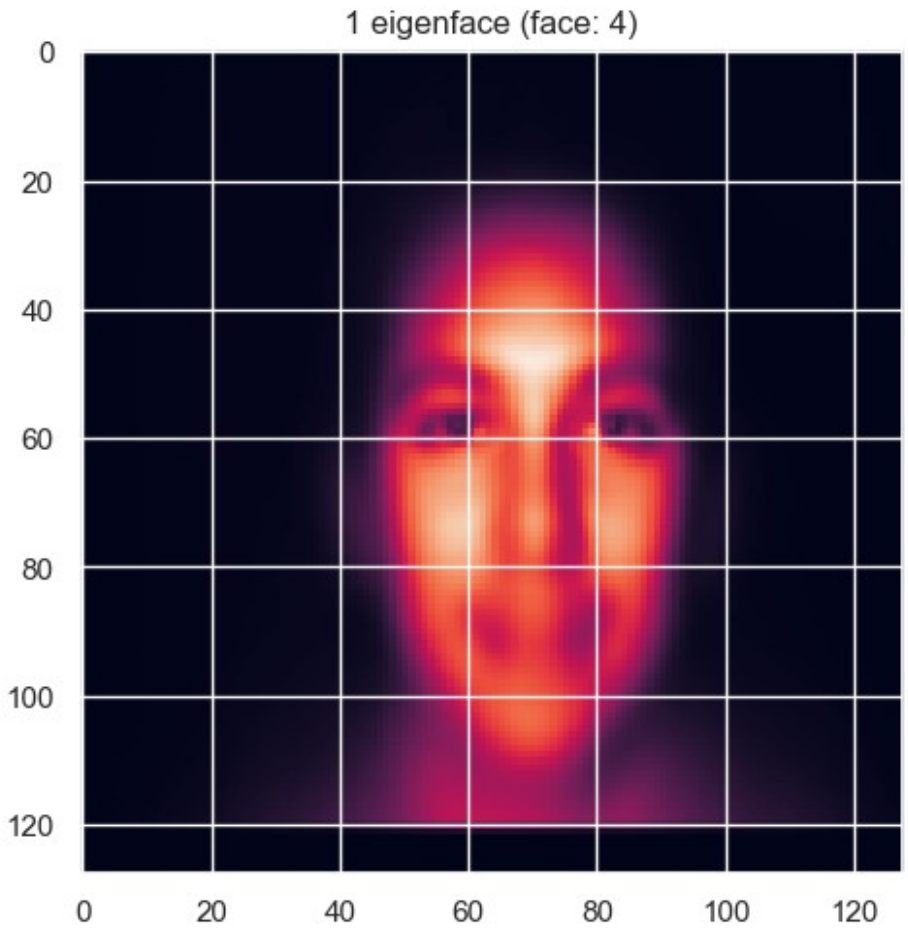
```
for index in range(0, 5):
    face1 = averageImage +
np.sum(np.dot(imagesReduced[index].reshape(1,-1)[:,:1],
eigenImagenFaces[:1,:]), axis = 0)
    plotImage(face1, label= f"1 eigenface (face: {index})")
```


1 eigenface (face: 0)
0
20
40
60
80
100
120
0
20
40
60
80
100
120


1 eigenface (face: 1)
0
20
40
60
80
100
120
0
20
40
60
80
100
120

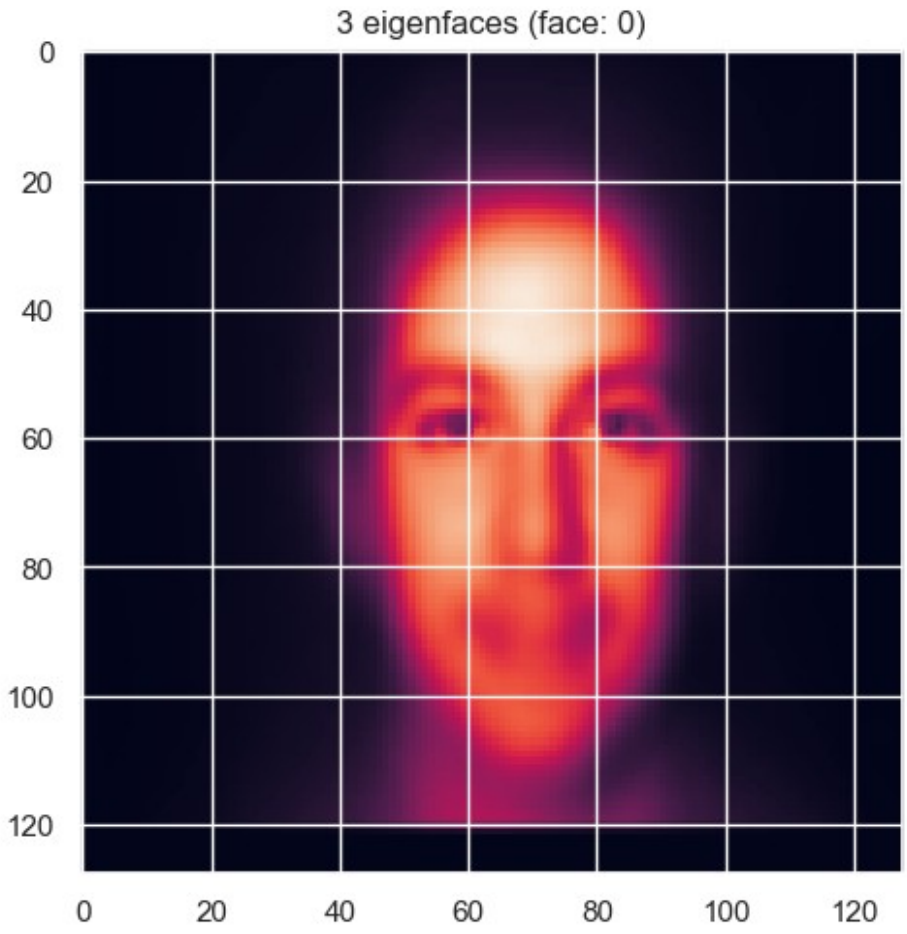

1 eigenface (face: 2)
0
20
40
60
80
100
120
0
20
40
60
80
100
120

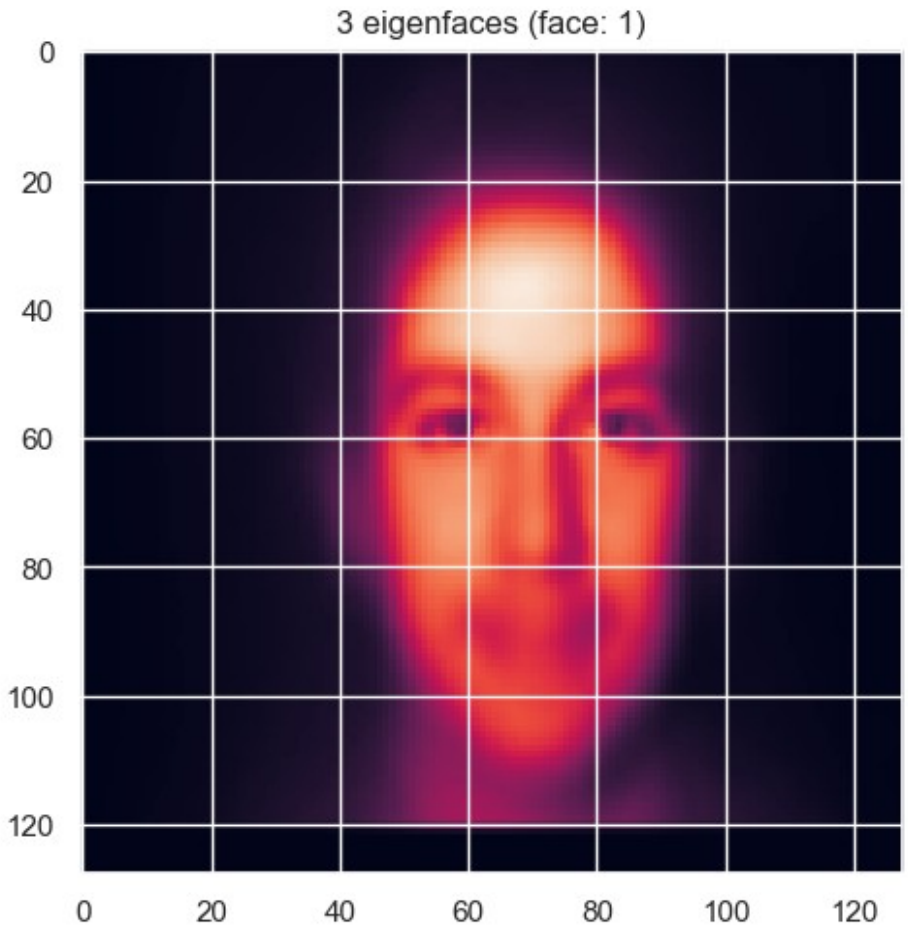

1 eigenface (face: 3)
0
20
40
60
80
100
120
0
20
40
60
80
100
120

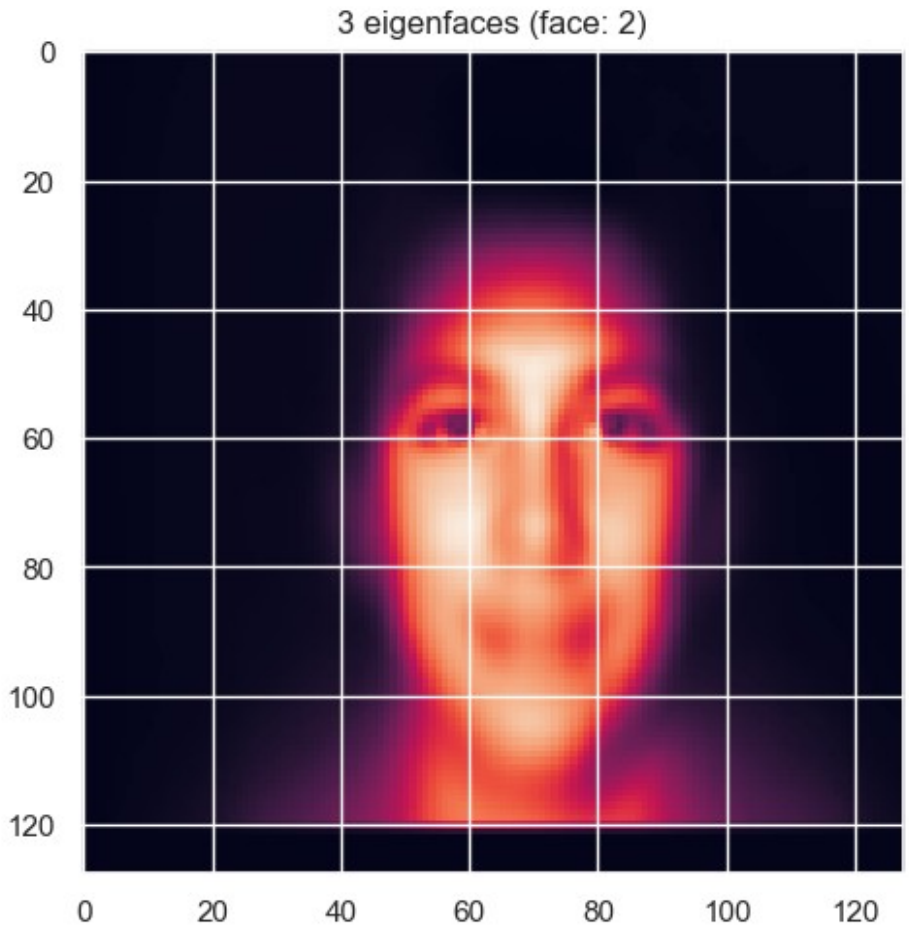


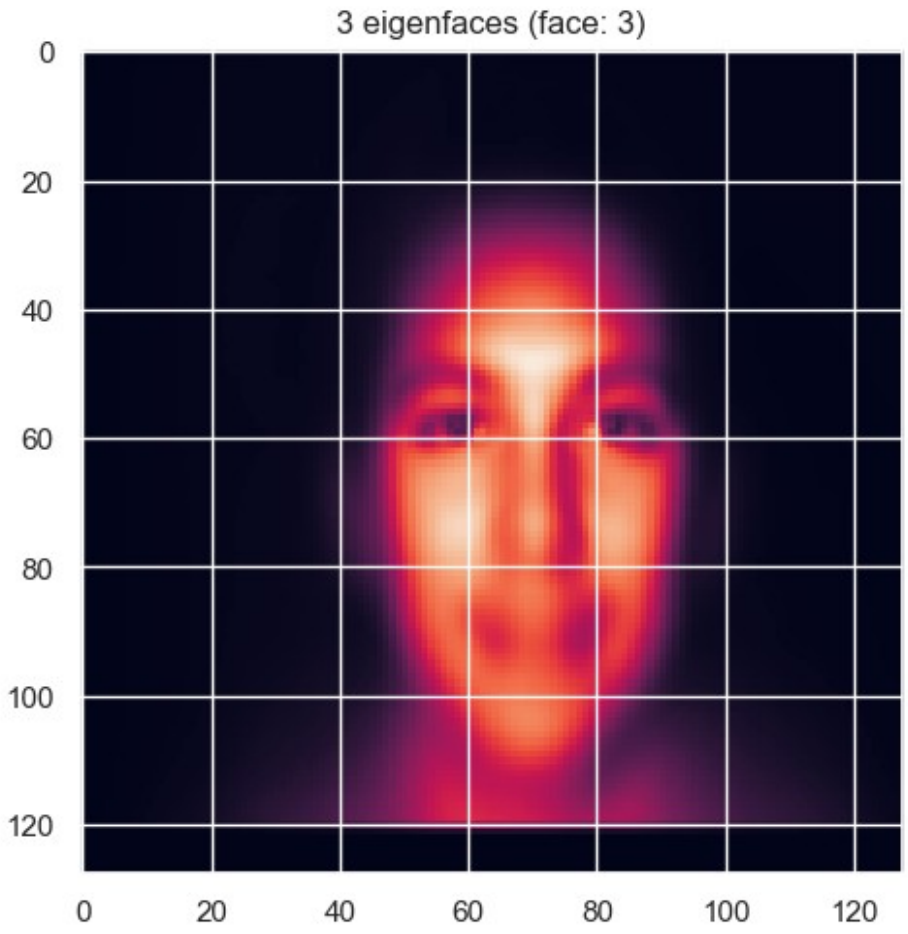
3 Components

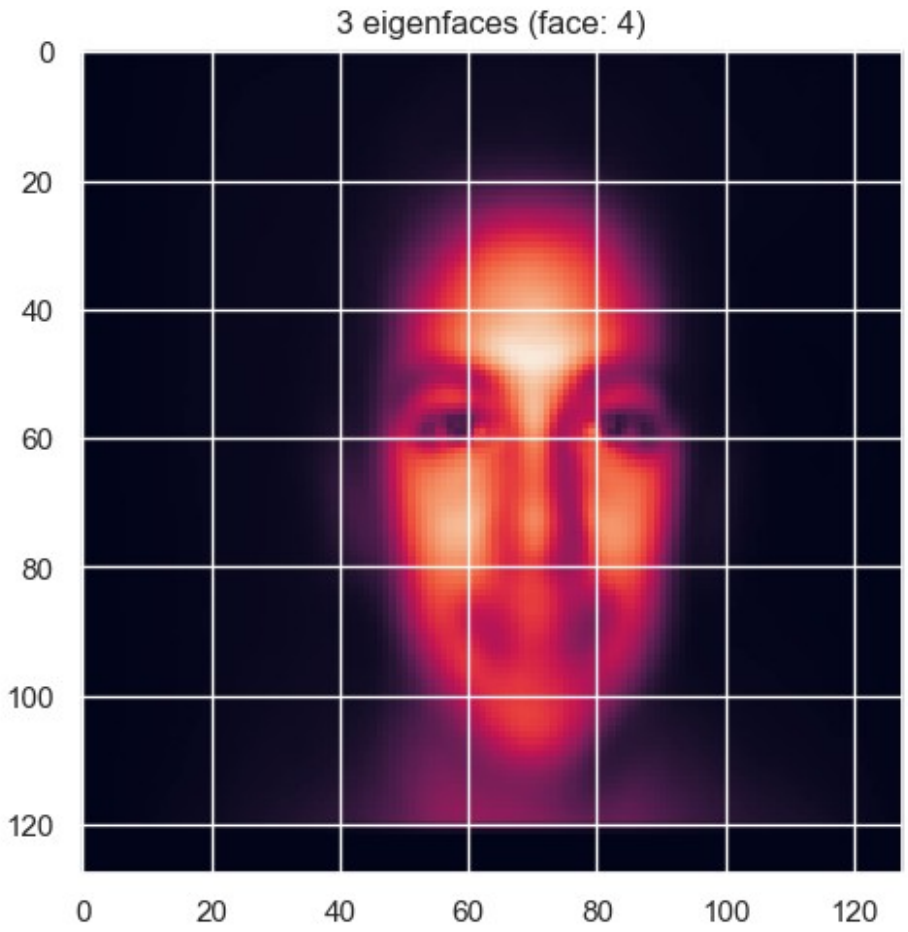
```
for index in range(0, 5):
    face3 = averageImage +
np.sum(np.dot(imagesReduced[index].reshape(1,-1)[:,:3],
eigenImagenFaces[:3,:]), axis = 0)
    plotImage(face3, label= f"3 eigenfaces (face: {index})")
```


3 eigenfaces (face: 0)
0
20
40
60
80
100
120
0
20
40
60
80
100
120


3 eigenfaces (face: 1)
0
20
40
60
80
100
120
0
20
40
60
80
100
120

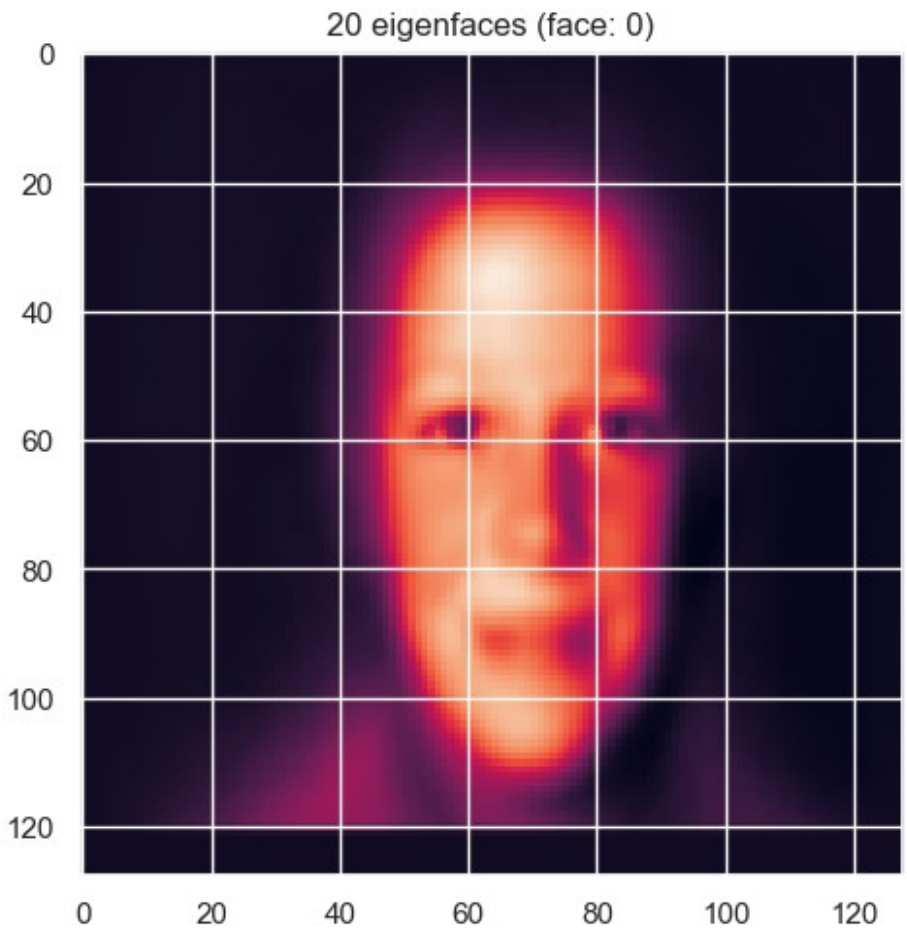

3 eigenfaces (face: 2)
0
20
40
60
80
100
120
0
20
40
60
80
100
120

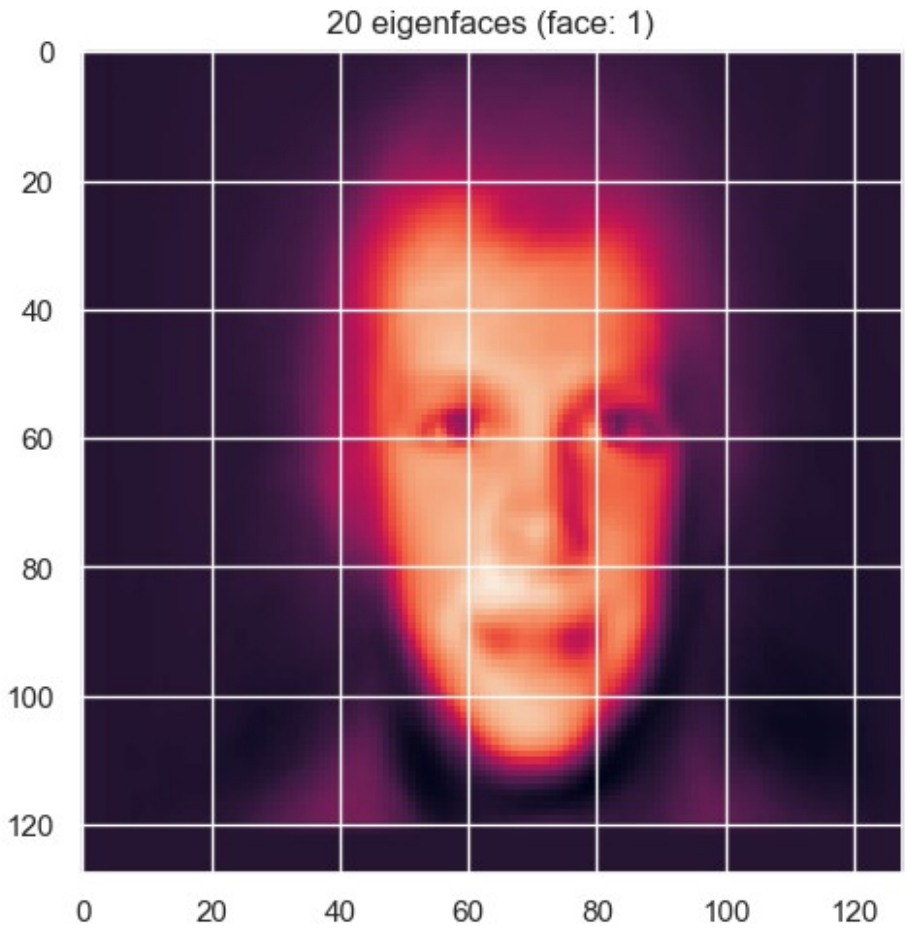

3 eigenfaces (face: 3)
0
20
40
60
80
100
120
0
20
40
60
80
100
120

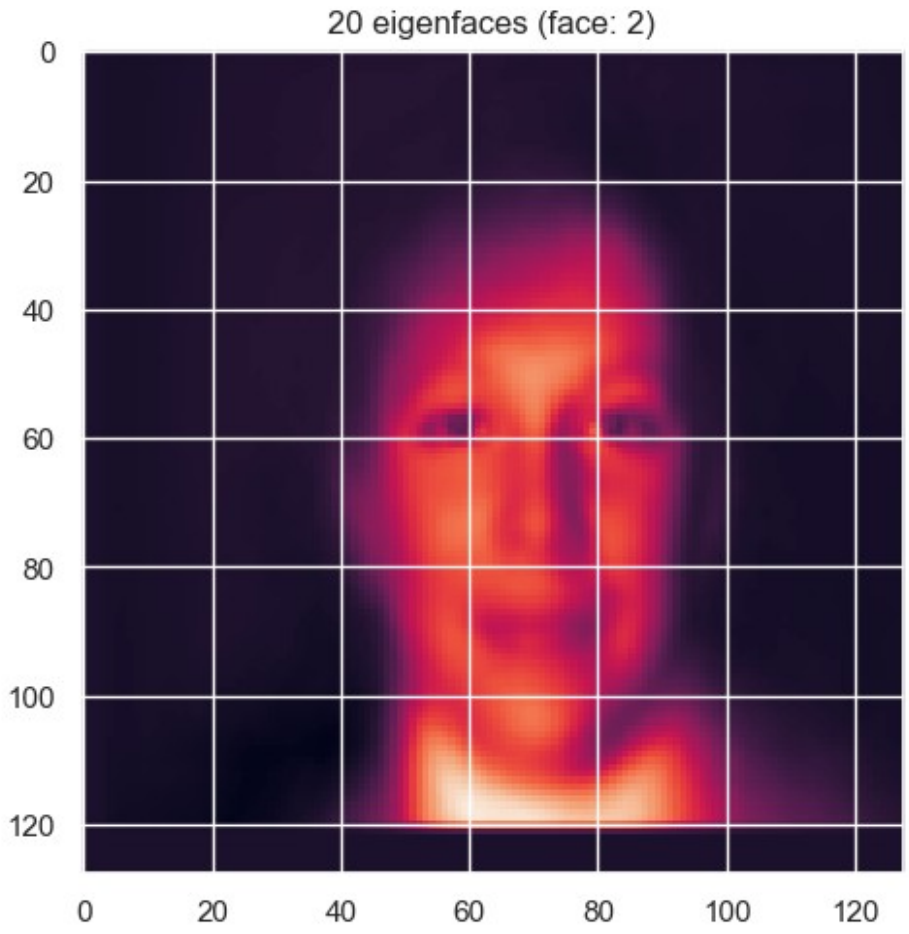


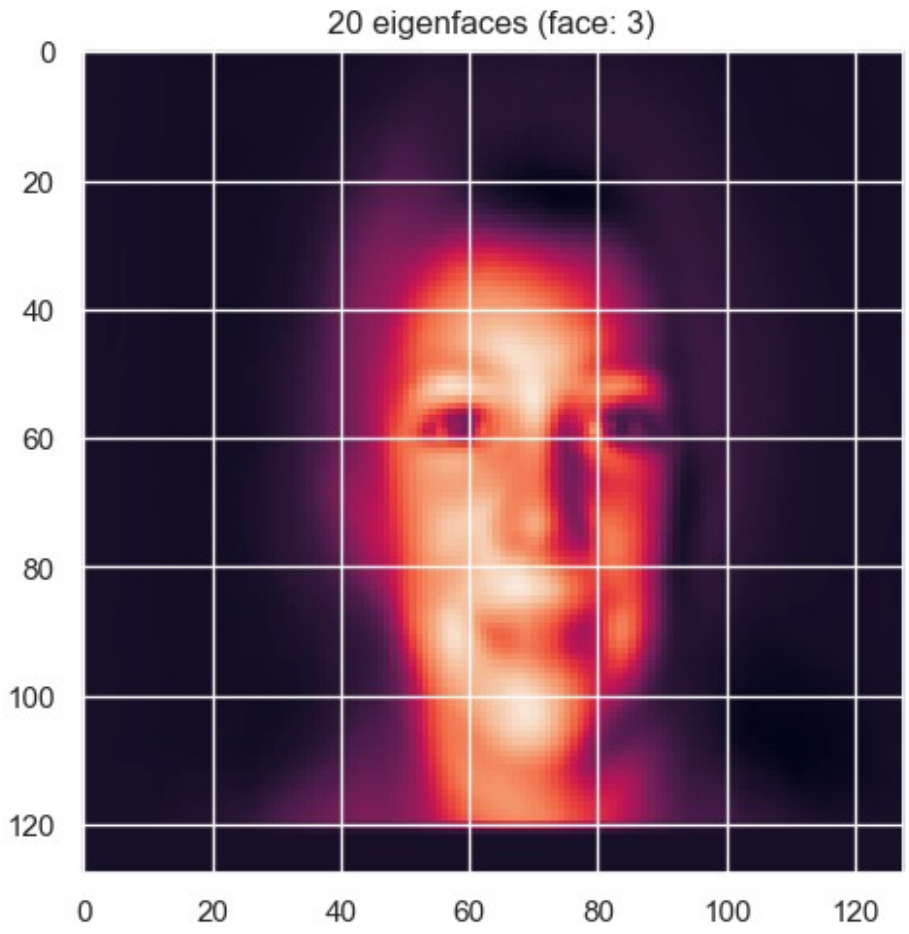
20 Components

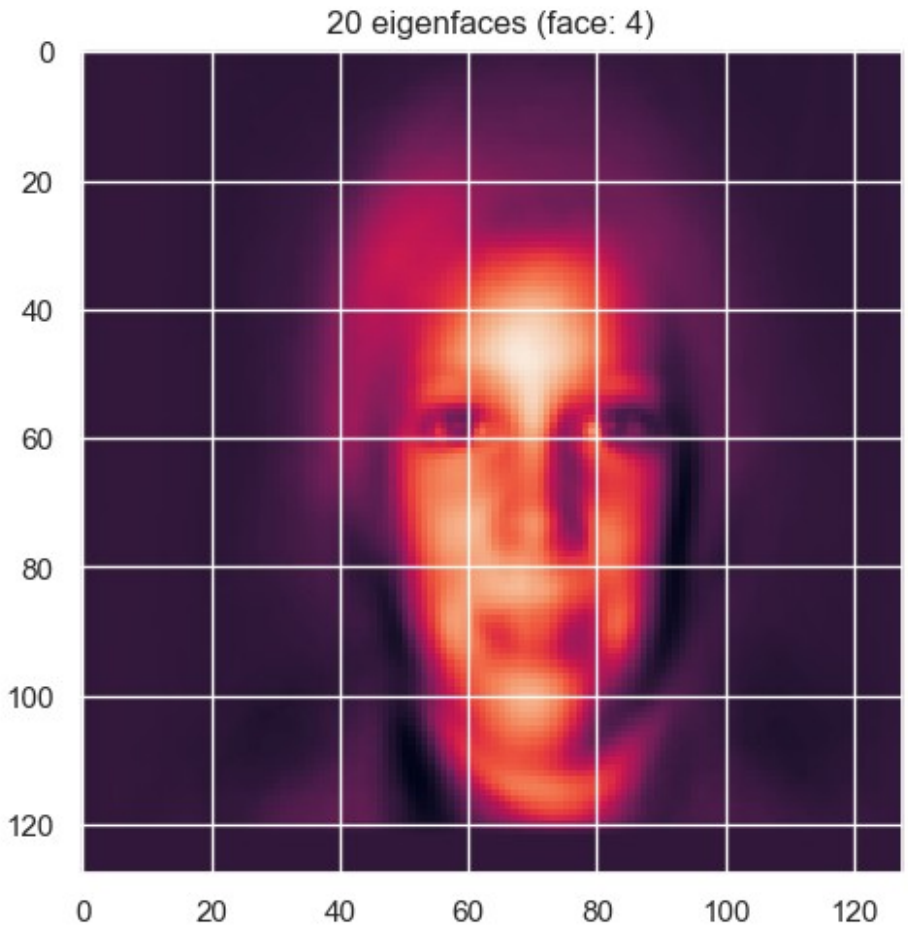
```
for index in range(0, 5):
    face20 = averageImage +
np.sum(np.dot(imagesReduced[index].reshape(1,-1)[:,:20],
eigenImagenFaces[:20,:]), axis = 0)
    plotImage(face20, label= f"20 eigenfaces (face: {index})")
```


20 eigenfaces (face: 0)
0
20
40
60
80
100
120
0
20
40
60
80
100
120


20 eigenfaces (face: 1)
0
20
40
60
80
100
120
0
20
40
60
80
100
120


20 eigenfaces (face: 2)
0
20
40
60
80
100
120
0
20
40
60
80
100
120

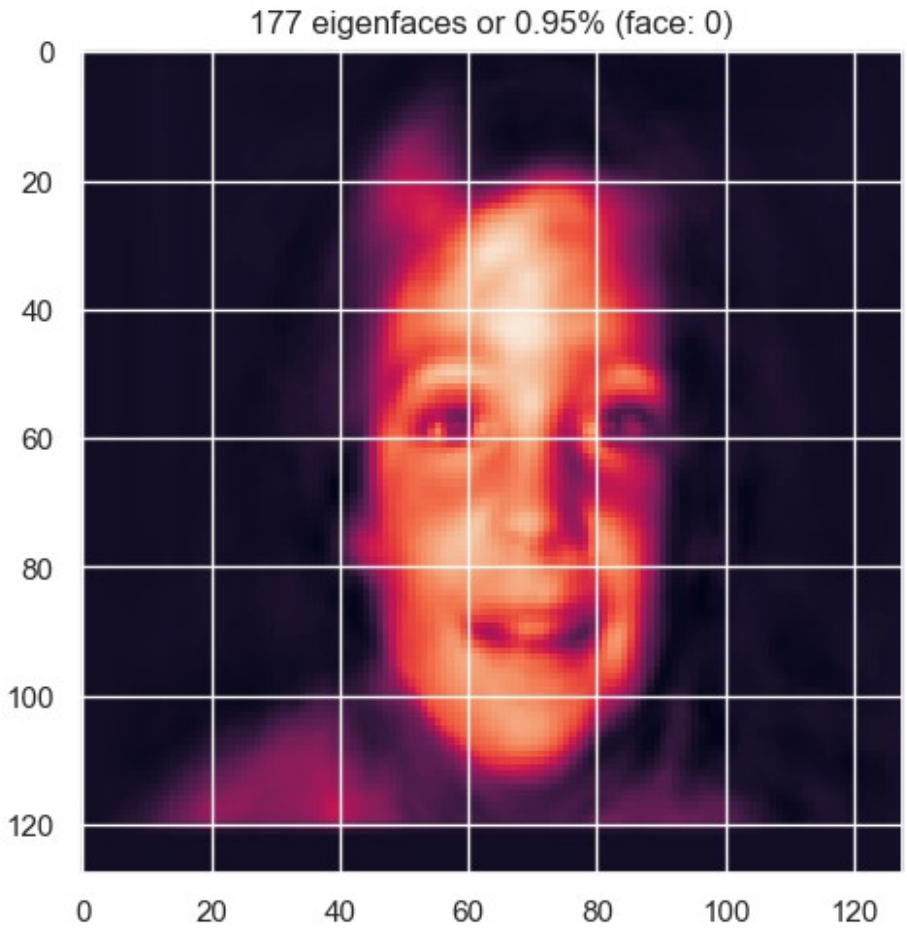

20 eigenfaces (face: 3)
0
20
40
60
80
100
120
0
20
40
60
80
100
120

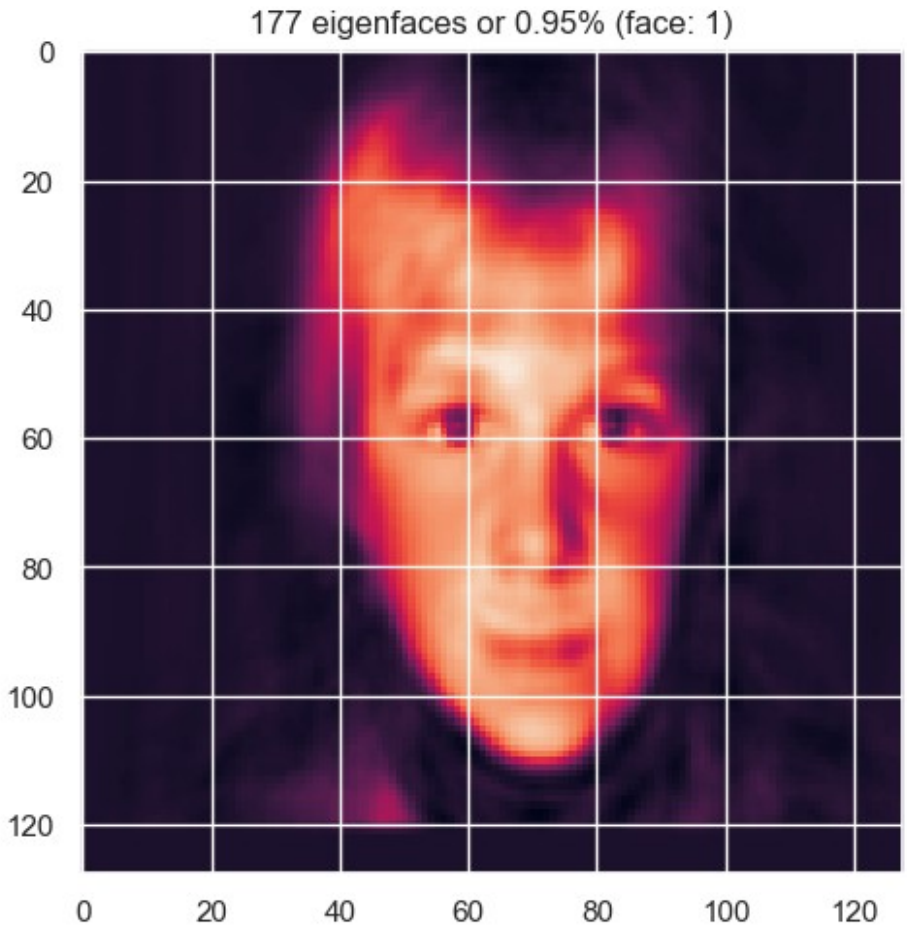


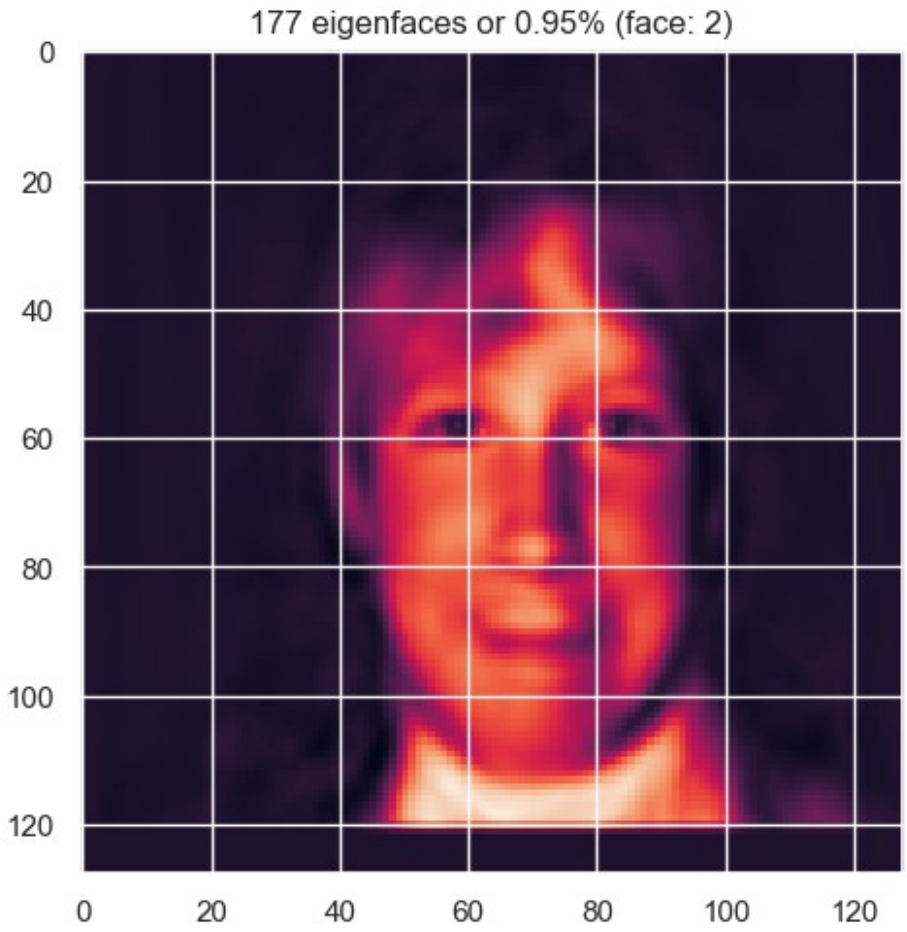

```
explainedVariance = np.cumsum(pca.explained_variance_ratio_)
```

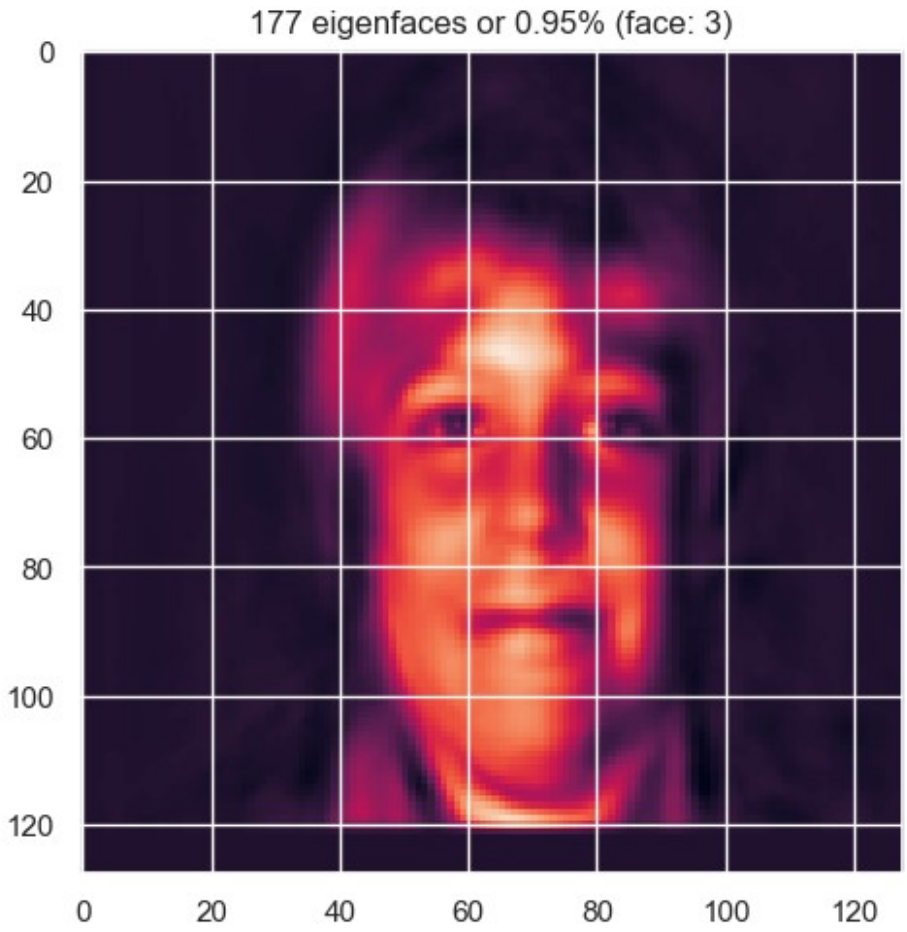
95% of the variance

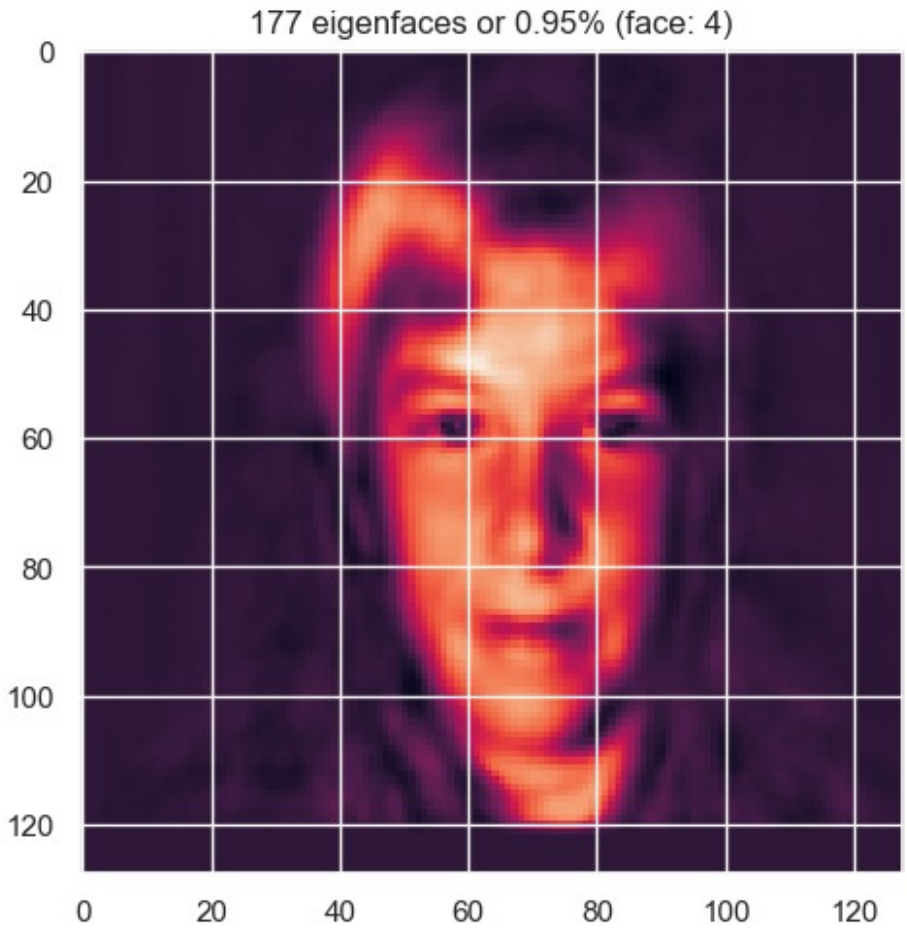
```
percent95 = 0.95
amountComponents95 = np.where(explainedVariance >= percent95)[0][0] +
1
for index in range(0, 5):
    face95Percent = averageImage +
np.sum(np.dot(imagesReduced[index].reshape(1,-1)
[:,:amountComponents95], eigenImagenFaces[:amountComponents95,:]),
axis=0)
    plotImage(face95Percent, label= f"{amountComponents95} eigenfaces
or {percent95}% (face: {index})")
```


177 eigenfaces or 0.95% (face: 0)
0
20
40
60
80
100
120
0
20
40
60
80
100
120


177 eigenfaces or 0.95% (face: 1)
0
20
40
60
80
100
120
0
20
40
60
80
100
120

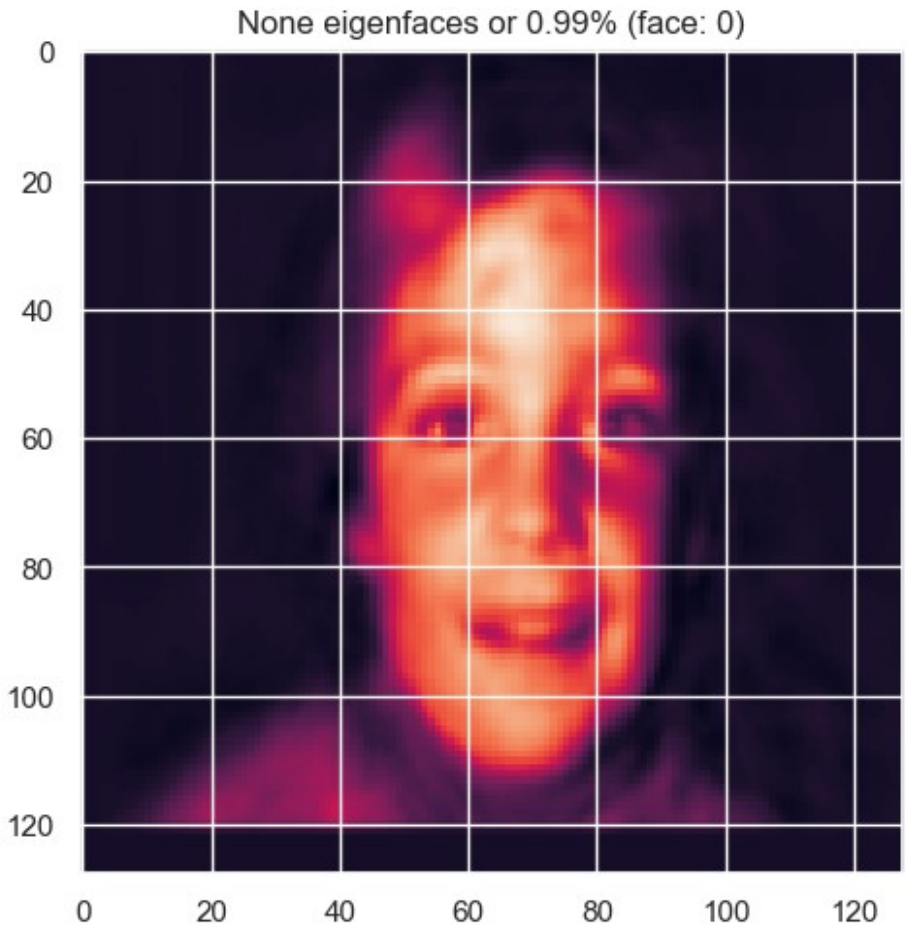

177 eigenfaces or 0.95% (face: 2)
0
20
40
60
80
100
120
0
20
40
60
80
100
120

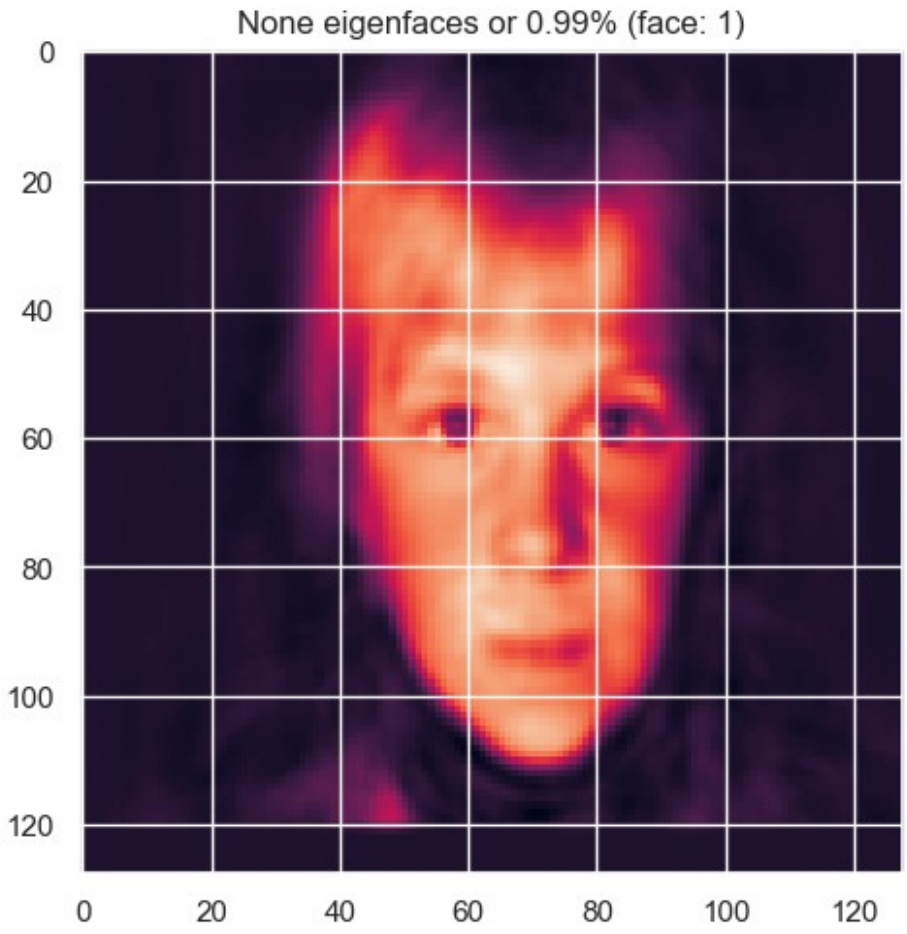

177 eigenfaces or 0.95% (face: 3)
0
20
40
60
80
100
120
0
20
40
60
80
100
120

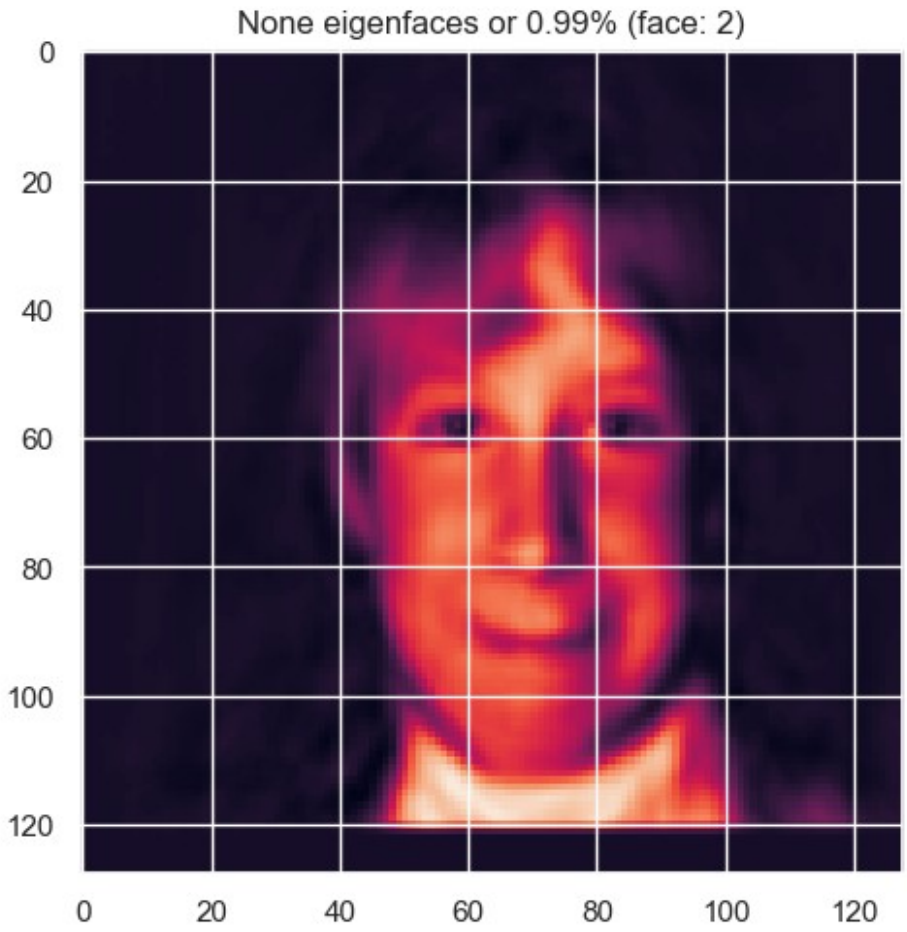


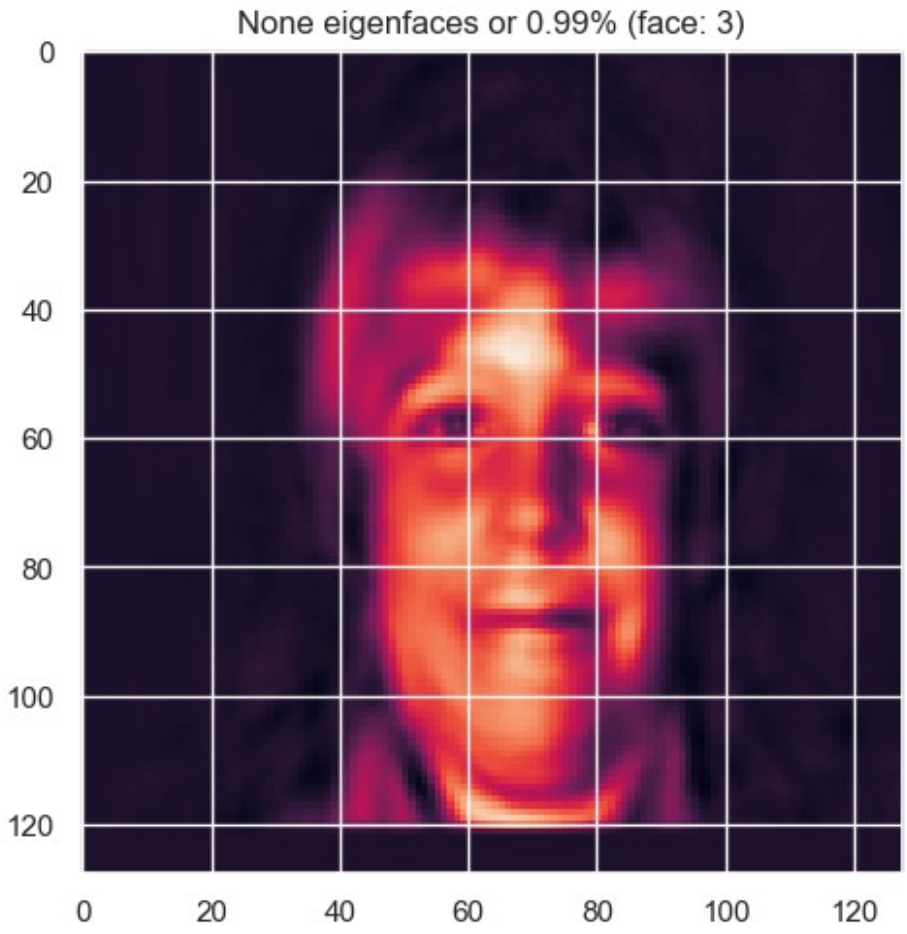
95% of the variance

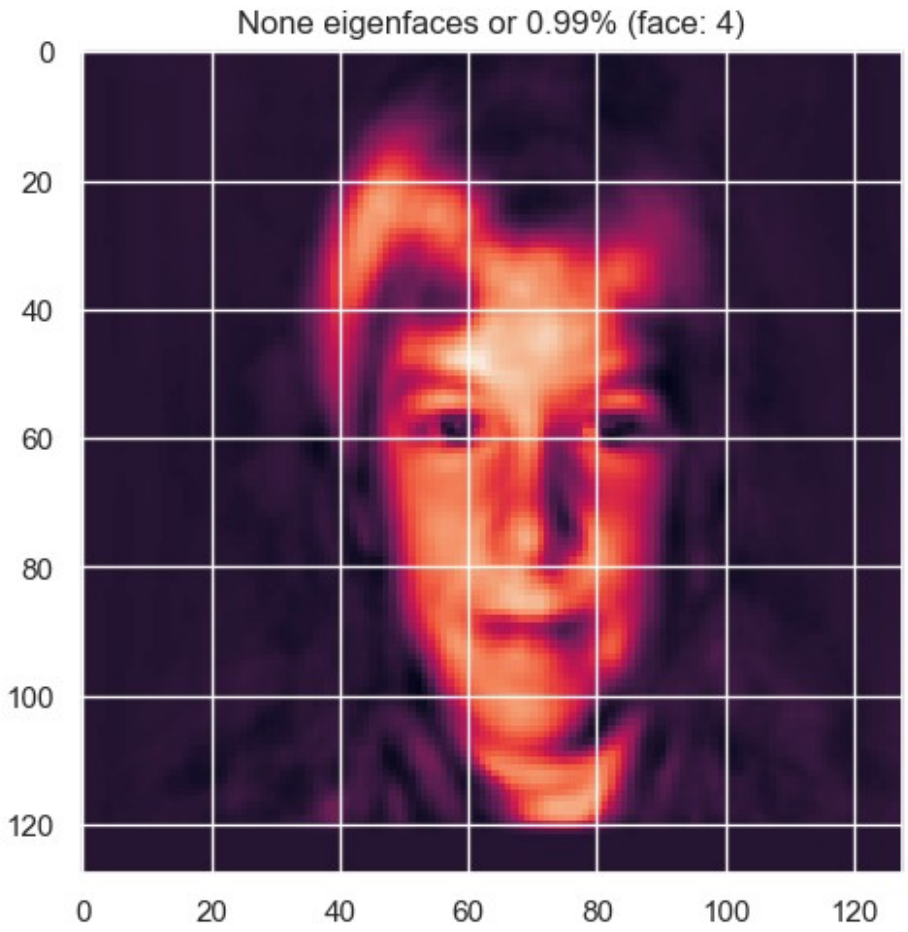
```
percent99 = 0.99
amountComponents99 = (np.where(explainedVariance >= percent99)[0][0] +
1) if (len(np.where(explainedVariance >= 0.99)[0]) > 0) else None

for index in range(0, 5):
    face99Percent = averageImage +
np.sum(np.dot(imagesReduced[index].reshape(1,-1)
[:,:amountComponents99], eigenImagenFaces[:amountComponents99,:]),
axis=0)
    plotImage(face99Percent, label= f"{amountComponents99} eigenfaces
or {percent99}% (face: {index})")
```


None eigenfaces or 0.99% (face: 0)
0
20
40
60
80
100
120
0
20
40
60
80
100
120


None eigenfaces or 0.99% (face: 1)
0
20
40
60
80
100
120
0
20
40
60
80
100
120


None eigenfaces or 0.99% (face: 2)
0
20
40
60
80
100
120
0
20
40
60
80
100
120


None eigenfaces or 0.99% (face: 3)
0
20
40
60
80
100
120
0
20
40
60
80
100
120



Observations

The more eigenfaces there are, the more quality the photo has, but when we pass the barrier of 177 eigenfaces the changes begin to become imperceptible. Under that logic, the image with 1 eigen face is very similar to the average face, in turn, the image with 177 or more (95% or more) eigen faces is very similar to the original image