1- Principal Component Analysis (PCA):

1-1Prepare dataset :

First, we prepare the dataset; we will use the Olivetti Faces dataset in CSV format (Comma Separated Values), a simple file format used to store many tabular variables or data in plain text. In this CSV file, ten different images(positions) for 40 people with grayscale images cropped to 64*64 pixels.

1-2 Importing libraries:

we used the following python libraries :

• NumPy library: for working at arrays.

• pandas library: It allows us to read data in the CSV file.

• matplotlib.pyplot library: It allows us to generate plots to represent data.

• sklearn library: for data processing.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from time import time
```

We also used two helper functions :

• show_orignal_images() : for display the orinal images.

• show_eigenfaces() : for display the eigenfaces.

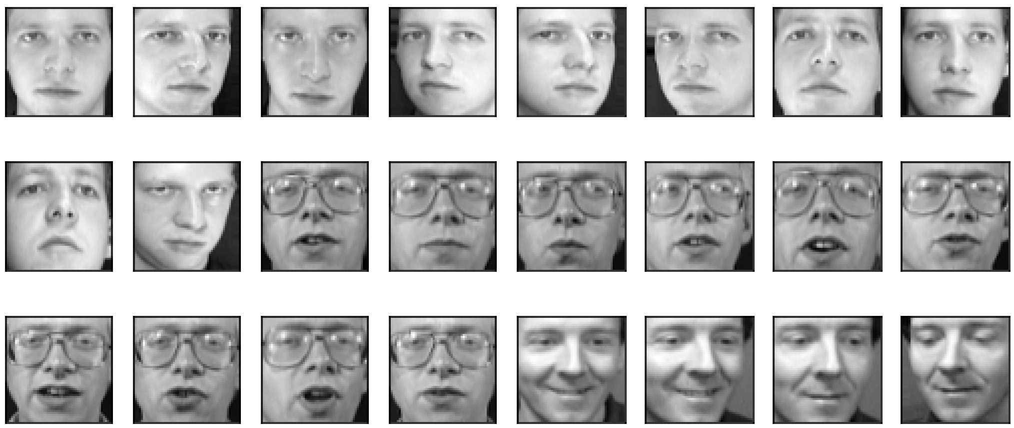
In [2]:
```
###Helper functions
def show_orignal_images(pixels):
        #Displaying Orignal Images
        fig, axes = plt.subplots(3, 8, figsize=(9, 4),
                                 subplot_kw={'xticks':[], 'yticks':[]})
        for i, ax in enumerate(axes.flat):
            ax.imshow(np.array(pixels)[i].reshape(64, 64), cmap='gray')
        plt.show()

def show_eigenfaces(pca):
        #Displaying Eigenfaces
        fig, axes = plt.subplots(3, 8, figsize=(9, 4),
                                 subplot_kw={'xticks':[], 'yticks':[]})
        for i, ax in enumerate(axes.flat):
            ax.imshow(pca.components_[i].reshape(64, 64), cmap='gray')
            ax.set_title("PC " + str(i+1))
        plt.show()
####
```

1-3 Read the data and visualize it:

we read csv file and shap it,then we can visualize it using "show_orignal_images(pixels)" method.

In [3]:
```
# Read dataset and visualize it
df = pd.read_csv("face_data.csv")
target = df["target"].values
pixels = df.drop(["target"], axis=1).values
print ('the shape of the dataframe :')
print (np.array(pixels).shape)
show_orignal_images(pixels)
```

```
the shape of the dataframe :
(400, 4096)
```



1-4 Split Dataset into training set and test set:

first, we import "train_test_split" from the sklearn library because we need to split the data into training data and test data

In [4]:
```
# Split Dataset into training set (75%) and test set (25%)
x_train, x_test, y_train, y_test = train_test_split(pixels, target,test_size=0.25,random_
```

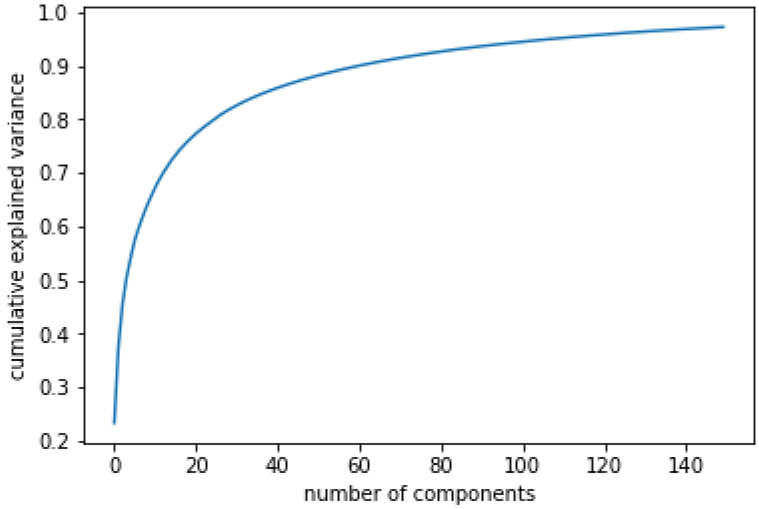
1-5 Perform PCA (Calculate the Eigenfaces):

We identify the principal components by finding the eigenvectors corresponding to the most significant eigenvalues of the covariance matrix of the data and specify it to the PCA algorithm; here, we specify 150 PC.

In [5]:
```
# Reduse the dimensionality
n_components=150
pca = PCA(n_components=150).fit(x_train)
```

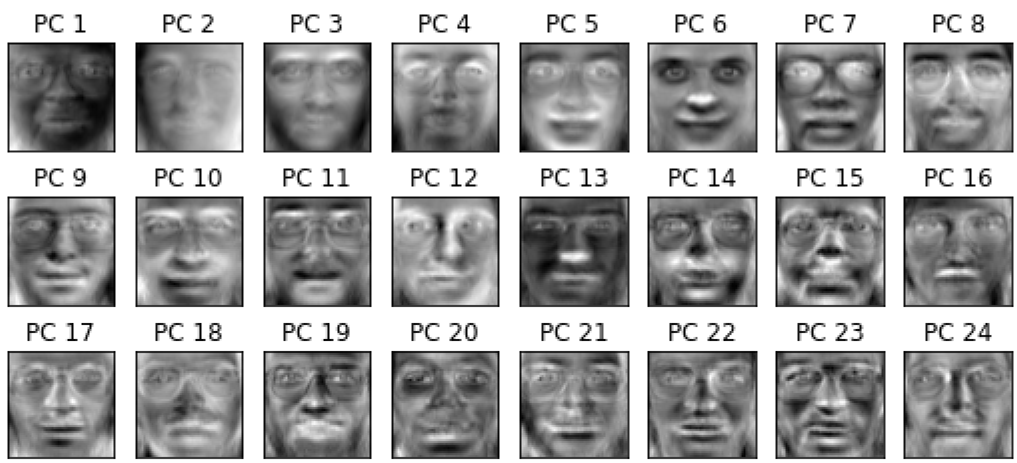
Then we can represent the variance captured by this component using the "explained_variance*ratio*" function from sklearn library.

In [6]:
```
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.show()
```



Visualize the eigen faces

In [7]:
```
# Visualize the eigen faces
show_eigenfaces(pca)
```



1-6 Project Training data to PCA:

here we get the training data and transform it to PCA.

In [8]:
```
# Projecting the input data on the eigenfaces orthonormal basis
Xtrain_pca = pca.transform(x_train)
print("Current shape of input data matrix: ", Xtrain_pca.shape)
```

```
Current shape of input data matrix:  (300, 150)
```

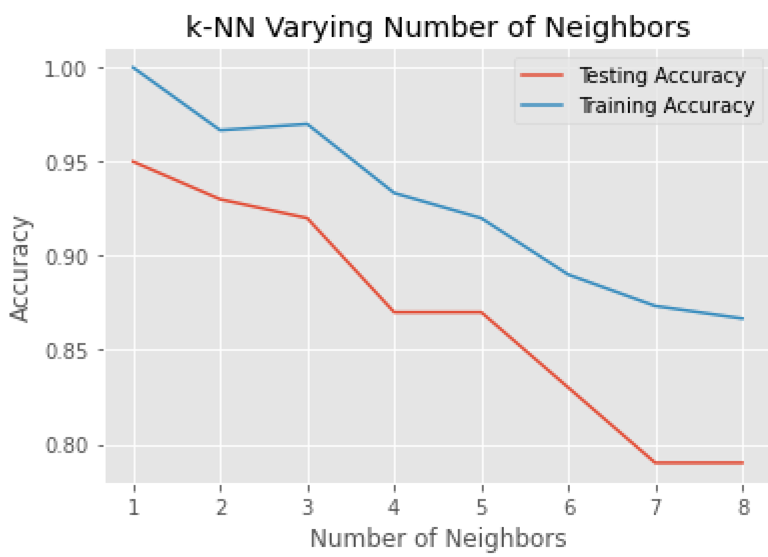
2- K-nearest neighbors classification

2-1 Create a classifier using K-nearest neighbors classification

First, let's observe the accuracies for different values of k using the K-NN Varying Number of Neighbors plot.

In [9]:
```
#
#
#
#
```

```
# Setup arrays to store training and test accuracies
plt.style.use('ggplot')
neighbors = np.arange(1, 9)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

for i, k in enumerate(neighbors):
        # Setup a knn classifier with k neighbors
        knn = KNeighborsClassifier(n_neighbors=k)

        # Fit the model
        knn.fit(x_train, y_train)

        # Compute accuracy on the training set
        train_accuracy[i] = knn.score(x_train, y_train)

        # Compute accuracy on the test set
        test_accuracy[i] = knn.score(x_test, y_test)
#Generate plot
plt.title('k-NN Varying Number of Neighbors')
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.plot(neighbors, train_accuracy, label='Training Accuracy')
plt.legend()
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.show()
```



Then we create a KNeighborsClassifier with maximum testing accuracy that we can observe from the plot; then, we fit the training data on that classifier.

In [10]:
```
#Setup a knn classifier with k neighbors
knn = KNeighborsClassifier(n_neighbors=2)
#Fit the model
knn.fit(x_train,y_train)
#Get the accurency
print("accurency :")
print(knn.score(x_test,y_test))
```

```
accurency :
0.93
```

2-2 Perform testing and get classification report

Scikit-learn provides the facility to calculate Classification reports using the classification_report method. Here we test the accuracy of the model then display the classification report.

In [11]:
```python
t0 = time()
print("predictions :")
y_pred = knn.predict(x_test)
print("done in %0.3fs" % (time() - t0))
print(classification_report(y_test,y_pred))
```

```
predictions :
done in 0.368s
              precision    recall  f1-score   support

           0       1.00      0.50      0.67         2
           1       1.00      1.00      1.00         3
           2       1.00      1.00      1.00         3
           3       1.00      0.67      0.80         3
           4       0.50      1.00      0.67         2
           5       1.00      1.00      1.00         3
           6       1.00      1.00      1.00         2
           7       0.75      1.00      0.86         3
           8       1.00      1.00      1.00         2
           9       1.00      0.50      0.67         2
          10       1.00      1.00      1.00         3
          11       1.00      1.00      1.00         3
          12       0.67      1.00      0.80         2
          13       1.00      1.00      1.00         2
          14       1.00      1.00      1.00         3
          15       1.00      0.50      0.67         2
          16       1.00      1.00      1.00         2
          17       0.67      1.00      0.80         2
          18       1.00      1.00      1.00         3
          19       1.00      1.00      1.00         2
          20       0.75      1.00      0.86         3
          21       1.00      1.00      1.00         3
          22       0.75      1.00      0.86         3
          23       1.00      1.00      1.00         3
          24       1.00      1.00      1.00         2
          25       1.00      1.00      1.00         2
          26       1.00      1.00      1.00         2
          27       1.00      1.00      1.00         2
          28       1.00      0.67      0.80         3
          29       1.00      1.00      1.00         3
          30       1.00      1.00      1.00         3
          31       1.00      1.00      1.00         2
          32       1.00      1.00      1.00         3
          33       1.00      1.00      1.00         2
          34       1.00      0.50      0.67         2
          35       1.00      1.00      1.00         3
          36       1.00      1.00      1.00         3
          37       1.00      1.00      1.00         2
          38       1.00      1.00      1.00         3
          39       1.00      0.50      0.67         2

    accuracy                           0.93       100
   macro avg       0.95      0.92      0.92       100
weighted avg       0.95      0.93      0.93       100
```