Mario Soto

Objective: Use of clustering algorithm and dimensionality reduction algorithms

The classic Olivetti faces dataset contains 400 grayscale 64 × 64–pixel images of faces. Each image is flattened to a 1D vector of size 4,096. 40 different people were photographed (10 times each), and the usual task is to train a model that can predict which person is represented in each picture.

Import the libraries

```
"""

- Sklearn Fetch Olivetti Faces dataset https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_olivetti_faces.html

- sklearn Train and Test split
- sklearn PCA
- sklearn Kmeans
- sklearn Silhouette score

- sklearn Random Forest clasifier https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
"""
from sklearn.datasets import fetch_olivetti_faces
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.ensemble import RandomForestClassifier
```

Load the dataset using the sklearn.datasets.fetch_olivetti_faces() function.

```
# load the sklearn built-in fetch olivetti faces dataset
faces_data = fetch_olivetti_faces()
X_faces = faces_data.data
y_faces = faces_data.target
```

```
# print the Description
print(faces_data.DESCR)
```

```
.. _olivetti_faces_dataset:

The Olivetti faces dataset
--------------------------

`This dataset contains a set of face images`_ taken between April 1992 and
April 1994 at AT&T Laboratories Cambridge. The
:func:`sklearn.datasets.fetch_olivetti_faces` function is the data
fetching / caching function that downloads the data
archive from AT&T.

.. _This dataset contains a set of face images: https://cam-orl.co.uk/facedatabase.html

As described on the original website:

    There are ten different images of each of 40 distinct subjects. For some
    subjects, the images were taken at different times, varying the lighting,
    facial expressions (open / closed eyes, smiling / not smiling) and facial
    details (glasses / no glasses). All the images were taken against a dark
    homogeneous background with the subjects in an upright, frontal position
    (with tolerance for some side movement).

**Data Set Characteristics:**

    =================   =====================
    Classes                                40
    Samples total                         400
    Dimensionality                       4096
    Features            real, between 0 and 1
    =================   =====================
The image is quantized to 256 grey levels and stored as unsigned 8-bit
integers; the loader will convert these to floating point values on the
interval [0, 1], which are easier to work with for many algorithms.

The "target" for this database is an integer from 0 to 39 indicating the
identity of the person pictured; however, with only 10 examples per class, this
relatively small dataset is more interesting from an unsupervised or
semi-supervised perspective.

The original dataset consisted of 92 x 112, while the version available here
consists of 64x64 images.

When using these images, please give credit to AT&T Laboratories Cambridge.
```

```
# Show the data (i.e., the X)
X_faces
```

```
array([[0.30991736, 0.3677686 , 0.41735536, ..., 0.15289256, 0.16115703,
        0.1570248 ],
       [0.45454547, 0.47107437, 0.5123967 , ..., 0.15289256, 0.15289256,
        0.15289256],
       [0.3181818 , 0.40082645, 0.49173555, ..., 0.14049587, 0.14876033,
        0.15289256],
       ...,
       [0.5       , 0.53305787, 0.607438  , ..., 0.17768595, 0.14876033,
        0.19008264],
       [0.21487603, 0.21900827, 0.21900827, ..., 0.57438016, 0.59090906,
        0.60330576],
       [0.5165289 , 0.46280992, 0.28099173, ..., 0.35950413, 0.3553719 ,
        0.38429752]], dtype=float32)
```

```
# Show the targets (i.e., the y)
y_faces
```

```
array([ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  3,  3,  3,  3,
        3,  3,  3,  3,  3,  3,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  5,
        5,  5,  5,  5,  5,  5,  5,  5,  5,  6,  6,  6,  6,  6,  6,  6,  6,
        6,  6,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  8,  8,  8,  8,  8,
        8,  8,  8,  8,  8,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9, 10, 10,
       10, 10, 10, 10, 10, 10, 10, 10, 11, 11, 11, 11, 11, 11, 11, 11, 11,
       11, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 13, 13, 13, 13, 13, 13,
       13, 13, 13, 13, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 15, 15, 15,
       15, 15, 15, 15, 15, 15, 15, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16,
       17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 18, 18, 18, 18, 18, 18, 18,
       18, 18, 18, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 20, 20, 20, 20,
       20, 20, 20, 20, 20, 20, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 22,
       22, 22, 22, 22, 22, 22, 22, 22, 22, 23, 23, 23, 23, 23, 23, 23, 23,
       23, 23, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 25, 25, 25, 25, 25,
       25, 25, 25, 25, 25, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 27, 27,
       27, 27, 27, 27, 27, 27, 28, 28, 28, 28, 28, 28, 28, 28, 28,
       28, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 30, 30, 30, 30, 30, 30,
       30, 30, 30, 30, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32,
       32, 32, 32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33,
       34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 35, 35, 35, 35, 35, 35, 35,
       35, 35, 35, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 37, 37, 37, 37,
       37, 37, 37, 37, 37, 37, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 39,
       39, 39, 39, 39, 39, 39, 39, 39, 39])
```

Split the dataset into training, validation, and test sets

Since the dataset is quite small, you probably want to use stratified sampling to ensure that there are the same number of images per person in each set.

To get three sets we are going to use the following technique:

- Split first into two sets. A larger set and a smaller set. The smaller set will be our test set
- Split the larger set into two sets. A large set for our training, and the small set for out validation set.

```
# First split our data into two sets (X and test)
# Use the following parameters in the method:
# test_size = 40 (not 40%, but 40 elements)
# radom_state 42, shuffle = True,
# for the stratify parameter, use the target variable
X = faces_data.data
y = faces_data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=40, random_state=42, shuffle=True, stratify=y)
```

```
# print the shape of the larger set
print(X_train.shape)
# print the shape of the target set for the large set
print(X_test.shape)
# print the shape of the smaller set (i.e., test set) We should have 40 rows
print(y_train.shape)
# print the shape of the smaller set (i.e., test set)
print(y_test.shape)
```

```
(360, 4096)
(40, 4096)
(360,)
(40,)
```

```
# Now split our larger data set into two sets (train and validation)
# Use the following parameters in the method:
# test_size = 80 (not 80%, but 80 elements)
# radom_state 43, shuffle = True,
# stratify, use the larger set's target variable
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=80, random_state=43, shuffle=True, stratify=y_train)
```

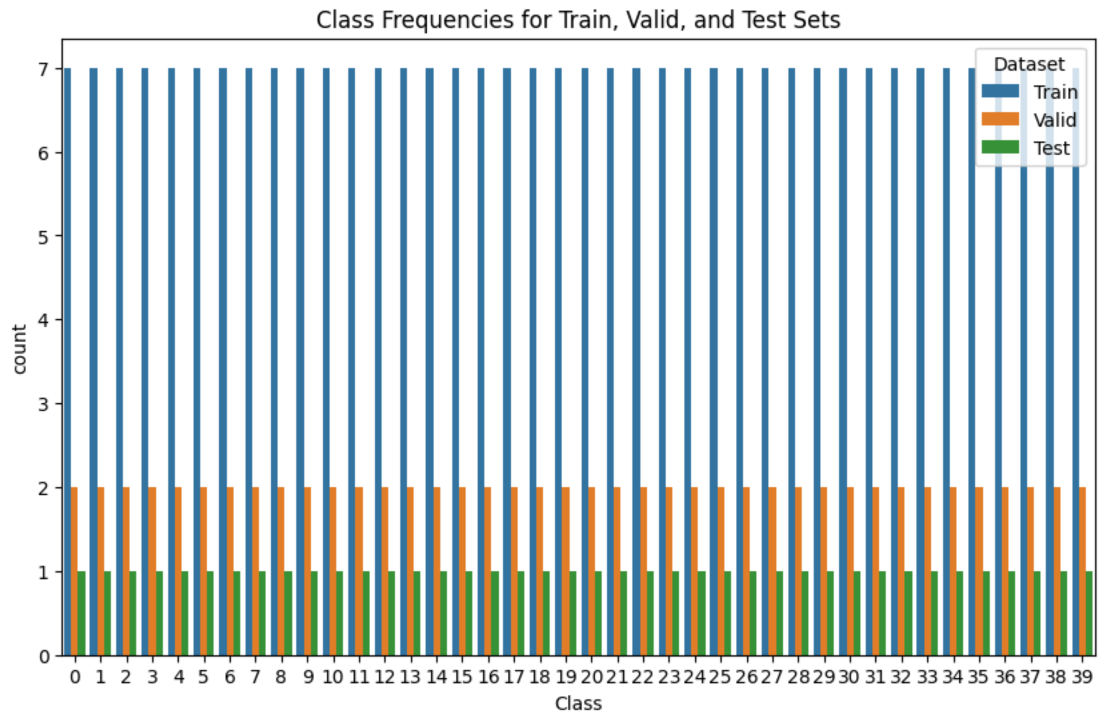
```
# print the shape of the training set
print(X_train.shape)
# print the shape of the target set for the training set
print(X_valid.shape)

# print the shape of the validation set
print(y_train.shape)
# print the shape of the validation set
print(y_valid.shape)
```

```
(280, 4096)
(80, 4096)
(280,)
(80,)
```

Let's visualize if out sets (train, validation, and test) are stratified.

Do not worry about this code.

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Combine the datasets and labels for easier plotting
dataset_labels = ['Train'] * len(y_train) + ['Valid'] * len(y_valid) + ['Test'] * len(y_test)
all_labels = np.concatenate([y_train, y_valid, y_test])

# Create a DataFrame for plotting
data = {'Dataset': dataset_labels, 'Class': all_labels}
df = pd.DataFrame(data)

# Plot the class frequencies
plt.figure(figsize=(10, 6))
sns.countplot(x='Class', hue='Dataset', data=df)
plt.title('Class Frequencies for Train, Valid, and Test Sets')
plt.show()
```



Cool! we mantained the proportional representation of the classes in each set

Let's reduce the dimensionality of our data

Each instance has 4096 features (pixels). To speed things up, we'll reduce the data' dimensionality using PCA

Let's reduce our data while maintaining a 99% of information

```
# Initialize a PCA object using 99% of information as parameter
pca = PCA(n_components=0.99)
```

```
# Train (i.e. fit) the PCA object using our train data
pca.fit(X_train)
```

```
      PCA
PCA(n_components=0.99)
```

```
# Trasforms our train, validation, and test data

X_train_pca = pca.transform(X_train)

X_valid_pca = pca.transform(X_valid)

X_test_pca = pca.transform(X_test)
```

```
# print the final number of components to get 99% of information
print("Number of components:", pca.n_components_)
```

```
Number of components: 199
```

```
# print the shape of the new transofmed train, validation, and test sets
print(X_train_pca.shape)
print(X_valid_pca.shape)
print(X_test_pca.shape)
```

```
(280, 199)
(80, 199)
(40, 199)
```

Cool! We have reduced from 4096 columns to 199 columnd (a reduction of ~95%) of our data while maintaining 99% of the information.

Now, let's cluster the images

As we don't know how many clusters we have in our data, we need to find the best value for 'k'

```
# Define a range of values for the number of clusters (k) from 5 to 150 with a
#step of 5
k_values = range(5, 151, 5)

# Initialize an empty list to store KMeans models for each value of k
kmeans_models = []

# Iterate over each value of k in the specified range, and do:
#   Print the current value of k for tracking progress
#   Create a KMeans model with the current value of k. Additionally, use
#        n_init=10, and random state to 42
#   Fit the KMeans model to the training data
#   Append the trained KMeans model to the list

for k in k_values:
    print("k =", k)

    kmeans = KMeans(n_clusters=k, n_init=10, random_state=42)

    kmeans.fit(X_train_pca)

    kmeans_models.append(kmeans)
```

```
k = 5
k = 10
k = 15
k = 20
k = 25
k = 30
k = 35
k = 40
k = 45
k = 50
k = 55
k = 60
k = 65
k = 70
k = 75
k = 80
k = 85
k = 90
k = 95
k = 100
k = 105
k = 110
k = 115
k = 120
k = 125
k = 130
k = 135
k = 140
k = 145
k = 150
```

Let's find the best number of clusters using the Silhoutte score

We need to Calculate silhouette scores for each KMeans model in our previous list

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html

```
# Create a list to store the Silhouette scores
silhouette_scores = []

# Iterate over each KMeans model in the list of models
for kmeans in kmeans_models:
    # Calculate silhouette score for the current KMeans model
    score = silhouette_score(X_train_pca, kmeans.labels_)

    # Append the calculated score to the silhouette scores list
    silhouette_scores.append(score)



# Find the index of the model with the highest silhouette score in the list
best_index = np.argmax(silhouette_scores)
# print the index
print(best_index)
```

```
     23
```

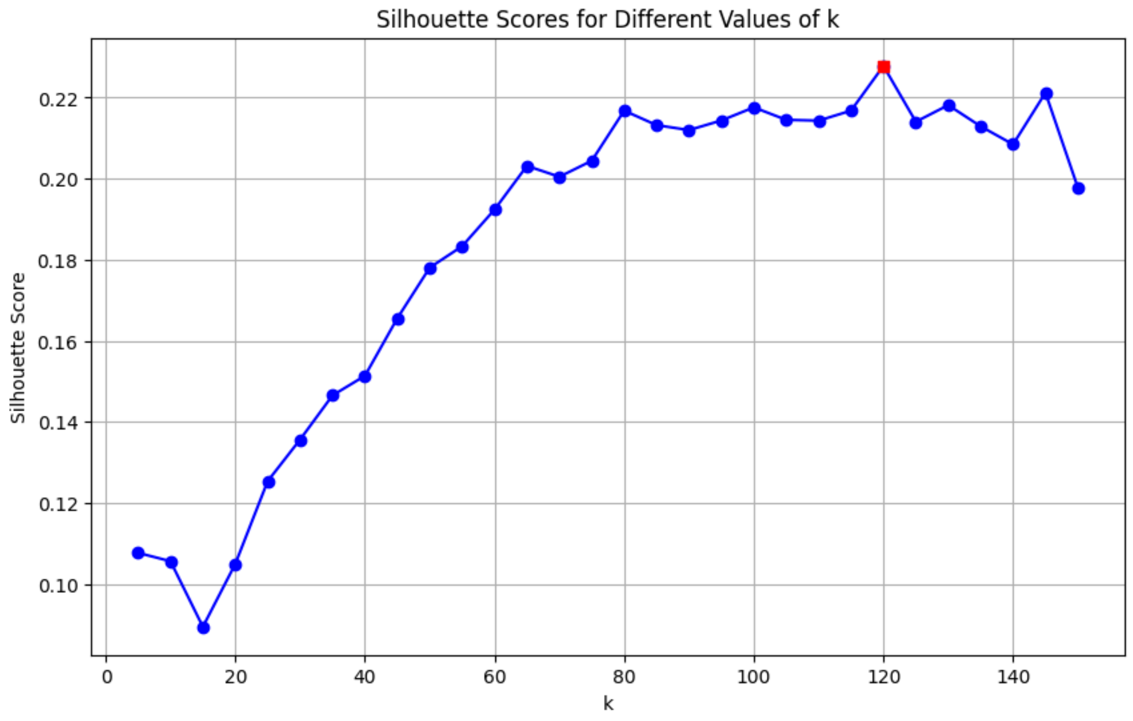
```
# Use the index to extract the best value of k from the range
#based on the highest silhouette score
best_k = k_values[best_index]
# Use the index to extract the silhouette score corresponding to
#the best value of k
best_silhouette_score = silhouette_scores[best_index]


# print the best k
print(best_k)
```

```
     120
```

```
#print the best score for the best k
print(best_silhouette_score)
```

```
     0.22782244
```

```
### Visualize the diifferent Silhouette scores and the
# best one with a different color

# Plot the silhouette scores
plt.figure(figsize=(10, 6))
plt.plot(k_values, silhouette_scores, marker='o', linestyle='-', color='blue')

# Highlight the best silhouette score with a different color and larger marker
plt.scatter(best_k, best_silhouette_score, marker='s', color='red', zorder=5)

# Add labels and title
plt.xlabel('k')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Scores for Different Values of k')
plt.grid(True)


# Show plot
plt.show()
```



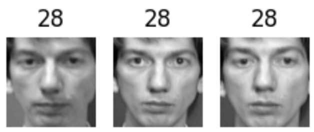
It looks like the best number of clusters is quite high, at 120.

You might have expected it to be 40, since there are 40 different people on the pictures. However, the same person may look quite different on different pictures (e.g., with or without glasses, or simply shifted left or right).

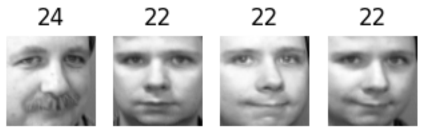
Let's visualize the clusters

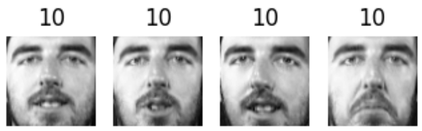
```
# First, put the best model into a variable.
# You find the best model in the previous list of models using the index
best_model = kmeans_models[best_index]


## Do not change this method

def plot_faces(faces, labels, n_cols=5):
    faces = faces.reshape(-1, 64, 64)
    n_rows = (len(faces) - 1) // n_cols + 1
    plt.figure(figsize=(n_cols, n_rows * 1.1))
    for index, (face, label) in enumerate(zip(faces, labels)):
        plt.subplot(n_rows, n_cols, index + 1)
        plt.imshow(face, cmap="gray")
        plt.axis("off")
        plt.title(label)
    plt.show()
```

```
# Use the plot_faces() method to show the elements of each cluster

# Iterate over unique cluster IDs in the labels assigned by the best KMeans model and do:
#   Print the current cluster ID for clarity
#   Create a boolean mask to filter samples belonging to the current cluster
#   [I am giving you this line]
#    (  In simpler terms, this boolean mask is used to filter samples in
#       the dataset based on whether they belong to the
#       current cluster (cluster id). Elements corresponding to True in
#       the mask represent samples in the cluster, and elements corresponding
#       to False represent samples outside the cluster.)
#       (You can compare the best model's labels with the cluster id)
#       If you have questions about this step contact me.
#   Extract faces and labels for samples in the current cluster (form X_train and y_train)
#   Call a function (assumed to be defined elsewhere) to plot the faces in the current cluster


for cluster_id in np.unique(best_model.labels_):
    print("Cluster ID:", cluster_id)

    # Do not change this line. It is the Boolean mask
    in_cluster = best_model.labels_==cluster_id

    # Extract faces and labels for samples in the current cluster
    faces_in_cluster = X_train[in_cluster]
    labels_in_cluster = y_train[in_cluster]

    # Plot the faces in the current cluster
    plot_faces(faces_in_cluster, labels_in_cluster)
```

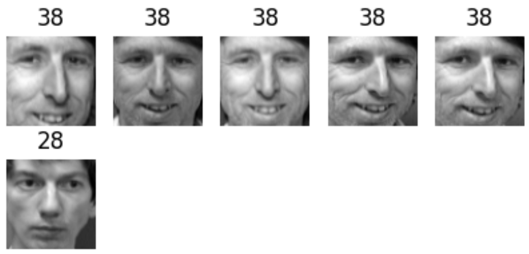
```
Cluster ID: 0
```


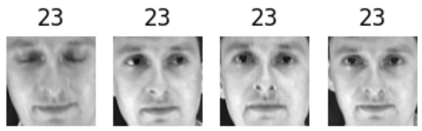
```
Cluster ID: 1
```

```
Cluster ID: 2
```


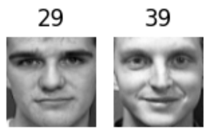
```
Cluster ID: 3
```


```
Cluster ID: 4
```

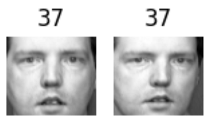
```
Cluster ID: 5
```


```
Cluster ID: 6
```


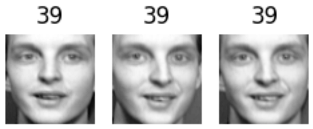
```
Cluster ID: 7
```

```
Cluster ID: 8
```


```
Cluster ID: 9
```

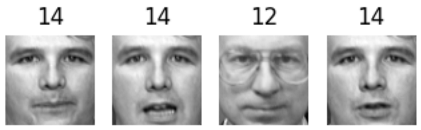
```
Cluster ID: 10
```


```
Cluster ID: 11
```

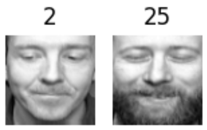


```
Cluster ID: 12
```

```
Cluster ID: 13
```



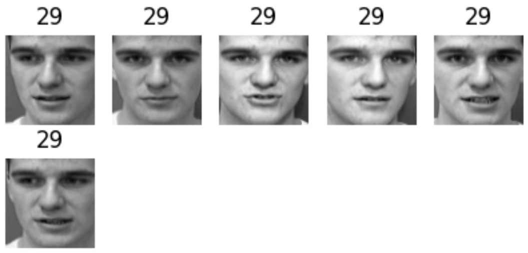
```
Cluster ID: 14
```



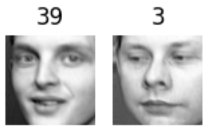
```
Cluster ID: 15
```

```
Cluster ID: 16
```

```
Cluster ID: 17
```



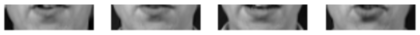
```
Cluster ID: 18
```



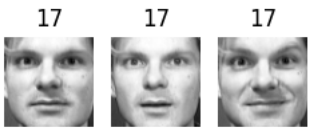
```
Cluster ID: 19
```

```
Cluster ID: 20
```

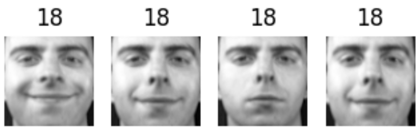
```
Cluster ID: 21
```



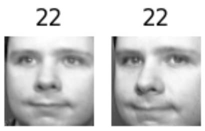
```
Cluster ID: 22
```



```
Cluster ID: 23
```



```
Cluster ID: 24
```

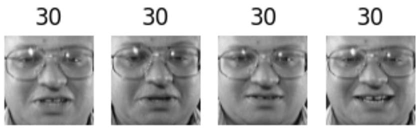


```
Cluster ID: 25
```

```
Cluster ID: 26
```

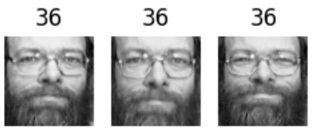
```
Cluster ID: 27
```

```
Cluster ID: 28
```

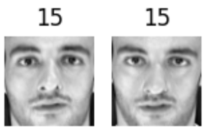


```
Cluster ID: 29
```

```
Cluster ID: 30
```

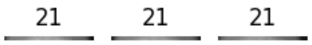


```
Cluster ID: 31
```

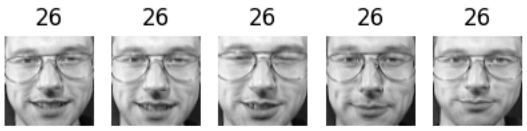


```
Cluster ID: 32
```

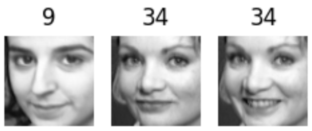
```
Cluster ID: 33
```



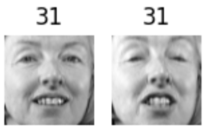
```
Cluster ID: 34
```



```
Cluster ID: 35
```



```
Cluster ID: 36
```



```
Cluster ID: 37
```

```
Cluster ID: 38
```

```
Cluster ID: 39
```

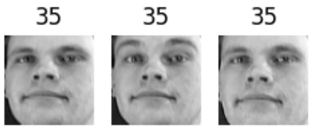
```
Cluster ID: 40
```

```
Cluster ID: 41
```

```
Cluster ID: 42
```

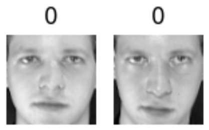
```
Cluster ID: 43
```

```
Cluster ID: 44
```



```
Cluster ID: 45
Cluster ID: 45
```

```
Cluster ID: 46
```

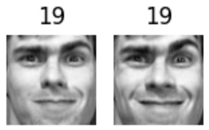
```
Cluster ID: 47
```


```
Cluster ID: 48
```

```
Cluster ID: 49
```

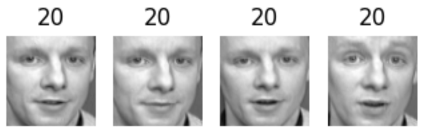
```
Cluster ID: 50
```

```
Cluster ID: 51
```

```
Cluster ID: 52
```


```
Cluster ID: 53
```

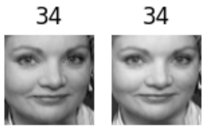
```
Cluster ID: 54
```

```
Cluster ID: 55
```


```
Cluster ID: 56
```

```
Cluster ID: 57
```

```
Cluster ID: 58
```



```
Cluster ID: 59
```

```
Cluster ID: 60
```

```
Cluster ID: 61
```

```
Cluster ID: 62
```

```
Cluster ID: 63
```

```
Cluster ID: 64
```

```
Cluster ID: 65
```

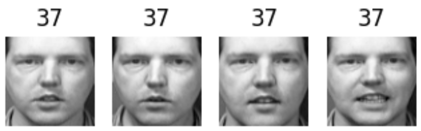
```
Cluster ID: 66
```

```
Cluster ID: 67
```

```
Cluster ID: 68
```

```
Cluster ID: 69
```

```
Cluster ID: 70
```



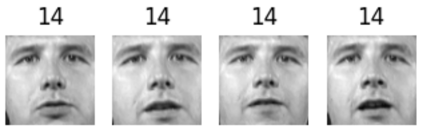
```
Cluster ID: 71
```

```
Cluster ID: 72
```

```
Cluster ID: 73
```

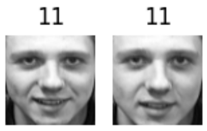
```
Cluster ID: 74
```

```
Cluster ID: 75
```



```
Cluster ID: 76
```

```
Cluster ID: 77
```



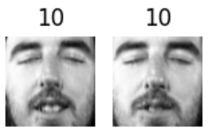
```
Cluster ID: 78
```

```
Cluster ID: 79
```

```
Cluster ID: 80
```

```
Cluster ID: 81
```

```
Cluster ID: 82
```

```
Cluster ID: 83
```


```
Cluster ID: 84
```

```
Cluster ID: 85
```

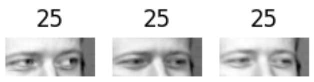
```
Cluster ID: 86
```

```
Cluster ID: 87
```

```
Cluster ID: 88
```

```
Cluster ID: 89
```

```
Cluster ID: 90
```

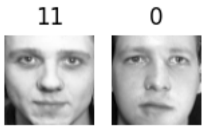
```
Cluster ID: 91
```


```
Cluster ID: 92
```

```
Cluster ID: 93
```

```
Cluster ID: 94
```

```
Cluster ID: 95
```



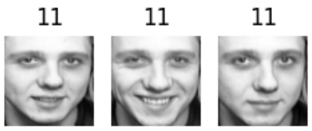
```
Cluster ID: 96
```

```
Cluster ID: 97
```

```
Cluster ID: 98
```

```
Cluster ID: 99
```

```
Cluster ID: 100
```



```
Cluster ID: 101
```

```
Cluster ID: 102
```

```
Cluster ID: 103
```

```
Cluster ID: 104
```

```
Cluster ID: 105
```

```
Cluster ID: 106
```

```
Cluster ID: 107
```

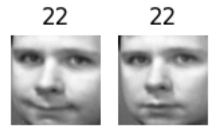
```
Cluster ID: 108
```

```
Cluster ID: 109
```

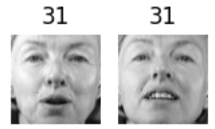
```
Cluster ID: 110
```

```
Cluster ID: 111
```

```
Cluster ID: 112
```



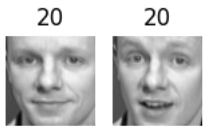
```
Cluster ID: 113
```



```
Cluster ID: 114
```

```
Cluster ID: 115
```

```
Cluster ID: 116
```



```
Cluster ID: 117
```

```
Cluster ID: 118
```

```
Cluster ID: 119
```

Well, it is not perfect but it can be tremendously useful when labeling images in a new dataset: it will usually make labelling much faster.

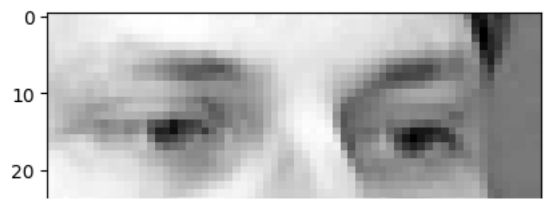
Let's predict the cluster for a test person

```
## We have 40 persons in the test dataframe
## From the original dataset (wihout reduction)
# Let's pick the first one and assign it
## to the face variable
face = X_test[0]
# print the shape of this face
print(face.shape)
# print the values of face
print(face)
```

```
(4096,)
[0.77272725 0.7644628  0.75206614 ... 0.3429752  0.3429752  0.3429752 ]
```

```
# Reshape the face to be (64, 64) [64x64=4096]
face_reshaped = face.reshape(64, 64)

# plot the reshaped face. Doo not change this line
plt.imshow(face_reshaped, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7b44a0af3280>
```



```
# Predict the cluster for this person
# Use the best kmeans model you found to predict the cluster
# label using the reduced 'face'
# get the reduced face (same index)
face_reduced = pca.transform(face.reshape(1, -1))
# print the shape for this face
print(face_reduced.shape)
```

```
(1, 199)
```

```
# print the best model
print(best_model)
```

```
KMeans(n_clusters=120, n_init=10, random_state=42)
```

```
# Do not change this line.
# https://github.com/pycaret/pycaret/issues/3774
best_model.cluster_centers_ = best_model.cluster_centers_.astype(float)
```

```
# Use the best model to predict the reduced face's cluster label
predicted_cluster_label = best_model.predict(face_reduced)

# Print the predicted cluster label
print("Predicted cluster label:", predicted_cluster_label[0])
```

Find "Cluster 56" in the plot for all clusters. Is this person correctly assignmed to that cluster? Pretty cool!

Finally, let's use the reduced data to predict which person is represented in each picture

We are going to use a poweful classifier called Random Forest

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

First, let's try with the original data without reduction

```
# Create a Ramdom Forest Classifier model. Use as parameters:
#   n_estimators =150, and random state to 42
random_forest_model = RandomForestClassifier(n_estimators=150, random_state=42)
```

```
# We will use the original X_train and y_train without reduction
# Check again the shape of these dataframes
print(X_train.shape)

# Check the shape of y_train
```