

# Guided Project Report

## Face detection and recognition

Name: Shruti Verma  
Course: AI and ML  
(Batch 4)  
Duration: 10 months

Problem Statement: Build a machine learning model using PCA for face detection and recognition.

### Prerequisites

What things you need to install the software and how to install them:

Python 3.8 or higher versions This setup requires that your machine has latest version of python. The following url <https://www.python.org/downloads/> can be referred to download python. Once you have python downloaded and installed, you will need to setup PATH variables (if you want to run python program directly, detail instructions are below in how to run software section). To do that check this: <https://www.pythoncentral.io/add-python-to-path-python-is-not-recognized-as-an-internal-or-external-command/>. Setting up PATH variable is optional as you can also run program without it and more instruction are given below on this topic.

Second and easier option is to download anaconda and use its anaconda prompt to run the commands. To install anaconda check this url <https://www.anaconda.com/download/> You will also need to download and install below 3 packages after you install either python or anaconda from the steps above Sklearn (scikit-learn) numpy scipy if you have chosen to install python 3.8 then run below commands in command prompt/terminal to install these packages `pip install -U scikit-learn` `pip install numpy` `pip install scipy` if you have chosen to install anaconda then run below commands in anaconda prompt to install these packages `conda install -c scikit-learn` `conda install -c anaconda numpy` `conda install -c anaconda scipy`

### Dataset used

The data source is LFW\_peoples( labelled faces in wild) dataset provided in the scikit-learn library. The dataset is used with the condition, classes that have a minimum (use `min_faces_per_person = 70`, `resize = 0.4` ) 70 images. The data set contains more than 13,000 images of faces collected from the web. Each face has been labeled with the name

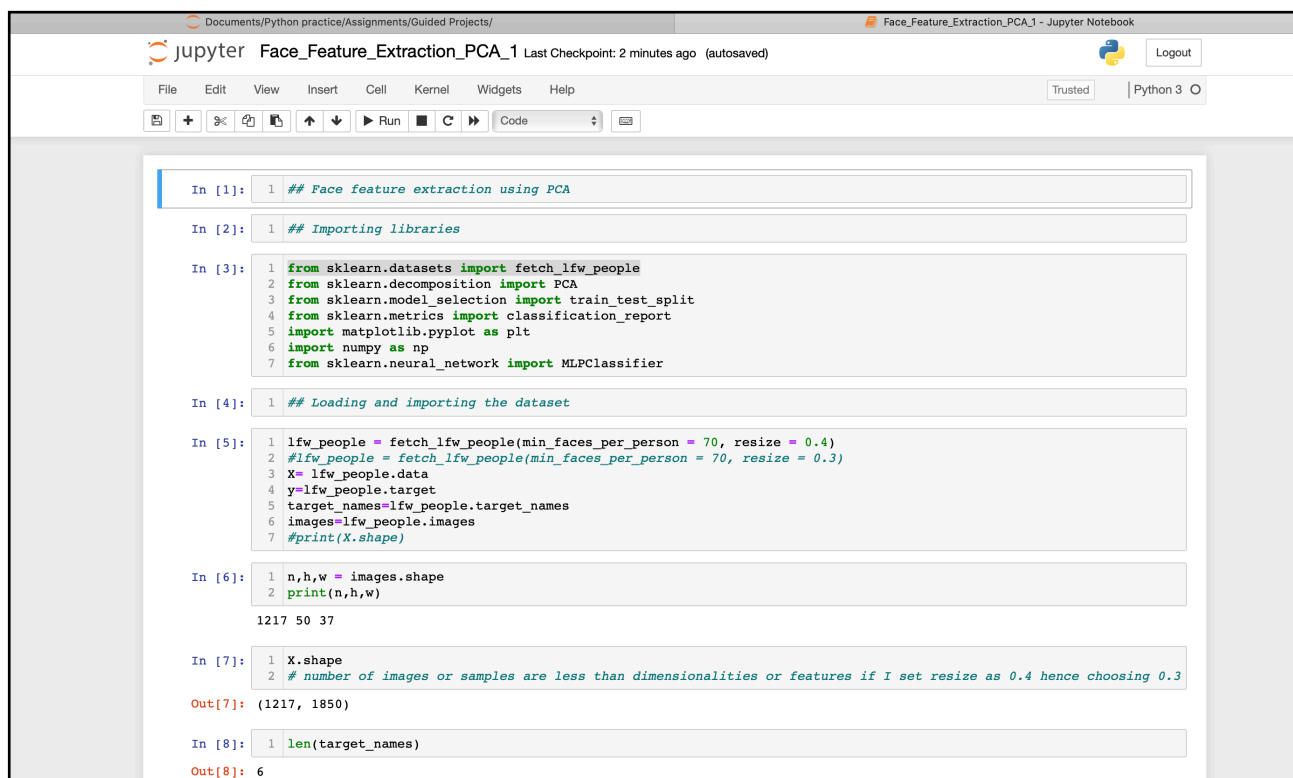
of the person pictured. 1680 of the people pictured have two or more distinct photos in the data set. There are close 6,000 dimensionality.

## Method used for detection

Principal component analysis

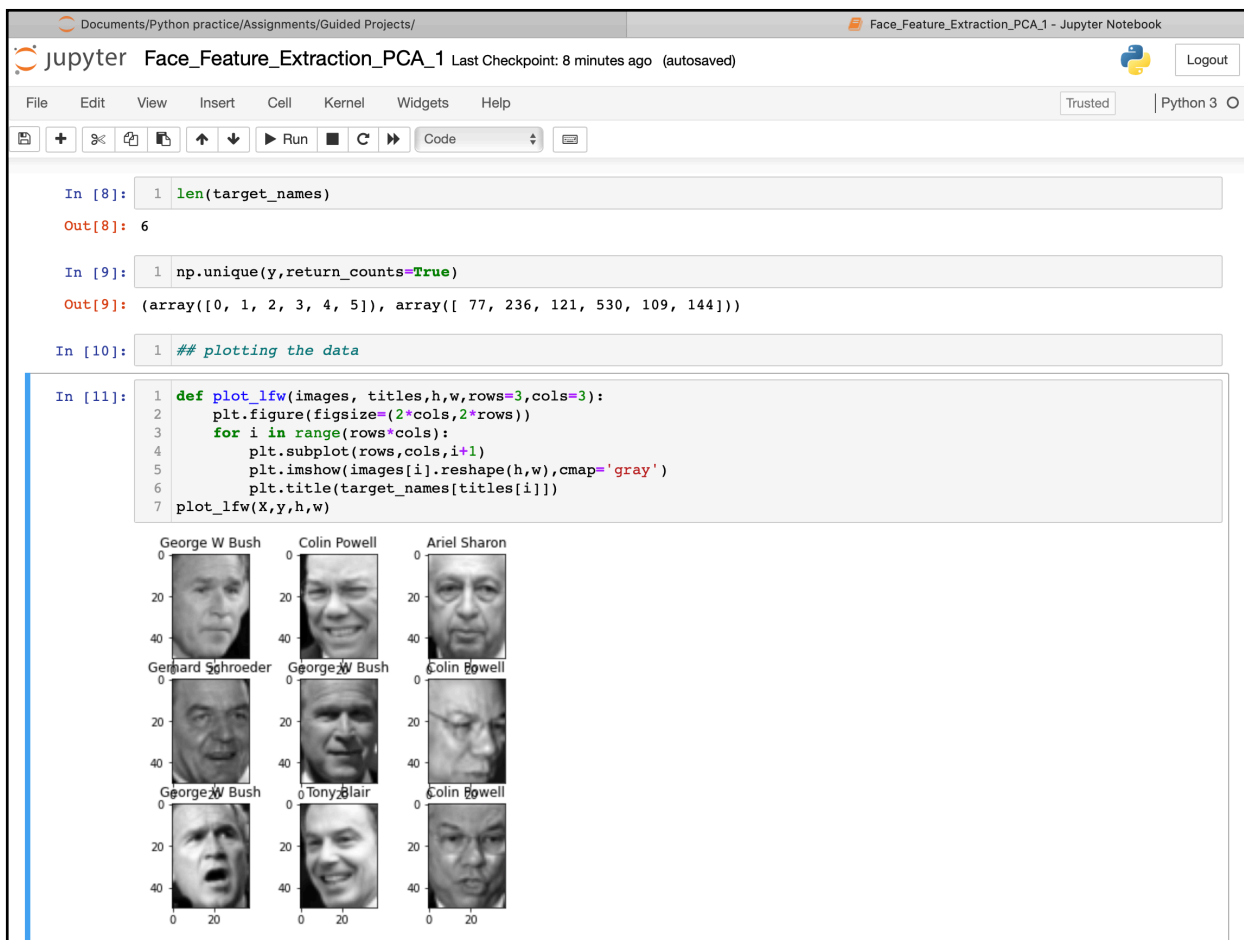
Flatten image dataset -> PCA -> MLP classification

Importing the libraries and capturing images:



```
Documents/Python practice/Assignments/Guided Projects/ Face_Feature_Extraction_PCA_1 - Jupyter Notebook
jupyter Face_Feature_Extraction_PCA_1 Last Checkpoint: 2 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
In [1]: 1 ## Face feature extraction using PCA
In [2]: 1 ## Importing libraries
In [3]: 1 from sklearn.datasets import fetch_lfw_people
        2 from sklearn.decomposition import PCA
        3 from sklearn.model_selection import train_test_split
        4 from sklearn.metrics import classification_report
        5 import matplotlib.pyplot as plt
        6 import numpy as np
        7 from sklearn.neural_network import MLPClassifier
In [4]: 1 ## Loading and importing the dataset
In [5]: 1 lfw_people = fetch_lfw_people(min_faces_per_person = 70, resize = 0.4)
        2 #lfw_people = fetch_lfw_people(min_faces_per_person = 70, resize = 0.3)
        3 X= lfw_people.data
        4 y=lfw_people.target
        5 target_names=lfw_people.target_names
        6 images=lfw_people.images
        7 #print(X.shape)
In [6]: 1 n,h,w = images.shape
        2 print(n,h,w)
1217 50 37
In [7]: 1 X.shape
        2 # number of images or samples are less than dimensionalities or features if I set resize as 0.4 hence choosing 0.3
Out[7]: (1217, 1850)
In [8]: 1 len(target_names)
Out[8]: 6
```

## Plotting the images

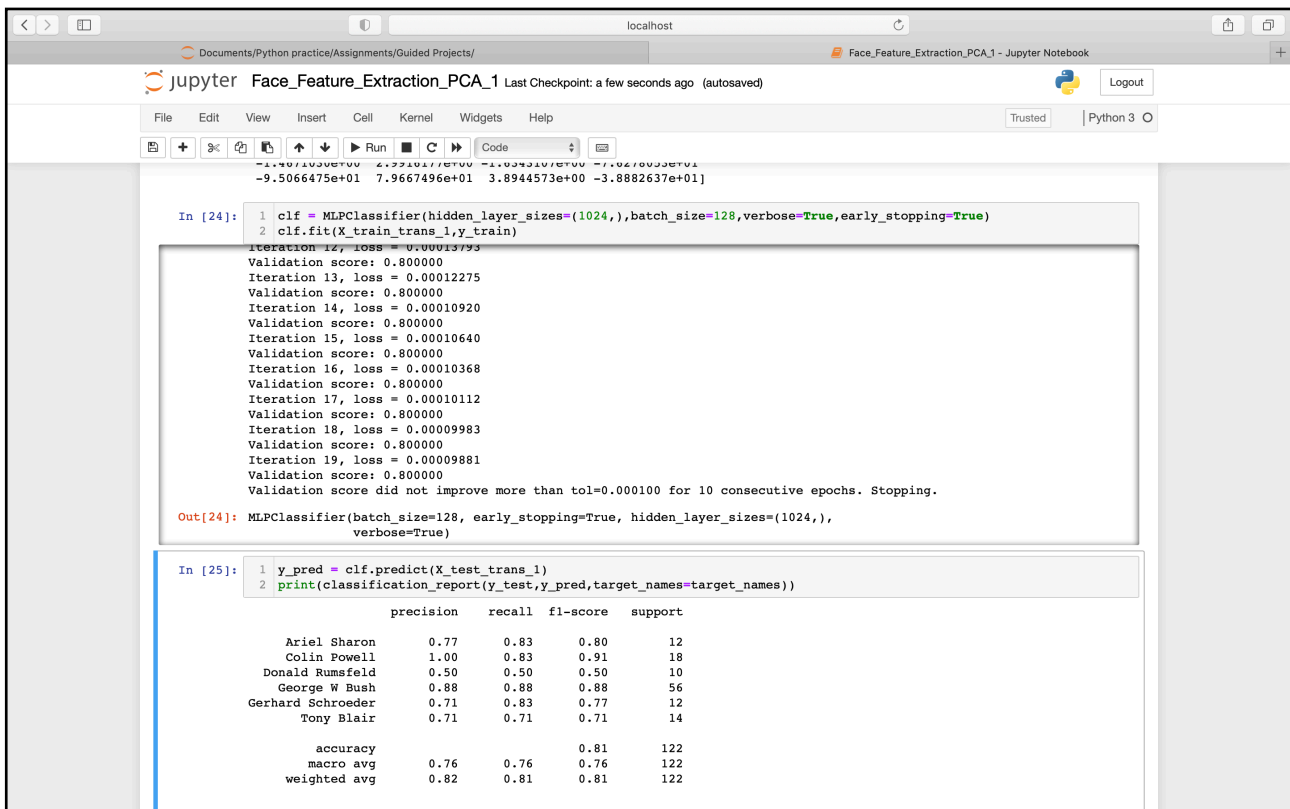


Splitting the training and testing data into 90 :10.

This is hyper parameter and can be varied to 85:15 or 80:20 ratio also.

```
Documents/Python practice/Assignments/Guided Projects/ Face_Feature_Extraction_PCA_1 - Jupyter Notebook
Jupyter Face_Feature_Extraction_PCA_1 Last Checkpoint: 12 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
In [12]: 1 # spilting the train and test data
In [13]: 1 X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.1)
In [14]: 1 X_train.shape
Out[14]: (1095, 1850)
In [15]: 1 pca=PCA(n_components=120)
          2 pca.fit(X_train)
Out[15]: PCA(n_components=120)
In [16]: 1 X_train_trans_1 = pca.transform(X_train)
          2 X_test_trans_1 = pca.transform(X_test)
In [17]: 1 ## checking the shape of transformed trained data
          2 X_train_trans_1.shape
Out[17]: (1095, 120)
In [18]: 1 print("Sample Data point after applying PCA\n", X_train_trans_1[0])
Sample Data point after applying PCA
[ 1.8843967e+03  1.2173498e+02 -8.8246045e+02 -6.4566980e+02
 2.7310669e+02 -2.6214212e+02  5.4722803e+02  7.9852470e+01
-3.6369867e+02  9.0613365e+00  2.0212723e+02  1.4928101e+02
-2.9516705e+02 -8.8462608e+01 -8.6939026e+01 -2.5131271e+02
 1.3522586e+02 -3.6708542e+01  1.3752499e+02 -4.9527328e+01
-2.5368770e+02  1.6897423e+01  5.2662872e+01  1.3238553e+02
-1.7678476e+02  1.3557181e+02  1.1225069e+02 -1.7076546e+01
-4.4958664e+01 -7.7479645e+01 -3.7796860e+01 -6.5475006e+01
-3.9000286e+01 -9.0301659e+01  4.6496651e+01  3.3838230e+01
-1.0851994e+02  6.7854546e+01 -1.2507789e+02  1.0922619e+02
 8.4207901e+01 -1.3380695e+02 -1.8661429e+02  2.6434549e+01
 3.2155800e+01  1.1505450e+01 -2.2860565e+00 -7.7489708e+01
 7.3646294e+01 -6.5107040e+01  1.1867351e+01 -7.8272713e+01
-6.7270584e+00  1.8410522e+01  1.9893097e+01  4.2238930e+01
 5.3691704e+01 -9.9404938e+01  1.7036217e+02 -2.2563335e+01]
```

## MLP Clarification and prediction



The screenshot shows a Jupyter Notebook titled 'Face\_Feature\_Extraction\_PCA\_1'. The notebook is running on a local host. The code in the notebook is as follows:

```
In [24]: 1 clf = MLPClassifier(hidden_layer_sizes=(1024,), batch_size=128, verbose=True, early_stopping=True)
2         clf.fit(X_train_trans_1, y_train)

Iteration 12, loss = 0.00013793
Validation score: 0.800000
Iteration 13, loss = 0.00012275
Validation score: 0.800000
Iteration 14, loss = 0.00010920
Validation score: 0.800000
Iteration 15, loss = 0.00010640
Validation score: 0.800000
Iteration 16, loss = 0.00010368
Validation score: 0.800000
Iteration 17, loss = 0.00010112
Validation score: 0.800000
Iteration 18, loss = 0.00009983
Validation score: 0.800000
Iteration 19, loss = 0.00009881
Validation score: 0.800000
Validation score did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.

Out[24]: MLPClassifier(batch_size=128, early_stopping=True, hidden_layer_sizes=(1024,),
                      verbose=True)

In [25]: 1 y_pred = clf.predict(X_test_trans_1)
2         print(classification_report(y_test, y_pred, target_names=target_names))

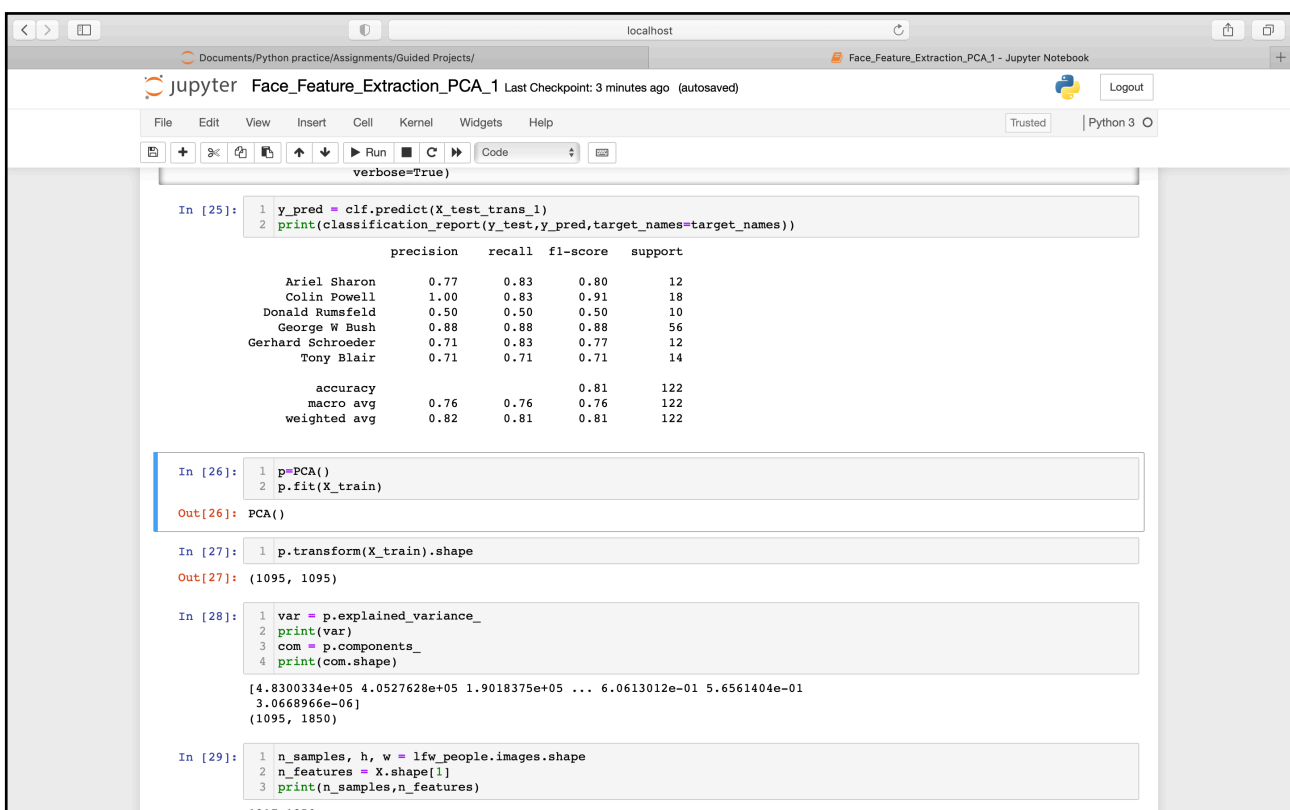
              precision    recall  f1-score   support

   Ariel Sharon         0.77       0.83       0.80         12
    Colin Powell         1.00       0.83       0.91         18
  Donald Rumsfeld         0.50       0.50       0.50         10
   George W Bush         0.88       0.88       0.88         56
  Gerhard Schroeder       0.71       0.83       0.77         12
    Tony Blair         0.71       0.71       0.71         14

   accuracy                   0.81         122
  macro avg                   0.76         122
 weighted avg                   0.81         122
```

Hidden layer size can be set to 128 or 512 to check the best f1 scores .

## PCA



The screenshot shows a Jupyter Notebook titled 'Face\_Feature\_Extraction\_PCA\_1'. The notebook is running on a local host. The code in the notebook is as follows:

```
verbose=True)

In [25]: 1 y_pred = clf.predict(X_test_trans_1)
2         print(classification_report(y_test, y_pred, target_names=target_names))

              precision    recall  f1-score   support

   Ariel Sharon         0.77       0.83       0.80         12
    Colin Powell         1.00       0.83       0.91         18
  Donald Rumsfeld         0.50       0.50       0.50         10
   George W Bush         0.88       0.88       0.88         56
  Gerhard Schroeder       0.71       0.83       0.77         12
    Tony Blair         0.71       0.71       0.71         14

   accuracy                   0.81         122
  macro avg                   0.76         122
 weighted avg                   0.81         122

In [26]: 1 p=PCA()
2         p.fit(X_train)

Out[26]: PCA()

In [27]: 1 p.transform(X_train).shape

Out[27]: (1095, 1095)

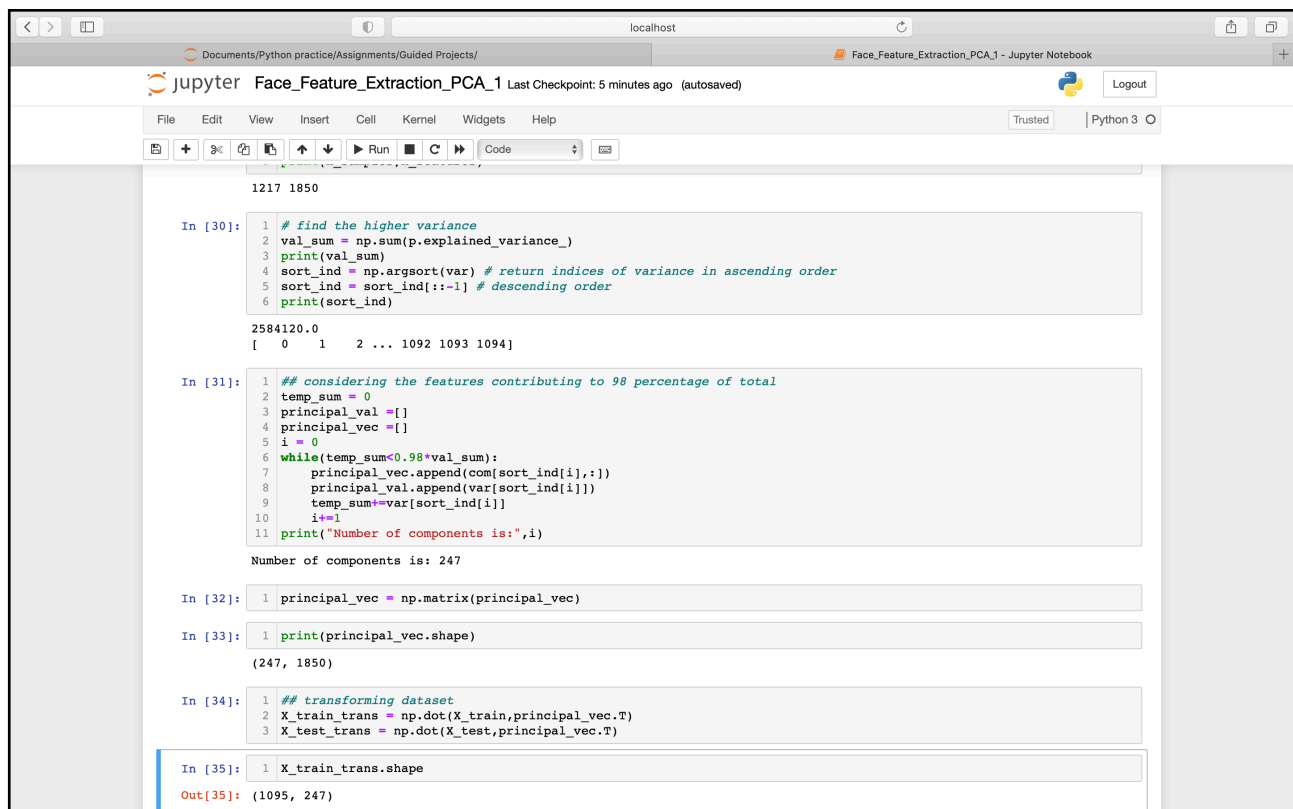
In [28]: 1 var = p.explained_variance_
2         print(var)
3         com = p.components_
4         print(com.shape)

[4.8300334e+05 4.0527628e+05 1.9018375e+05 ... 6.0613012e-01 5.6561404e-01
 3.0668966e-06]
(1095, 1850)

In [29]: 1 n_samples, h, w = lfw_people.images.shape
2         n_features = X.shape[1]
3         print(n_samples, n_features)

1217 1850
```

## Considering features which are having 98 percentage contribution



The screenshot shows a Jupyter Notebook titled 'Face\_Feature\_Extraction\_PCA\_1'. The notebook contains several code cells. The first cell shows the output of a previous cell: '1217 1850'. The second cell (In [30]) finds the higher variance components and returns their indices in ascending and descending order. The third cell (In [31]) iterates through the components, adding them to the principal vector until they account for 98% of the total variance. The fourth cell (In [32]) converts the principal vector to a matrix. The fifth cell (In [33]) prints the shape of the principal vector matrix, which is (247, 1850). The sixth cell (In [34]) transforms the dataset by dot-multiplying the training and testing data with the principal vector matrix. The seventh cell (In [35]) prints the shape of the transformed training data, which is (1095, 247).

```
1217 1850

In [30]: 1 # find the higher variance
        2 val_sum = np.sum(p.explained_variance_)
        3 print(val_sum)
        4 sort_ind = np.argsort(var) # return indices of variance in ascending order
        5 sort_ind = sort_ind[::-1] # descending order
        6 print(sort_ind)

2584120.0
[ 0  1  2 ... 1092 1093 1094]

In [31]: 1 ## considering the features contributing to 98 percentage of total
        2 temp_sum = 0
        3 principal_val = []
        4 principal_vec = []
        5 i = 0
        6 while(temp_sum<0.98*val_sum):
        7     principal_vec.append(com[sort_ind[i],:])
        8     principal_val.append(var[sort_ind[i]])
        9     temp_sum+=var[sort_ind[i]]
       10     i+=1
       11 print("Number of components is:",i)

Number of components is: 247

In [32]: 1 principal_vec = np.matrix(principal_vec)

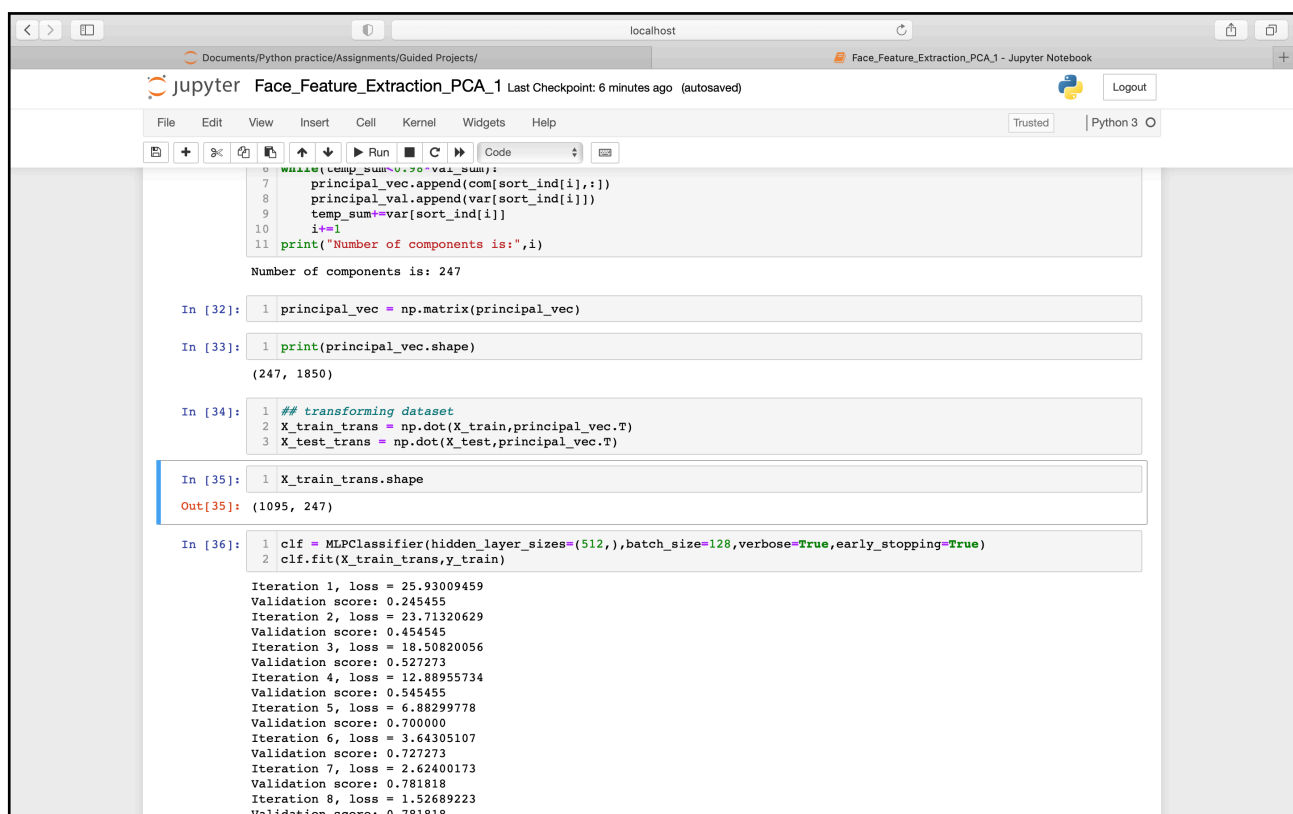
In [33]: 1 print(principal_vec.shape)

(247, 1850)

In [34]: 1 ## transforming dataset
        2 X_train_trans = np.dot(X_train,principal_vec.T)
        3 X_test_trans = np.dot(X_test,principal_vec.T)

In [35]: 1 X_train_trans.shape
Out[35]: (1095, 247)
```

## Classifying transformed data



The screenshot shows the same Jupyter Notebook as before, but with an additional code cell (In [36]) at the bottom. This cell imports the MLPClassifier from sklearn and fits it to the transformed training data. The output shows the training and validation loss and scores for 8 iterations.

```
0 while(temp_sum<0.98*val_sum):
    principal_vec.append(com[sort_ind[i],:])
    principal_val.append(var[sort_ind[i]])
    temp_sum+=var[sort_ind[i]]
    i+=1
11 print("Number of components is:",i)

Number of components is: 247

In [32]: 1 principal_vec = np.matrix(principal_vec)

In [33]: 1 print(principal_vec.shape)

(247, 1850)

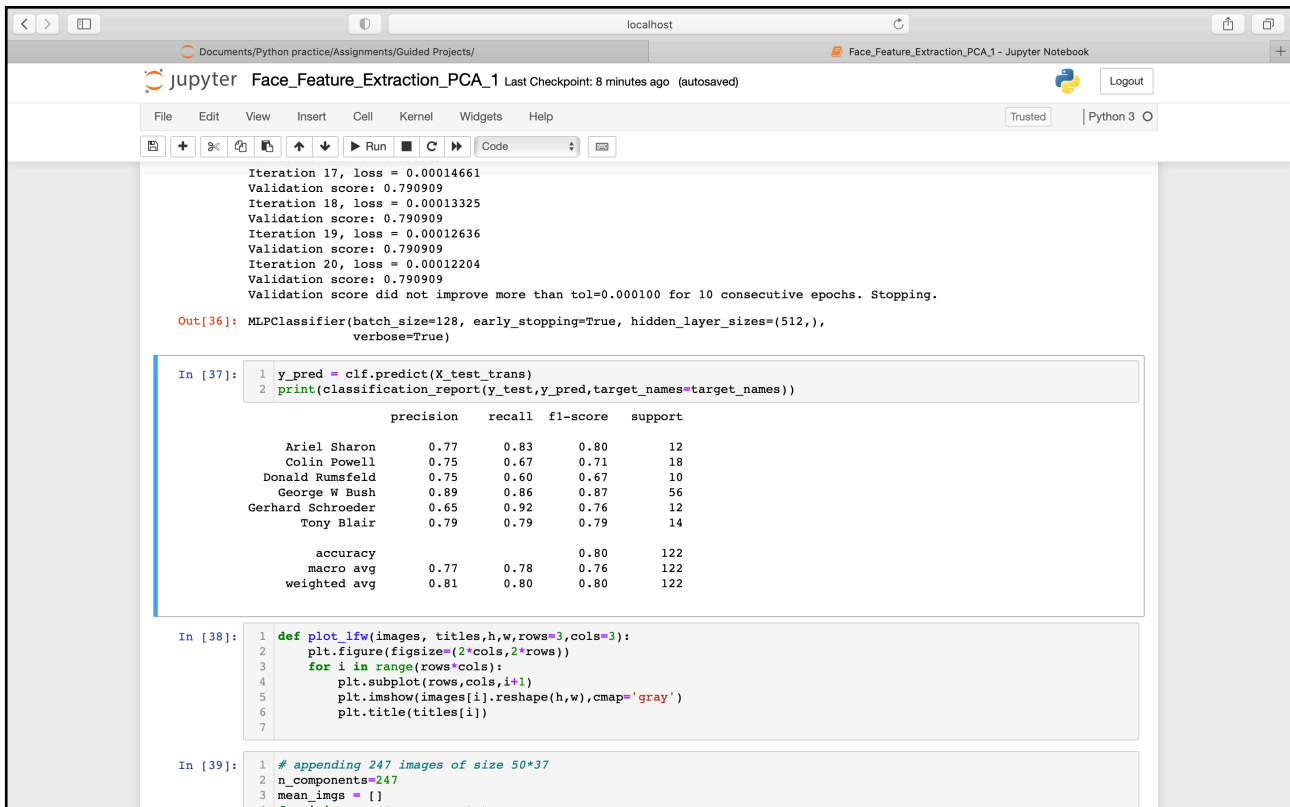
In [34]: 1 ## transforming dataset
        2 X_train_trans = np.dot(X_train,principal_vec.T)
        3 X_test_trans = np.dot(X_test,principal_vec.T)

In [35]: 1 X_train_trans.shape
Out[35]: (1095, 247)

In [36]: 1 clf = MLPClassifier(hidden_layer_sizes=(512,),batch_size=128,verbose=True,early_stopping=True)
        2 clf.fit(X_train_trans,y_train)

Iteration 1, loss = 25.93009459
Validation score: 0.245455
Iteration 2, loss = 23.71320629
Validation score: 0.454545
Iteration 3, loss = 18.50820056
Validation score: 0.527273
Iteration 4, loss = 12.88955734
Validation score: 0.545455
Iteration 5, loss = 6.88299778
Validation score: 0.700000
Iteration 6, loss = 3.64305107
Validation score: 0.727273
Iteration 7, loss = 2.62400173
Validation score: 0.781818
Iteration 8, loss = 1.52689223
Validation score: 0.781818
```

## Classification report



The screenshot shows a Jupyter Notebook interface with the following content:

```
Iteration 17, loss = 0.00014661
Validation score: 0.790909
Iteration 18, loss = 0.00013325
Validation score: 0.790909
Iteration 19, loss = 0.00012636
Validation score: 0.790909
Iteration 20, loss = 0.00012204
Validation score: 0.790909
Validation score did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
```

Out[36]: MLPClassifier(batch\_size=128, early\_stopping=True, hidden\_layer\_sizes=(512,), verbose=True)

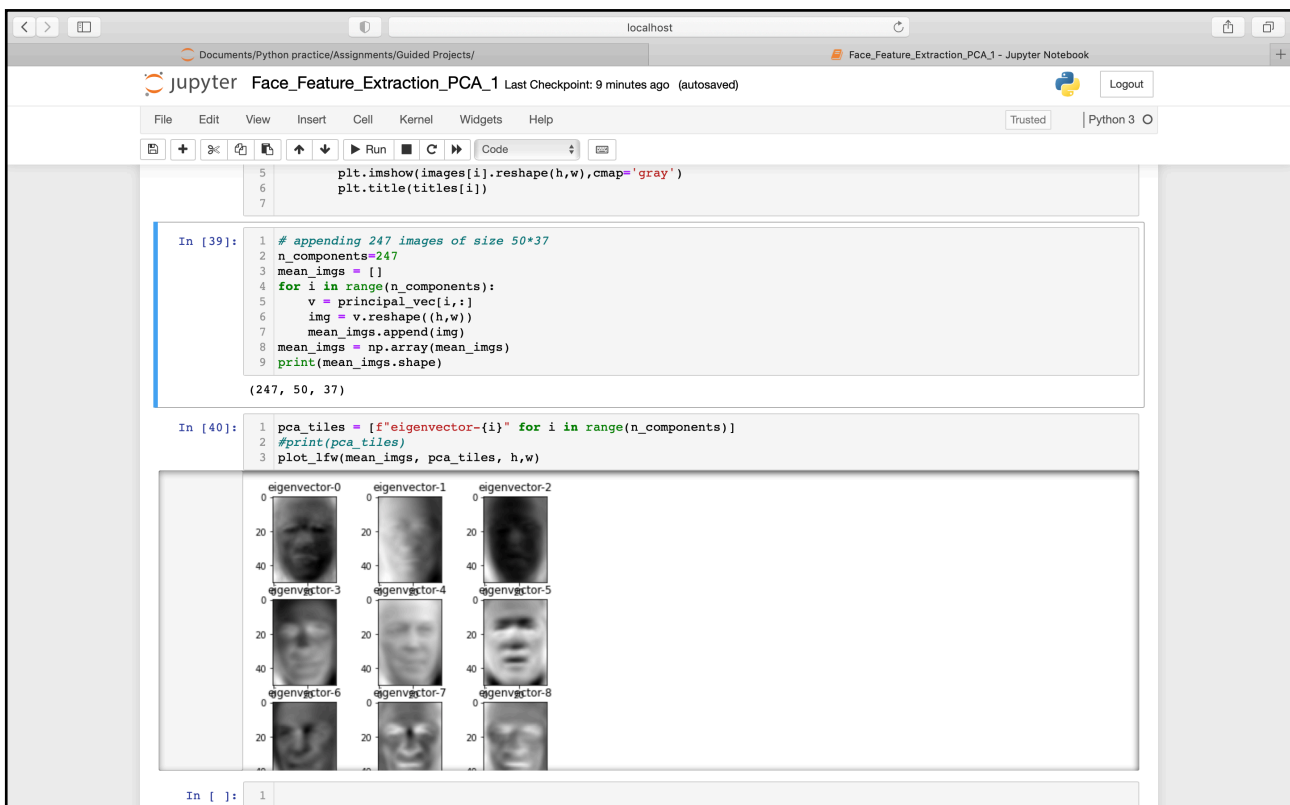
```
In [37]: 1 y_pred = clf.predict(X_test_trans)
2 print(classification_report(y_test,y_pred,target_names=target_names))
```

	precision	recall	f1-score	support
Ariel Sharon	0.77	0.83	0.80	12
Colin Powell	0.75	0.67	0.71	18
Donald Rumsfeld	0.75	0.60	0.67	10
George W Bush	0.89	0.86	0.87	56
Gerhard Schroeder	0.65	0.92	0.76	12
Tony Blair	0.79	0.79	0.79	14
accuracy			0.80	122
macro avg	0.77	0.78	0.76	122
weighted avg	0.81	0.80	0.80	122

```
In [38]: 1 def plot_lfw(images, titles,h,w,rows=3,cols=3):
2         plt.figure(figsize=(2*cols,2*rows))
3         for i in range(rows*cols):
4             plt.subplot(rows,cols,i+1)
5             plt.imshow(images[i].reshape(h,w),cmap='gray')
6             plt.title(titles[i])
7
```

```
In [39]: 1 # appending 247 images of size 50*37
2 n_components=247
3 mean_imgs = []
4 for i in range(n_components):
```

## Plotting the 247 features



The screenshot shows a Jupyter Notebook interface with the following content:

```
5 plt.imshow(images[i].reshape(h,w),cmap='gray')
6 plt.title(titles[i])
7
```

```
In [39]: 1 # appending 247 images of size 50*37
2 n_components=247
3 mean_imgs = []
4 for i in range(n_components):
5     v = principal_vec[i,:].
6     img = v.reshape((h,w))
7     mean_imgs.append(img)
8 mean_imgs = np.array(mean_imgs)
9 print(mean_imgs.shape)
```

(247, 50, 37)

```
In [40]: 1 pca_titles = [f'eigenvector-{i}' for i in range(n_components)]
2 #print(pca_titles)
3 plot_lfw(mean_imgs, pca_titles, h,w)
```

The output shows a grid of 9 grayscale images, each labeled with an eigenvector index (eigenvector-0 to eigenvector-8). These images represent the principal components of the face features.

