Department of Computer Science and Engineering (Data Science)

Lab Manual

Subject: Foundations of Data Analysis Laboratory (DJ19DSL303) Semester: III

Experiment 10 (Dimensionality Reduction)

NAME: Dev Patel **SAP ID: 60009200016**

BATCH: K/K1 **DATE: 18/01/2022**

Aim: Perform dimensionality reduction using Principal Component Analysis.

Theory:

Principal Component Analysis, or PCA, is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because smaller data sets are easier to explore and visualize and make analyzing data much easier and faster for machine learning algorithms without extraneous variables to process.

STEP 1: STANDARDIZATION

The aim of this step is to standardize the range of the continuous initial variables so that each one of them contributes equally to the analysis.

More specifically, the reason why it is critical to perform standardization prior to PCA, is that the latter is quite sensitive regarding the variances of the initial variables. That is, if there are large differences between the ranges of initial variables, those variables with larger ranges will dominate over those with small ranges (For example, a variable that ranges between 0 and 100 will dominate over a variable that ranges between 0 and 1), which will lead to biased results. So, transforming the data to comparable scales can prevent this problem.

Mathematically, this can be done by subtracting the mean and dividing by the standard deviation for each value of each variable.

$$z = \frac{value - mean}{standard\ deviation}$$

Once the standardization is done, all the variables will be transformed to the same scale.

STEP 2: COVARIANCE MATRIX COMPUTATION

The aim of this step is to understand how the variables of the input data set are varying from the mean with respect to each other, or in other words, to see if there is any relationship between them. Because sometimes, variables are highly correlated in such a way that they contain redundant information. So, in order to identify these correlations, we compute the covariance matrix.

The covariance matrix is a $p \times p$ symmetric matrix (where p is the number of dimensions) that has as entries the covariances associated with all possible pairs of the initial variables. For example, for a 3dimensional data set with 3 variables x, y, and z, the covariance matrix is a 3×3 matrix of this from:

$$\begin{bmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z) \end{bmatrix}$$

Covariance Matrix for 3-Dimensional Data

Since the covariance of a variable with itself is its variance (Cov(a,a)=Var(a)), in the main diagonal (Top left to bottom right) we actually have the variances of each initial variable. And since the covariance is commutative (Cov(a,b)=Cov(b,a)), the entries of the covariance matrix are symmetric with respect to the main diagonal, which means that the upper and the lower triangular portions are equal.

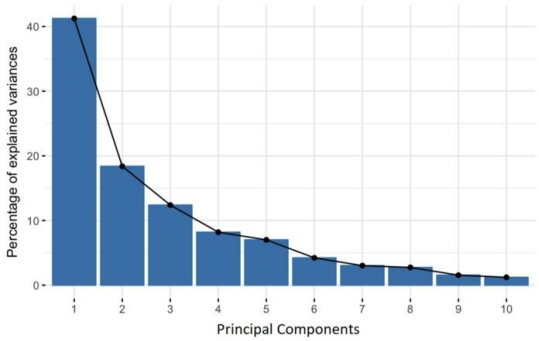
It's actually the sign of the covariance that matters:

- if positive then: the two variables increase or decrease together (correlated)
- if negative then: One increases when the other decreases (Inversely correlated)

STEP 3: COMPUTE THE EIGENVECTORS AND EIGENVALUES OF THE COVARIANCE MATRIX TO IDENTIFY THE PRINCIPAL COMPONENTS

Eigenvectors and eigenvalues are the linear algebra concepts that we need to compute from the covariance matrix in order to determine the *principal components* of the data. Before getting to the explanation of these concepts, let's first understand what do we mean by principal components.

Principal components are new variables that are constructed as linear combinations or mixtures of the initial variables. These combinations are done in such a way that the new variables (i.e., principal components) are uncorrelated and most of the information within the initial variables is squeezed or compressed into the first components. So, the idea is 10-dimensional data gives you 10 principal components, but PCA tries to put maximum possible information in the first component, then maximum remaining information in the second and so on, until having something like shown in the scree plot below.



Percentage of Variance (Information) for each by PC

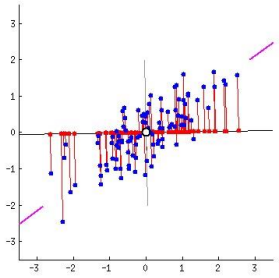
Organizing information in principal components this way, will allow you to reduce dimensionality without losing much information, and this by discarding the components with low information and considering the remaining components as your new variables.

An important thing to realize here is that, the principal components are less interpretable and don't have any real meaning since they are constructed as linear combinations of the initial variables.

Geometrically speaking, principal components represent the directions of the data that explain a maximal amount of variance, that is to say, the lines that capture most information of the data. The relationship between variance and information here, is that, the larger the variance carried by a line, the larger the dispersion of the data points along it, and the larger the dispersion along a line, the more the information it has. To put all this simply, just think of principal components as new axes that provide the best angle to see and evaluate the data, so that the differences between the observations are better visible.

How PCA Constructs the Principal Components

As there are as many principal components as there are variables in the data, principal components are constructed in such a manner that the first principal component accounts for the largest possible variance in the data set. For example, let's assume that the scatter plot of our data set is as shown below, can we guess the first principal component ? Yes, it's approximately the line that matches the purple marks because it goes through the origin and it's the line in which the projection of the points (red dots) is the most spread out. Or mathematically speaking, it's the line that maximizes the variance (the average of the squared distances from the projected points (red dots) to the origin).



The second principal component is calculated in the same way, with the condition that it is uncorrelated with (i.e., perpendicular to) the first principal component and that it accounts for the next highest variance.

This continues until a total of p principal components have been calculated, equal to the original number of variables.

Now that we understood what we mean by principal components, let's go back to eigenvectors and eigenvalues. What you firstly need to know about them is that they always come in pairs, so that every eigenvector has an eigenvalue. And their number is equal to the number of dimensions of the data. For example, for a 3-dimensional data set, there are 3 variables, therefore there are 3 eigenvectors with 3 corresponding eigenvalues.

Without further ado, it is eigenvectors and eigenvalues who are behind all the magic explained above, because the eigenvectors of the Covariance matrix are actually *the directions of the axes where there is the most variance*(most information) and that we call Principal Components. And eigenvalues are simply the coefficients attached to eigenvectors, which give the *amount of variance carried in each Principal Component*.

By ranking your eigenvectors in order of their eigenvalues, highest to lowest, you get the principal components in order of significance.

Example:

Let's suppose that our data set is 2-dimensional with 2 variables *x,y* and that the eigenvectors and eigenvalues of the covariance matrix are as follows:

$$v1 = \begin{bmatrix} 0.6778736 \\ 0.7351785 \end{bmatrix} \qquad \lambda_1 = 1.284028$$

$$v2 = \begin{bmatrix} -0.7351785 \\ 0.6778736 \end{bmatrix} \qquad \lambda_2 = 0.04908323$$

If we rank the eigenvalues in descending order, we get λ1>λ2, which means that the eigenvector that corresponds to the first principal component (PC1) is *v1* and the one that corresponds to the second component (PC2) is*v2.*

After having the principal components, to compute the percentage of variance (information) accounted for by each component, we divide the eigenvalue of each component by the sum of eigenvalues. If we apply this on the example above, we find that PC1 and PC2 carry respectively 96% and 4% of the variance of the data.

STEP 4: FEATURE VECTOR

As we saw in the previous step, computing the eigenvectors and ordering them by their eigenvalues in descending order, allow us to find the principal components in order of significance. In this step, what we do is, to choose whether to keep all these components or discard those of lesser significance (of low eigenvalues), and form with the remaining ones a matrix of vectors that we call *Feature vector*.

So, the feature vector is simply a matrix that has as columns the eigenvectors of the components that we decide to keep. This makes it the first step towards dimensionality reduction, because if we choose to keep only *p* eigenvectors (components) out of *n*, the final data set will have only *p* dimensions.

Example:

Continuing with the example from the previous step, we can either form a feature vector with both of the eigenvectors *v*1 and *v*2:

$$\begin{bmatrix} 0.6778736 & -0.7351785 \\ 0.7351785 & 0.6778736 \end{bmatrix}$$

Or discard the eigenvector *v*2, which is the one of lesser significance, and form a feature vector with *v*1 only:

$$\begin{bmatrix} 0.6778736 \\ 0.7351785 \end{bmatrix}$$

Discarding the eigenvector *v2* will reduce dimensionality by 1, and will consequently cause a loss of information in the final data set. But given that *v*2 was carrying only 4% of the information, the loss will be therefore not important and we will still have 96% of the information that is carried by *v*1.

LAST STEP: RECAST THE DATA ALONG THE PRINCIPAL COMPONENTS AXES

In the previous steps, apart from standardization, you do not make any changes on the data, you just select the principal components and form the feature vector, but the input data set remains always in terms of the original axes (i.e, in terms of the initial variables).

In this step, which is the last one, the aim is to use the feature vector formed using the eigenvectors of the covariance matrix, to reorient the data from the original axes to the ones represented by the principal components (hence the name Principal Components Analysis). This can be done by multiplying the transpose of the original data set by the transpose of the feature vector.

$$FinalDataSet = FeatureVector^T * StandardizedOriginalDataSet^T$$

Lab Assignments to complete in this session

1. **Randomly generate a small dataset and show dimensionality reduction using PCA by writing the code from scratch in python.**
   STEPS:

   a. We import the required libraries.
   b. We generate a small dataset.
   c. For dimensionality reduction, we find the mean centring the data.
   d. Then we calculate the covariance matrix of the mean-centred data.
   e. After that, we calculate the Eigen-values and Eigen-vectors of the covariance matrix.
   f. We sort the Eigenvalues in descending order. Similarly, we sort Eigen-vectors.
   g. We select the first n Eigen-vectors, where n is the desired dimension of the final reduced data.
   h. Finally, we transform the data.

   CODE:

```
import numpy as np #We import the required libraries.

def PCA(X , num_components):

    X_meaned = X - np.mean(X , axis = 0)      # We find the mean centri
ng the data.

    cov_mat = np.cov(X_meaned , rowvar = False)    # Then we calculate
the covariance matrix of the mean-centred data.

    eigen_values , eigen_vectors = np.linalg.eigh(cov_mat)  # We calcu
late the Eigen-values and Eigen-vectors of the covariance matrix.

    # We sort the Eigenvalues in descending order. Similarly, we sort
Eigen-vectors.
```

```
    sorted_index = np.argsort(eigen_values)[::-1]
    sorted_eigenvalue = eigen_values[sorted_index]
    sorted_eigenvectors = eigen_vectors[:,sorted_index]

    # We select the first num_components Eigen-
vectors, where num_components is the desired dimension of the final re
duced data.
    eigenvector_subset = sorted_eigenvectors[:,0:num_components]

    # We transform the data
    X_reduced = np.dot(eigenvector_subset.transpose() , X_meaned.trans
pose() ).transpose()

    return X_reduced

X = np.random.randint(10,50,100).reshape(20,5)  # We generate a small
dataset.
print(X,"\n")
print(PCA(X,3))
```

OUTPUT:

```
[[13 43 42 44 42]
 [23 28 14 45 47]
 [14 11 29 36 19]
 [11 15 27 23 26]
 [19 18 34 26 10]
 [31 38 11 40 38]
 [19 20 42 24 26]
 [25 44 32 49 32]
 [27 18 34 34 40]
 [25 19 32 45 46]
 [10 45 13 26 22]
 [29 43 10 42 11]
 [30 36 29 24 34]
 [11 10 33 17 43]
 [31 12 20 19 30]
 [16 11 13 38 38]
 [28 36 26 42 30]
 [12 33 30 36 48]
 [46 43 13 46 13]
 [31 16 16 41 37]]

[[  2.91873937  20.39012901 -19.73557746]
 [  5.15958295  15.42155083  12.47426568]
 [-12.17433601 -12.41989275  -0.41679405]
 [-16.90166147 -10.71094469  -2.93285528]
 [ -9.05311886 -22.11869622  -9.47277302]
```

```
[ 16.44712302   6.55850237   9.07620556]
[-15.39763708  -7.10861535 -12.01502481]
[ 16.2223457   10.87713434 -10.50751305]
[-10.41218915   6.1959156    2.93137414]
[ -6.67480739  15.77558658   6.24548165]
[ 11.23352233  -8.81468752 -11.29293301]
[ 27.29129779 -14.85465756  -0.49603587]
[  2.58184334  -0.15292417  -5.71666623]
[-29.39525526   1.62500941  -0.71171081]
[-11.86212607 -12.79545454  11.97273662]
[ -9.69457238   1.54373246  16.11686325]
[ 11.4148326    3.14232534  -2.60576981]
[ -5.06258529  17.9679178   -6.56893811]
[ 33.59746828 -12.62478591   5.78844356]
[ -0.23846642   2.10285498  17.86722104]]
```

2. **Implement above code and reduce the dimensions of IRIS dataset from 4 attribute to 2 attributes.**

STEPS:

a. Get the IRIS dataset from the url https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data
b. Prepare the data. Also, prepare the target.
c. We apply the data to our PCA function.
d. Then we create a dataframe using Pandas of the reduced dataset.
e. We concatenate it with the target variable to create a complete dataset.

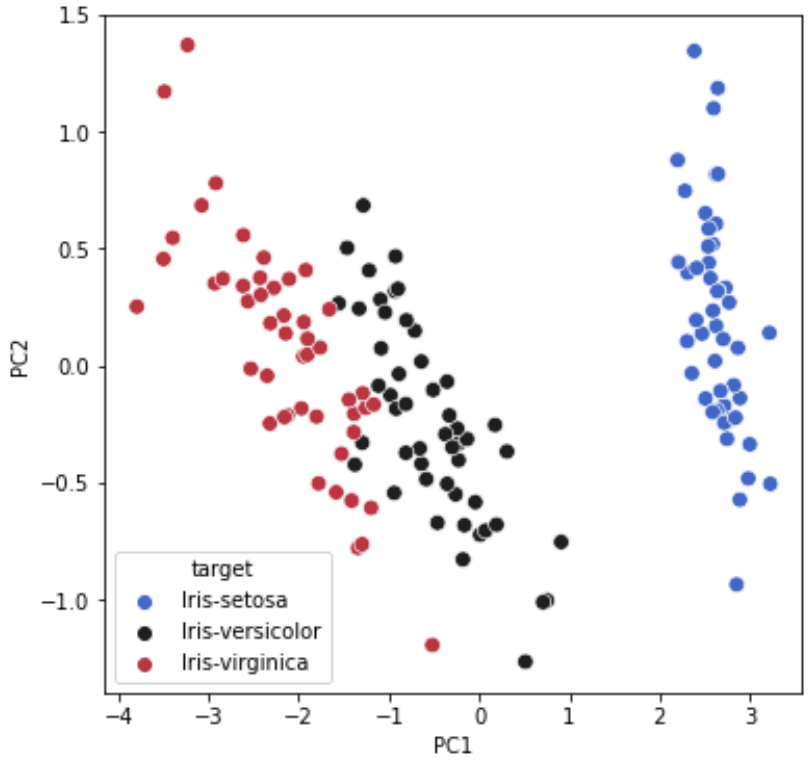
CODE:

```
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt

# Get the IRIS dataset from url
url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data"
data = pd.read_csv(url, names=['sepal length','sepal width','petal len
gth','petal width','target'])

# Prepare the data
x = data.iloc[:,0:4]

# Prepare the target
target = data.iloc[:,4]
```

```
# We apply it to our PCA function
mat_reduced = PCA(x , 2)

# Then we create a Pandas DataFrame of reduced Dataset
principal_df = pd.DataFrame(mat_reduced , columns = ['PC1','PC2'])

# We concatenate it with the target variable to create a complete data
set.
principal_df = pd.concat([principal_df , pd.DataFrame(target)] , axis
= 1)

plt.figure(figsize = (6,6))
sb.scatterplot(data = principal_df , x = 'PC1',y = 'PC2' , hue = 'targ
et' , s = 60 , palette= 'icefire')
```

OUTPUT:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb336911cd0>
```



3. **Use PCA code from sklearn.decomposition library and perform compression on your image.**

Code for Step-1:
(We import required libraries, load and obtain the dataset)

```
# Import required libraries
from __future__ import print_function

import numpy as np
from sklearn.decomposition import PCA
from sklearn.datasets import fetch_olivetti_faces
import matplotlib.pyplot as plt
%matplotlib inline
# Load dataset
#We will use the Olivetti Faces dataset. It has 10 faces each of 40 pe
rsons as 64x64 images.

# Get Olivetti Faces dataset
faces_data = fetch_olivetti_faces()
n_samples, height, width = faces_data.images.shape
X = faces_data.data
n_features = X.shape[1]
y = faces_data.target
n_classes = int(max(y)+1)

print("Number of samples: {}, \nHeight of each image: {}, \nWidth of e
ach image: {}, \nNumber of input features: {},\nNumber of output class
es: {}\n".format(n_samples,height,width,n_features,n_classes))
```

Output of Step-1:

```
Number of samples: 400,
Height of each image: 64,
Width of each image: 64,
Number of input features: 4096,
Number of output classes: 40
```

Code for Step-2:
(We shuffle the data randomly and split it into training set and test set)

```
from sklearn.model_selection import train_test_split
# Shuffle the data randomly and make train and test splits
# Split into a training set (75%) and a test set (25%)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
25,       random_state=42)
mean_image = np.mean(X_train,axis=0)
plt.figure
plt.imshow(mean_image.reshape((64,64)), cmap=plt.cm.gray)
plt.xticks(())
plt.yticks(())
plt.show()
```

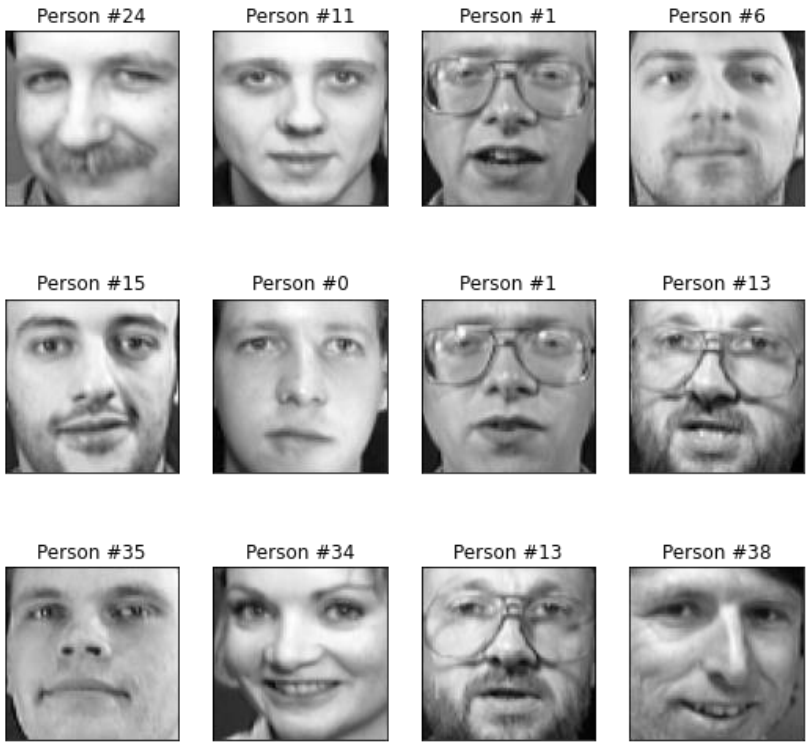
Output of Step-2:

Code for Step-3:
(We write a function for visualisation of the images as an album and visualise some faces from the dataset)

```
# Make a function for visualization of the images as an album
def plot_gallery(images, h, w, titles=None, n_row=3, n_col=4):
    """
    Helper function to plot a gallery of portraits
    Taken from: http://scikit-
learn.org/stable/auto_examples/applications/face_recognition.html
    """
```

```
    plt.figure(figsize=(1.8 * n_col, 2.4 * n_row))
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace
=.35)
    for i in range(n_row * n_col):
        plt.subplot(n_row, n_col, i + 1)
        plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
        if titles != None:
            plt.title(titles[i], size=12)
        plt.xticks(())
        plt.yticks(())
# Visualize some faces from the training set
chosen_images = X_train[:12]
chosen_labels = y_train[:12]
titles = ['Person #'+str(i) for i in chosen_labels]

plot_gallery(chosen_images, height, width, titles)
```

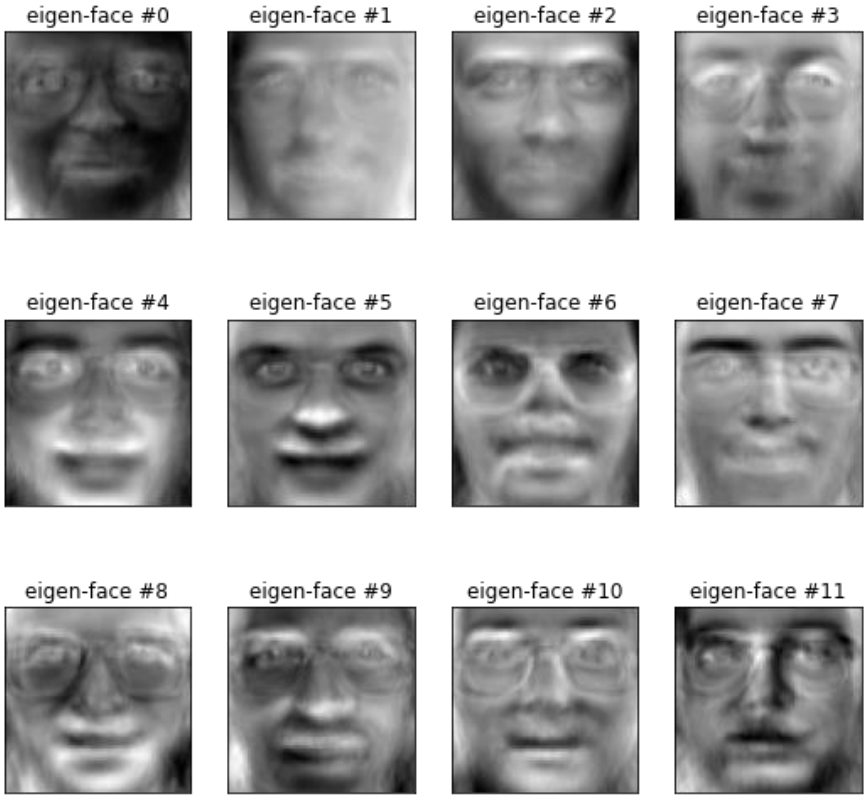
Output of Step-3:



Code for Step-4: (We calculate set of eigen-faces, reduce the dimensionality of feature space and visualise them as explained in detail below)

```
# Calculate a set of eigen-faces
''' We find the eigen vectors corresponding to the biggest eigen value
s of the covariance matrix of the data.
These eigen vectors are the directions along which the data shows maxi
mum amount of variation.
Each eigen vector can be considered as an eigen face.
We can represent any image in the dataset as a linear combination of t
hese eigen faces with minimum error.'''

#Reduce the dimensionality of the feature space
n_components = 150

#Finding the top n_components principal components in the data
pca = PCA(n_components=n_components, whiten=True).fit(X_train)

#Find the eigen-vectors of the feature space
eigenfaces = pca.components_.reshape((n_components, height, width))
# Visualize the eigen faces

titles = ['eigen-face #'+str(i) for i in range(12)]
plot_gallery(eigenfaces, height, width, titles)
# Transform the data to the vector space spanned by the eigen faces
# Projecting the data onto the eigenspace
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

print("Current shape of input data matrix: ", X_train_pca.shape)
```

Output of Step-4:




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

4. **Use PCA code from sklearn.decomposition library and perform dimensionality reduction on olivetti_face dataset**

Code:

```
# Compressing one Image
import matplotlib.image as mpimg
from sklearn.decomposition import PCA
img = mpimg.imread('/content/drive/MyDrive/Datasets/fda10.jfif')
#Now, let's look at the size of this numpy array object img as well as p
lot it using imshow.
print (img.shape)
plt.axis('off')
plt.imshow(img)
```

```
from google.colab import drive
drive.mount('/content/drive')
#Okay, so the array has 800 rows each of pixel 1200x3. Let's reshape it
into a format that PCA can understand. # 3600 = 1200 * 3
img_r = np.reshape(img, (167, 906))
print (img_r.shape)
# Now lets run RandomizedPCA with 64 components (8x8 pixels) and transfo
rm the image.
ipca = PCA(167).fit(img_r)
img_c = ipca.transform(img_r)
print (img_c.shape)
print (np.sum(ipca.explained_variance_ratio_))
#OK, now to visualize how PCA has performed this compression, let's inve
rse transform the PCA output and #reshape for visualization using imshow
.
temp = ipca.inverse_transform(img_c)
#reshaping 2988 back to the original 1200 * 3
temp = np.reshape(temp, (167,302,3))
#Great, now lets visualize like before with imshow
plt.axis('off')
plt.imshow(temp)
```

Output:

```
(167, 302, 3)
<matplotlib.image.AxesImage at 0x7fe8e5ef6d50>
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).

Reshaped image:  (167, 906)
Transformed image shape:  (167, 167)
IPCA variance ratio = 1.0
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
<matplotlib.image.AxesImage at 0x7fe8e60dcf90>
```