# Q4. Eigenfaces - Face classification using PCA (40 classes)

a) Use the following "face.csv" file to classify the faces of 40 different people.

b) Do not use in-built function for implementing PCA.

c) Use appropriate classifier taught in class (any classification algorithm taught in class like Bayes classifier, minimum distance clasifier, and so on)

d) Refer to the following link for a description of the dataset

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
```

In [3]:

```
df = pd.read_csv('face.csv')
df.drop(df.columns[-1], axis = 1, inplace = True)
df
```

Out[3]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.309917 | 0.367769 | 0.417355 | 0.442149 | 0.528926 | 0.607438 | 0.657025 | 0.677686 | 0.690083 |
| **1** | 0.454545 | 0.471074 | 0.512397 | 0.557851 | 0.595041 | 0.640496 | 0.681818 | 0.702479 | 0.710744 |
| **2** | 0.318182 | 0.400826 | 0.491736 | 0.528926 | 0.586777 | 0.657025 | 0.681818 | 0.685950 | 0.702479 |
| **3** | 0.198347 | 0.194215 | 0.194215 | 0.194215 | 0.190083 | 0.190083 | 0.243802 | 0.404959 | 0.483471 |
| **4** | 0.500000 | 0.545455 | 0.582645 | 0.623967 | 0.648760 | 0.690083 | 0.694215 | 0.714876 | 0.723140 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **395** | 0.400826 | 0.495868 | 0.570248 | 0.632231 | 0.648760 | 0.640496 | 0.661157 | 0.636364 | 0.665289 |
| **396** | 0.367769 | 0.367769 | 0.351240 | 0.301653 | 0.247934 | 0.247934 | 0.367769 | 0.512397 | 0.574380 |
| **397** | 0.500000 | 0.533058 | 0.607438 | 0.628099 | 0.657025 | 0.632231 | 0.657025 | 0.669422 | 0.673554 |
| **398** | 0.214876 | 0.219008 | 0.219008 | 0.223140 | 0.210744 | 0.202479 | 0.276859 | 0.400826 | 0.487603 |
| **399** | 0.516529 | 0.462810 | 0.280992 | 0.252066 | 0.247934 | 0.367769 | 0.574380 | 0.615702 | 0.661157 |

400 rows × 4096 columns

In [5]:

```
train_data_set = df.iloc[2:10]

for i in range(10, 400, 10):
  train_data_set = train_data_set.append(df.iloc[i + 2:i + 10], ignore_index = True)
```

In [6]:

```
test_data_set = df.iloc[0:2]

for i in range(10, 400, 10):
  test_data_set = test_data_set.append(df.iloc[i:i + 2], ignore_index = True)
```

In [7]:

```
train_data_set = np.transpose(train_data_set)

test_data_set = np.transpose(test_data_set)

d = 4096
```

In [8]:

```
def mean_PCA(train_data_set):
    train_data_set_mean = np.mean(train_data_set, axis = 1)
    return np.array(train_data_set_mean)
```

In [9]:

```
train_data_set_mean = mean_PCA(train_data_set)

train_data_set_covariance_matrix = np.cov(train_data_set)

eigen_values, eigen_vectors = np.linalg.eigh(train_data_set_covariance_matrix)
```

In [10]:

```
eigen_vectors
```

Out[10]:

```
array([[ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00, ...,
         4.07541819e-04,  2.90708397e-02, -2.74754798e-03],
       [ 1.41323302e-01,  1.47697747e-02, -3.20600723e-01, ...,
        -1.51739158e-03,  3.42810801e-02, -5.43886813e-03],
       [ 4.52142226e-01,  2.75796727e-01,  3.75162941e-01, ...,
        -2.33185371e-03,  3.90819016e-02, -7.60074745e-03],
       ...,
       [-1.08510497e-02, -2.50947201e-02, -6.58641110e-03, ...,
        -1.30268290e-02, -2.99621494e-02, -2.94621031e-03],
       [ 1.45648223e-03,  2.43254843e-02,  3.17920811e-03, ...,
        -1.02592154e-02, -2.74033230e-02,  8.39296650e-04],
       [-6.55733247e-03,  1.23748159e-02, -1.40301892e-02, ...,
        -8.67573068e-03, -2.50393131e-02,  1.56090205e-03]])
```

In [12]:

```
eigen_pairs = [(np.abs(eigen_values[i]), eigen_vectors[:, i]) for i in range(len(eigen_valu

eigen_pairs.sort(key = lambda x : x[0], reverse = True)

eigen_pairs
```

Out[12]:

```
[(19.105796095321377,
  array([-0.00274755, -0.00543887, -0.00760075, ..., -0.00294621,
          0.0008393 ,  0.0015609 ])),
 (12.207087351051666,
  array([ 0.02907084,  0.03428108,  0.0390819 , ..., -0.02996215,
         -0.02740332, -0.02503931])),
 (6.166659790621014,
  array([ 0.00040754, -0.00151739, -0.00233185, ..., -0.01302683,
         -0.01025922, -0.00867573])),
 (3.766708963990769,
  array([-0.00253901, -0.00561563, -0.01419596, ...,  0.031047  ,
          0.02984297,  0.02726496])),
 (2.730287928495426,
  array([0.02352622, 0.02307565, 0.02036678, ..., 0.03885212, 0.03756989,
         0.0392973 ])),
 (2.491505137016736,
  array([-0.02104077, -0.0211751 , -0.01811579, ...,  0.0269678 ,
          0.02674315,  0.02640275])),
```

In [13]:

```
def get_reduced_dimension(eigen_pairs, eigen_values, percent):
    eigen_values_sum = np.sum(eigen_values)
    eigen_values_sum_reduced_dimension = percent * eigen_values_sum / 100
    sum = 0.0
    count = 0
    for i in eigen_pairs:
        if(sum < eigen_values_sum_reduced_dimension):
            sum += i[0]
            count += 1
        else:
            break
    return count
```

In [14]:

```
d_prime = get_reduced_dimension(eigen_pairs, eigen_values, 95)
d_prime
```

Out[14]:

```
111
```

In [15]:

```
matrix_w = eigen_pairs[0][1].reshape(d, 1)
```

In [16]:

```
for i in range(1, d_prime):
    matrix_w = np.hstack((matrix_w, eigen_pairs[i][1].reshape(d, 1)))
```

In [17]:

```
train_data_set_transformed = np.transpose(matrix_w.T.dot(train_data_set))

test_data_set_transformed = np.transpose(matrix_w.T.dot(test_data_set))
```

In [18]:

```
train_data_set_transformed
```

Out[18]:

```
array([[-3.84278500e+01,  1.05496914e+01,  2.76257385e+00, ...,
        -1.49311834e-01,  2.64169294e-01,  4.94872391e-01],
       [-3.80958302e+01, -2.36530844e+00,  2.77395344e+00, ...,
         1.07242465e-01, -2.50675502e-01,  2.44819087e-01],
       [-3.70319559e+01,  1.22432369e+01,  6.31392191e+00, ...,
        -1.36547366e-01, -4.01670809e-02,  3.49316185e-01],
       ...,
       [-3.11444304e+01,  1.01462209e+01,  5.96225995e+00, ...,
        -6.70871573e-02, -1.75148272e-01,  3.32349524e-01],
       [-3.88338294e+01,  9.01930153e-01,  1.98477303e+00, ...,
        -2.30756990e-02, -3.69142681e-01,  3.00644319e-01],
       [-3.45162687e+01,  6.32423245e+00,  3.80672052e+00, ...,
        -3.53049751e-01,  2.48143830e-01,  2.12767436e-01]])
```

In [19]:

```
test_data_set_transformed
```

Out[19]:

```
array([[-3.95811272e+01,  9.73374957e+00,  1.16472961e+00, ...,
         1.14704260e-02,  2.30335788e-02,  1.73677045e-01],
       [-3.39897567e+01,  1.54595402e+01,  4.52888350e+00, ...,
        -4.59040975e-01, -1.37985001e-01,  4.52161510e-01],
       [-3.42602930e+01,  9.76087593e+00, -1.34650656e+00, ...,
        -1.48750983e-01,  2.39329589e-01,  3.29394425e-01],
       ...,
       [-2.30611479e+01,  9.42401369e+00,  1.71149373e+00, ...,
        -8.64847903e-02, -4.13596812e-01,  5.43861931e-01],
       [-3.18795206e+01,  1.02691521e+01,  5.97999216e+00, ...,
        -2.37681112e-02,  6.24611657e-02,  2.73599124e-01],
       [-3.73080804e+01,  6.39686729e+00,  4.28431021e+00, ...,
        -2.95259371e-01, -4.42425011e-01,  2.70357075e-01]])
```

# Bayes Classifier Code

## Bayes Classifier with dimesion = 4096

In [20]:

```
train_data_set
```

Out[20]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.318182 | 0.198347 | 0.500000 | 0.549587 | 0.330578 | 0.128099 | 0.243802 | 0.380165 | 0.657025 |
| **1** | 0.400826 | 0.194215 | 0.545455 | 0.545455 | 0.305785 | 0.185950 | 0.297521 | 0.442149 | 0.677686 |
| **2** | 0.491736 | 0.194215 | 0.582645 | 0.541322 | 0.330578 | 0.247934 | 0.367769 | 0.483471 | 0.698347 |
| **3** | 0.528926 | 0.194215 | 0.623967 | 0.537190 | 0.351240 | 0.314050 | 0.454545 | 0.545455 | 0.706612 |
| **4** | 0.586777 | 0.190083 | 0.648760 | 0.537190 | 0.425620 | 0.388430 | 0.495868 | 0.582645 | 0.702479 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | .. |
| **4091** | 0.148760 | 0.743802 | 0.177686 | 0.661157 | 0.487603 | 0.157025 | 0.177686 | 0.173554 | 0.115702 |
| **4092** | 0.144628 | 0.764463 | 0.177686 | 0.690083 | 0.190083 | 0.152893 | 0.190083 | 0.173554 | 0.115702 |
| **4093** | 0.140496 | 0.752066 | 0.177686 | 0.714876 | 0.144628 | 0.152893 | 0.190083 | 0.173554 | 0.115702 |
| **4094** | 0.148760 | 0.752066 | 0.173554 | 0.706612 | 0.152893 | 0.173554 | 0.181818 | 0.173554 | 0.115702 |
| **4095** | 0.152893 | 0.739669 | 0.173554 | 0.702479 | 0.152893 | 0.173554 | 0.190083 | 0.173554 | 0.103306 |

4096 rows × 320 columns

In [21]:

```
test_data_set
```

Out[21]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.309917 | 0.454545 | 0.541322 | 0.644628 | 0.578512 | 0.628099 | 0.169422 | 0.561983 | 0.454545 |
| **1** | 0.367769 | 0.471074 | 0.586777 | 0.690083 | 0.603306 | 0.665289 | 0.264463 | 0.665289 | 0.429752 |
| **2** | 0.417355 | 0.512397 | 0.640496 | 0.702479 | 0.632231 | 0.685950 | 0.219008 | 0.698347 | 0.537190 |
| **3** | 0.442149 | 0.557851 | 0.661157 | 0.702479 | 0.665289 | 0.694215 | 0.280992 | 0.698347 | 0.611570 |
| **4** | 0.528926 | 0.595041 | 0.685950 | 0.706612 | 0.677686 | 0.719008 | 0.421488 | 0.731405 | 0.652893 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | .. |
| **4091** | 0.132231 | 0.152893 | 0.363636 | 0.111570 | 0.194215 | 0.177686 | 0.363636 | 0.231405 | 0.376033 |
| **4092** | 0.148760 | 0.152893 | 0.111570 | 0.115702 | 0.165289 | 0.161157 | 0.198347 | 0.214876 | 0.409091 |
| **4093** | 0.152893 | 0.152893 | 0.095041 | 0.111570 | 0.177686 | 0.185950 | 0.210744 | 0.219008 | 0.409091 |
| **4094** | 0.161157 | 0.152893 | 0.111570 | 0.107438 | 0.161157 | 0.161157 | 0.235537 | 0.219008 | 0.384298 |
| **4095** | 0.157025 | 0.152893 | 0.111570 | 0.119835 | 0.152893 | 0.190083 | 0.214876 | 0.227273 | 0.384298 |

4096 rows × 80 columns

In [22]:

```
train_data_set = np.transpose(train_data_set)
test_data_set = np.transpose(test_data_set)

X_train = train_data_set
y_train = []

for i in range(40):
    for j in range(int(len(train_data_set)/ 40)):
        y_train.append(str(i))
```

In [23]:

```
X_test = test_data_set
y_test = []

for i in range(40):
    for j in range(int(len(test_data_set)/ 40)):
        y_test.append(str(i))
```

In [24]:

```
gnb = GaussianNB()

gnb.fit(X_train, y_train)

y_pred = gnb.predict(X_test)

overall_accuracy = metrics.accuracy_score(y_test, y_pred)
```

In [25]:

```
print("Bayes Classifier with dimension = " + str(d))
print("\nOverall Accuracy = " + str(overall_accuracy))
```

```
Bayes Classifier with dimension = 4096

Overall Accuracy = 0.9
```

## Bayes Classifier with dimesion = 111

In [26]:

```
X_train_reduced_dimension = train_data_set_transformed
y_train_reduced_dimension = []

for i in range(40):
    for j in range(int(len(train_data_set_transformed)/ 40)):
        y_train_reduced_dimension.append(str(i))

X_train_reduced_dimension
```

Out[26]:

```
array([[-3.84278500e+01,  1.05496914e+01,  2.76257385e+00, ...,
        -1.49311834e-01,  2.64169294e-01,  4.94872391e-01],
       [-3.80958302e+01, -2.36530844e+00,  2.77395344e+00, ...,
         1.07242465e-01, -2.50675502e-01,  2.44819087e-01],
       [-3.70319559e+01,  1.22432369e+01,  6.31392191e+00, ...,
        -1.36547366e-01, -4.01670809e-02,  3.49316185e-01],
       ...,
       [-3.11444304e+01,  1.01462209e+01,  5.96225995e+00, ...,
        -6.70871573e-02, -1.75148272e-01,  3.32349524e-01],
       [-3.88338294e+01,  9.01930153e-01,  1.98477303e+00, ...,
        -2.30756990e-02, -3.69142681e-01,  3.00644319e-01],
       [-3.45162687e+01,  6.32423245e+00,  3.80672052e+00, ...,
        -3.53049751e-01,  2.48143830e-01,  2.12767436e-01]])
```

In [27]:

```
X_train_reduced_dimension.shape
```

Out[27]:

```
(320, 111)
```

In [28]:

```
X_test_reduced_dimension = test_data_set_transformed
y_test_reduced_dimension = []

for i in range(40):
    for j in range(int(len(test_data_set_transformed)/ 40)):
        y_test_reduced_dimension.append(str(i))

X_test_reduced_dimension
```

Out[28]:

```
array([[-3.95811272e+01,  9.73374957e+00,  1.16472961e+00, ...,
         1.14704260e-02,  2.30335788e-02,  1.73677045e-01],
       [-3.39897567e+01,  1.54595402e+01,  4.52888350e+00, ...,
        -4.59040975e-01, -1.37985001e-01,  4.52161510e-01],
       [-3.42602930e+01,  9.76087593e+00, -1.34650656e+00, ...,
        -1.48750983e-01,  2.39329589e-01,  3.29394425e-01],
       ...,
       [-2.30611479e+01,  9.42401369e+00,  1.71149373e+00, ...,
        -8.64847903e-02, -4.13596812e-01,  5.43861931e-01],
       [-3.18795206e+01,  1.02691521e+01,  5.97999216e+00, ...,
        -2.37681112e-02,  6.24611657e-02,  2.73599124e-01],
       [-3.73080804e+01,  6.39686729e+00,  4.28431021e+00, ...,
        -2.95259371e-01, -4.42425011e-01,  2.70357075e-01]])
```

In [29]:

```
X_test_reduced_dimension.shape
```

Out[29]:

```
(80, 111)
```

In [30]:

```
gnb_reduced_dimension = GaussianNB()

gnb_reduced_dimension.fit(X_train_reduced_dimension, y_train_reduced_dimension)

y_pred_reduced_dimension = gnb_reduced_dimension.predict(X_test_reduced_dimension)

overall_accuracy_reduced_dimension = metrics.accuracy_score(y_test_reduced_dimension, y_pre

print("Bayes Classifier with dimension = " + str(d_prime))
print("\nOverall Accuracy = " + str(overall_accuracy_reduced_dimension))
```

```
Bayes Classifier with dimension = 111

Overall Accuracy = 0.975
```

In [ ]:

```

```