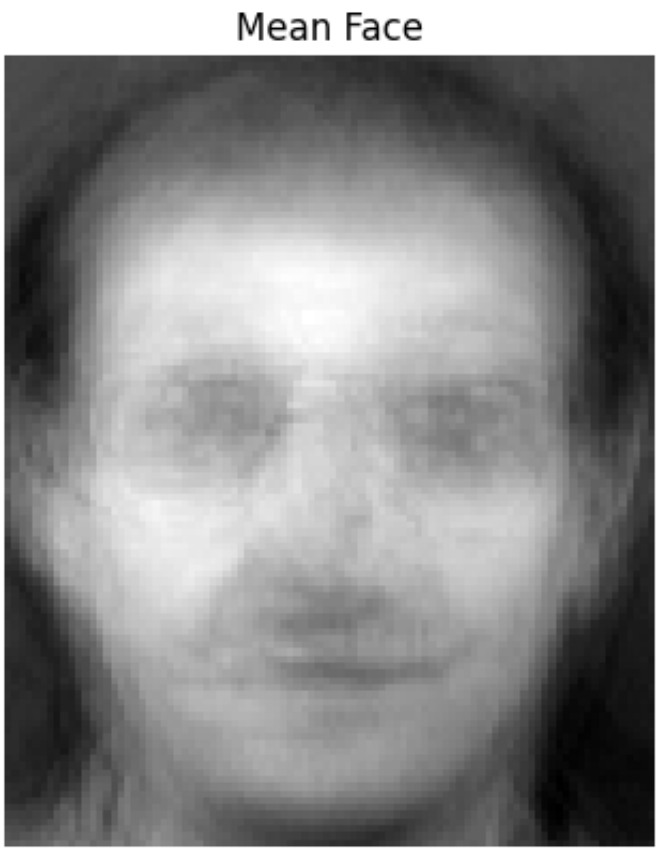
```
# Project 11: Projections, eigenvectors, Principal Component Analysis, and face
import numpy as np
from PIL import Image
import os
import numpy as np
from PIL import Image
import os
import matplotlib.pyplot as plt


print("\nSubtask 1\n")
# Parameters
Database_Size = 30
database_path = "/content/database"
# Initialize list to store image vectors
P = []
# Reading images from the database
for j in range(1, Database_Size + 1):
    image_path = os.path.join(database_path, f'person{j}.pgm')
    image = Image.open(image_path)
    image_array = np.array(image)
    # Get dimensions of the image
    m, n = image_array.shape
    # Reshape the image array to a column vector
    image_vector = image_array.reshape(m * n, 1)
    P.append(image_vector)
# Convert list to numpy array (matrix)
P = np.hstack(P)
# Print out the variables for verification
print(f"Database Size: {Database_Size}")
print(f"Image dimensions (m, n): ({m}, {n})")
print(f"P matrix shape: {P.shape}")
```

```
Subtask 1

Database Size: 30
Image dimensions (m, n): (112, 92)
P matrix shape: (10304, 30)
```

```
print("\nSubtask 2\n")
# Compute the mean face
mean_face = np.mean(P, axis=1)
# Reshape the mean face back to the original image dimensions
mean_face_image = mean_face.reshape(m, n)
# Display the mean face image
plt.imshow(mean_face_image, cmap='gray')
plt.title('Mean Face')
plt.axis('off') # Hide axis
plt.show()
```

```
Subtask 2
```



```
print("\nSubtask 3\n")
# Compute the mean face
mean_face = np.mean(P, axis=1)
# Convert P to double (float64 in numpy)
P = P.astype(np.float64)
# Subtract the mean face from each column of P
mean_face_column = mean_face.reshape(-1, 1)
P = P - mean_face_column @ np.ones((1, Database_Size))
# Print the first column of P to verify subtraction
print(P[:, 0])
```

```
Subtask 3

[-38.8        -38.3        -42.73333333 ... -14.56666667 -14.4
 -16.33333333]
```

```
print("\nSubtask 4\n")
# Compute the covariance matrix P^T * P
PTP = P.T @ P
# Compute the eigenvalues and eigenvectors of P^T * P
Values, Vectors = np.linalg.eig(PTP)
# Compute the actual eigenvectors of the covariance matrix
EigenVectors = P @ Vectors
# Normalize the eigenvectors
EigenVectors = EigenVectors / np.linalg.norm(EigenVectors, axis=0)
# Display the first few eigenvalues for verification
print("Eigenvalues:", Values)
```

```
Subtask 4

Eigenvalues: [9.69069623e+07 4.87292722e+07 4.37900456e+07 3.34756826e+07
 2.46521889e+07 2.04782347e+07 1.63378623e+07 1.60131047e+07
 1.41415577e+07 1.32446059e+07 5.46447125e-09 1.11004779e+07
 1.06792480e+07 9.84547441e+06 9.27154963e+06 8.44340124e+06
 7.68223964e+06 3.30696241e+06 3.42829390e+06 3.52946256e+06
 3.76629399e+06 4.17775602e+06 7.12503632e+06 6.92200481e+06
 4.74323775e+06 4.98622750e+06 5.37363278e+06 5.60944036e+06
 6.08634551e+06 6.37529704e+06]
```

```
print("\nSubtask 5\n")
# Display the set of eigenfaces
eigenfaces = []
for j in range(1, Database_Size):
    eigenface = EigenVectors[:, j] + mean_face
    eigenface_image = eigenface.reshape(m, n)
    eigenfaces.append(eigenface_image)
# Concatenate the eigenfaces horizontally
EigenFaces = np.hstack(eigenfaces)
# Display the eigenfaces
plt.figure(figsize=(15, 5))
plt.imshow(EigenFaces, cmap='gray')
plt.title('Eigenfaces')
plt.axis('off') # Hide axis
plt.show()
```

```
Subtask 5
```


```
print("\nSubtask 6\n")
# Compute the Products matrix
Products = EigenVectors.T @ EigenVectors
# Print the Products matrix to verify orthogonality
print("Products matrix:")
print(Products)
# Check if Products matrix is diagonal
is_diagonal = np.allclose(Products, np.diag(np.diagonal(Products)))
print(f"Is Products matrix diagonal? {is_diagonal}")
```

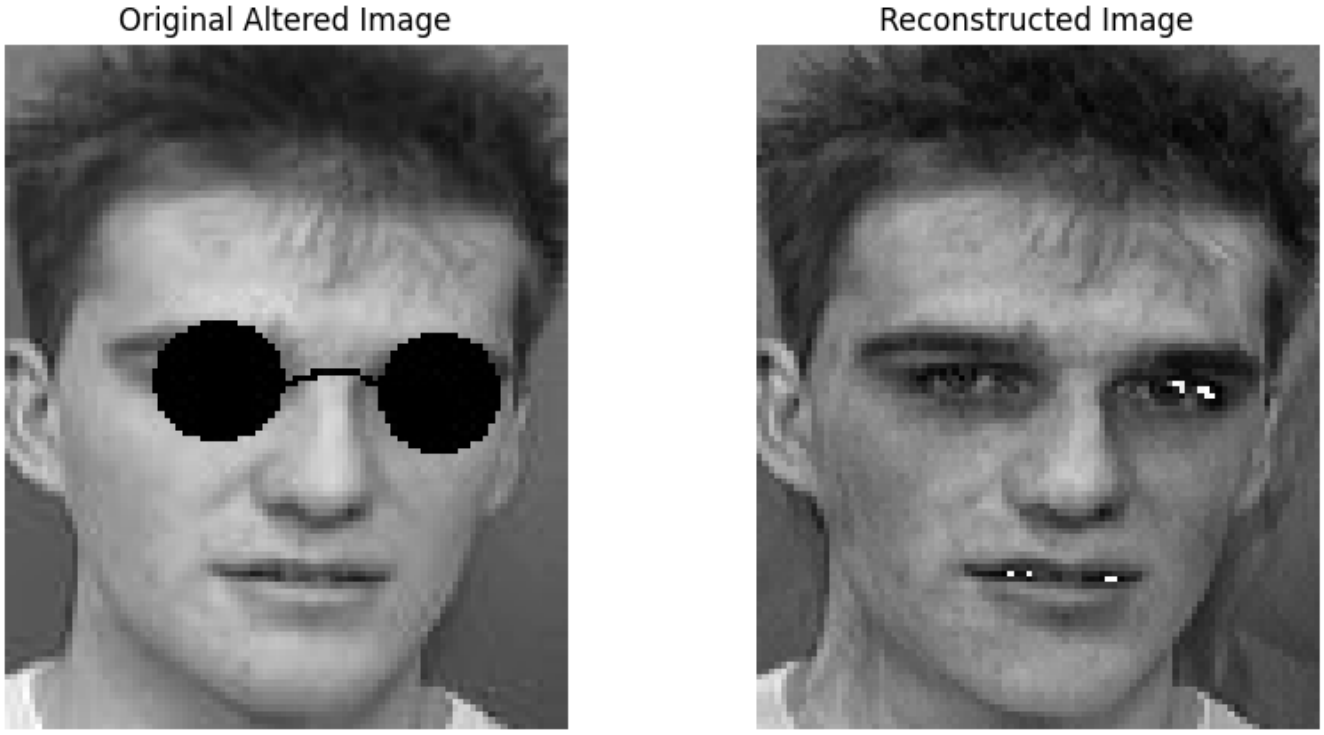
```
 -4.70896109e-16 -1.19934444e-15 -8.48279780e-16  1.09114107e-15
  2.12850571e-15  4.76615275e-16  2.63217182e-16 -9.22872889e-16
 -4.48751279e-16 -1.22341373e-15  1.00000000e+00 -5.24753851e-15
  4.96998276e-16 -1.71553310e-15  9.58651561e-16 -8.82106888e-16
 -9.94647073e-16  1.56645530e-15]
[ 4.02455846e-16  4.32596667e-16 -4.62303806e-16  8.58688121e-17
  2.16840434e-16  8.86877377e-17 -6.28620420e-16  3.10515502e-16
  5.59448321e-17  2.09901541e-16 -2.63691413e-03 -1.99140834e-16
 -1.38777878e-17  2.66280054e-16  8.03176969e-16  3.26128013e-16
  9.26776017e-16 -1.64278313e-15  5.90673344e-16  5.95443833e-16
 -1.65638987e-16 -1.77809156e-16 -5.24753851e-15  1.00000000e+00
  7.75204553e-17 -1.04755614e-15  1.07726328e-15 -4.74013190e-16
 -6.24500451e-16  1.85051627e-15]
[ 6.31439345e-16  9.02056208e-17  1.76941795e-16 -2.06432094e-16
```

```
 -1.42247325e-16  5.06539255e-16  7.11236625e-17  1.96620064e-16
 -5.03069808e-17  1.56125113e-16 -1.92157242e-02  5.55111512e-17
 -1.62196645e-16  9.71445147e-17  2.48932819e-16 -5.86336535e-16
  6.78276879e-16  0.00000000e+00 -1.56298585e-15 -9.71445147e-17
  1.29930788e-15 -8.52399748e-16  4.96998276e-16  7.75204553e-17
  1.00000000e+00 -1.86829718e-15 -5.86336535e-16  1.10068205e-15
 -8.31799907e-16  9.59302082e-16]
[-1.06577074e-16  2.08383658e-16 -3.17454396e-16  4.11129464e-16
  1.37043155e-16  7.80625564e-18  2.37657116e-16 -3.66894015e-16
 -3.48679419e-16 -9.51929507e-17  1.46197596e-02 -7.10586104e-16
 -4.83120488e-16 -2.01227923e-16  2.08166817e-16  1.28304485e-15
  3.29814301e-16  1.29063427e-15 -2.01271291e-15 -1.55691432e-16
  1.66533454e-16  1.81365339e-15 -1.71553310e-15 -1.04755614e-15
 -1.86829718e-15  1.00000000e+00  2.70790335e-15 -1.23945992e-15
 -7.07147150e-16  1.46909394e-16]
[ 1.71737624e-16  3.72857127e-16  2.25514052e-16  1.42247325e-16
 -2.25514052e-17 -3.55618313e-16 -3.46944695e-17 -3.20490162e-16
 -1.03974988e-16  1.99493200e-17 -5.70794590e-02  3.02709247e-16
 -4.38668199e-16  4.82686807e-16 -7.19910243e-16  5.37764278e-17
  2.68014777e-16  1.03042574e-15 -7.41594286e-17 -2.42861287e-16
  4.81385765e-16 -1.73819292e-15  9.58651561e-16  1.07726328e-15
 -5.86336535e-16  2.70790335e-15  1.00000000e+00  5.83560977e-15
 -1.37650308e-15  8.23126289e-16]
[-8.93382590e-17 -4.87890978e-18 -1.97758476e-16  7.19910243e-17
  2.48065457e-16 -2.67581096e-16  4.06792655e-16  2.44596010e-16
  3.14039159e-16  1.40512602e-16 -1.26993703e-01 -6.50521303e-17
  7.68482500e-16 -7.97972799e-17  3.48245738e-16 -5.60749364e-16
  1.55452907e-15 -8.77770079e-16  2.72785267e-16  5.20417043e-16
 -7.64579372e-16  2.92300906e-16 -8.82106888e-16 -4.74013190e-16
  1.10068205e-15 -1.23945992e-15  5.83560977e-15  1.00000000e+00
 -2.38351006e-15  2.44335802e-15]
[-4.90276222e-16  3.72965547e-16 -1.12973866e-16 -1.46584134e-16
 -1.70761842e-16 -1.42247325e-16 -3.48028897e-16  1.55257751e-16
  1.14708590e-16 -3.07046055e-16  1.41150240e-01  1.51788304e-16
 -2.77230496e-16 -8.37871439e-16 -3.85108612e-16  2.07949977e-16
  9.25474974e-16  6.70470623e-16  6.56159155e-16 -9.29811783e-16
 -4.63171168e-16  1.17267307e-15 -9.94647073e-16 -6.24500451e-16
 -8.31799907e-16 -7.07147150e-16 -1.37650308e-15 -2.38351006e-15
  1.00000000e+00  1.64798730e-16]
[ 2.60859043e-16 -6.46184495e-17 -1.29887420e-16 -4.46691295e-17
 -8.95034304e-17 -3.13600396e-16  1.13299127e-16 -1.79977561e-16
  4.60785923e-16  5.14345511e-16  8.70618276e-02 -2.07299455e-16
 -3.94649591e-17 -2.48065457e-16  1.01806584e-16  1.48839274e-15
  2.82759927e-16  4.73579509e-16 -1.98625838e-16  5.39499001e-16
 -5.23886490e-16 -5.53376789e-16  1.56645530e-15  1.85051627e-15
```

```
print("\nSubtask 7\n")
# Define image dimensions
m, n = 112, 92
# Read the altered image
altered_image_path ="/content/database/person30altered1.pgm"
image_read = Image.open(altered_image_path)
image_array = np.array(image_read)
```

```
U = image_array.reshape(m * n, 1)
# Compute the norms of the eigenvectors
Products = EigenVectors.T @ EigenVectors
NormsEigenVectors = np.diag(Products)
# Compute the projection coefficients
W = EigenVectors.T @ (U.astype(np.float64) - mean_face.reshape(-1, 1))
W = W / NormsEigenVectors.reshape(-1, 1) # Ensure proper division
# Reconstruct the image from the projection
U_approx = EigenVectors @ W + mean_face.reshape(-1, 1)
# Print shapes for debugging
print("Shape of U_approx:", U_approx.shape)
print("Expected shape:", (m * n, 1))
# Ensure the shape matches for reshaping
if U_approx.shape[0] == m * n and U_approx.shape[1] == 1:
    image_approx = U_approx.reshape(m, n).astype(np.uint8)
else:
    raise ValueError(f"Cannot reshape array of size {U_approx.size} into shape
# Display the original altered image and the reconstructed image
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image_array, cmap='gray')
plt.title('Original Altered Image')
plt.axis('off') # Hide axis
plt.subplot(1, 2, 2)
plt.imshow(image_approx, cmap='gray')
plt.title('Reconstructed Image')
plt.axis('off') # Hide axis
plt.show()
```

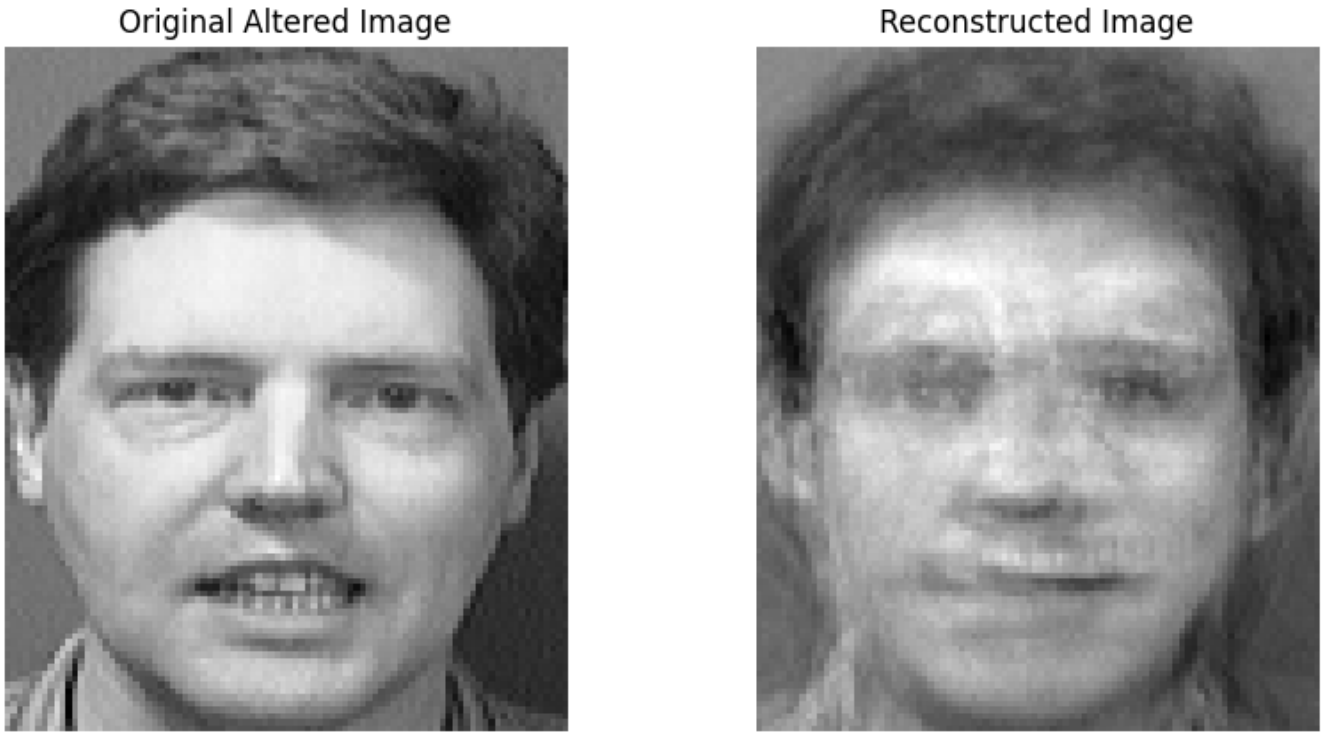
```
Subtask 7

Shape of U_approx: (10304, 1)
Expected shape: (10304, 1)
```




```
print("\nSubtask 8\n")
# Define image dimensions
m, n = 112, 92
# Read the altered image
altered_image_path ="/content/database/person31.pgm"
image_read = Image.open(altered_image_path)
image_array = np.array(image_read)
U = image_array.reshape(m * n, 1)
# Compute the norms of the eigenvectors
Products = EigenVectors.T @ EigenVectors
NormsEigenVectors = np.diag(Products)
# Compute the projection coefficients
W = EigenVectors.T @ (U.astype(np.float64) - mean_face.reshape(-1, 1))
W = W / NormsEigenVectors.reshape(-1, 1) # Ensure proper division
# Reconstruct the image from the projection
U_approx = EigenVectors @ W + mean_face.reshape(-1, 1)
# Print shapes for debugging
print("Shape of U_approx:", U_approx.shape)
print("Expected shape:", (m * n, 1))
```

```
# Ensure the shape matches for reshaping
if U_approx.shape[0] == m * n and U_approx.shape[1] == 1:
    image_approx = U_approx.reshape(m, n).astype(np.uint8)
else:
    raise ValueError(f"Cannot reshape array of size {U_approx.size} into shape
# Display the original altered image and the reconstructed image
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image_array, cmap='gray')
plt.title('Original Altered Image')
plt.axis('off') # Hide axis
plt.subplot(1, 2, 2)
plt.imshow(image_approx, cmap='gray')
plt.title('Reconstructed Image')
plt.axis('off') # Hide axis
plt.show()
```

```
Subtask 8

Shape of U_approx: (10304, 1)
Expected shape: (10304, 1)
```




```
print("\nSubtask 9\n")
# # Recognition and approximation of a new face (person31.pgm)
import numpy as np
```

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
# Define image dimensions
m, n = 112, 92
# Read the new image (person31.pgm)
new_image_path ="/content/database/person31.pgm"
image_read = Image.open(new_image_path)
image_array = np.array(image_read)
U = image_array.reshape(m * n, 1)
# Compute the norms of the eigenvectors
Products = EigenVectors.T @ EigenVectors
NormsEigenVectors = np.diag(Products)
# Compute the projection coefficients
W = EigenVectors.T @ (U.astype(np.float64) - mean_face.reshape(-1, 1))
W = W / NormsEigenVectors.reshape(-1, 1) # Ensure proper division
# Reconstruct the image from the projection
U_approx = EigenVectors @ W + mean_face.reshape(-1, 1)
114
# Print shapes for debugging
print("Shape of U_approx:", U_approx.shape)
print("Expected shape:", (m * n, 1))
# Ensure the shape matches for reshaping
if U_approx.shape[0] == m * n and U_approx.shape[1] == 1:
    image_approx = U_approx.reshape(m, n).astype(np.uint8)
else:
    raise ValueError(f"Cannot reshape array of size {U_approx.size} into shape (
# Display the original new image and the reconstructed image
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image_array, cmap='gray')
plt.title('Original New Image')
plt.axis('off') # Hide axis
plt.subplot(1, 2, 2)
plt.imshow(image_approx, cmap='gray')
plt.title('Reconstructed Image')
plt.axis('off') # Hide axis
plt.show()
# Variables: image_read, U, NormsEigenVectors, W, U_approx
```

```
Subtask 9

Shape of U_approx: (10304, 1)
Expected shape: (10304, 1)
```