sarthakshrestha-worksheet1

March 1, 2025

1 Name = Sarthak Shrestha

2 Group 20

3 # Introduction to Python Imaging Library(PIL)

3.1 2.1 Exercise - 1:

4 Complete all the Task.

5 1. Read and display the image.

6 Read the image using the Pillow library and display it.

[ ]:
```
from PIL import Image
# display image in colab
image_colored = Image.open("Lenna_(test_image).png")
display(image_colored)
```

6.1 You can also use matplotlib to display the image.

```
[ ]: from PIL import Image
     import matplotlib.pyplot as plt
     import numpy as np

     # Open the image with PIL
     image_colored = Image.open("Lenna_(test_image).png")

     # Convert PIL image to numpy array
     image_array = np.array(image_colored)

     # Display the image using matplotlib
```

```
plt.imshow(image_array)
plt.axis('off')  # Turn off axis numbers and ticks
plt.show()
```

6.2 2. Display only the top left corner of 100x100 pixels.

6.3 • Extract the top-left corner of the image (100x100 pixels) and display it using NumPy and Array Indexing.

[2]:
```
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np  # Import numpy

# Open the image with PIL
image_colored = Image.open("Lenna_(test_image).png")

# Convert PIL image to numpy array
image_array = np.array(image_colored)

# Extract the top-left 100x100 pixels
top_left_corner = image_array[:100, :100]  # Selecting first 100 rows and␣
↪columns
```

```python
# Display the extracted portion
plt.imshow(top_left_corner)
plt.axis('off')  # Hide axis
plt.show()
```

6.4 3. Show the three color channels (R, G, B).

6.5 • Separate the image into its three color channels (Red, Green, and Blue) and display them individually, labeling each channel as R, G, and B.{Using NumPy.}

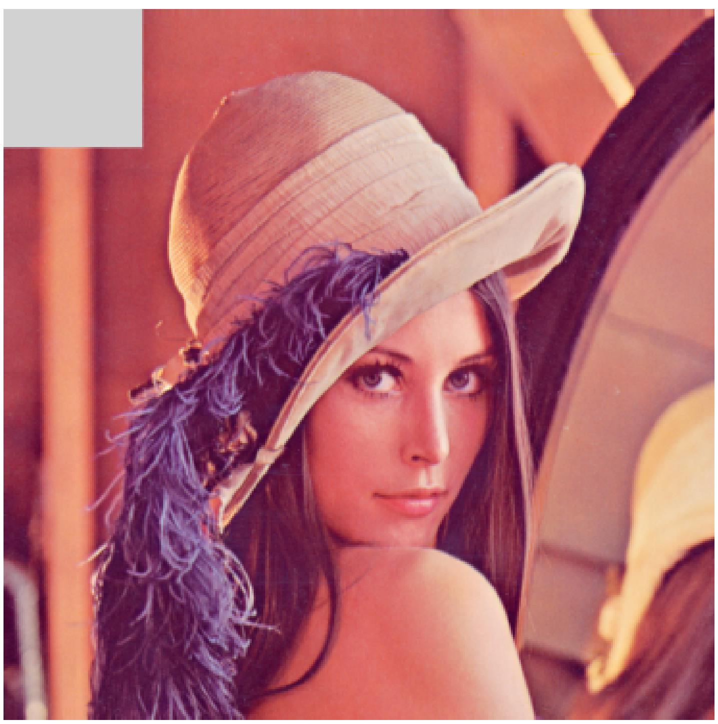
[3]:
```python
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np  # Import numpy

# Open the image with PIL
image_colored = Image.open("Lenna_(test_image).png")

# Convert PIL image to numpy array
image_array = np.array(image_colored)

# Extract Red, Green, and Blue channels
red_channel = image_array.copy()
```

```python
green_channel = image_array.copy()
blue_channel = image_array.copy()

# Keep only the respective channel by setting other channels to 0
red_channel[:, :, 1:] = 0  # Set Green and Blue to 0, keeping only Red
green_channel[:, :, [0, 2]] = 0  # Set Red and Blue to 0, keeping only Green
blue_channel[:, :, :2] = 0  # Set Red and Green to 0, keeping only Blue

# Display the three channels
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

axes[0].imshow(red_channel)
axes[0].set_title("Red Channel")
axes[0].axis('off')

axes[1].imshow(green_channel)
axes[1].set_title("Green Channel")
axes[1].axis('off')

axes[2].imshow(blue_channel)
axes[2].set_title("Blue Channel")
axes[2].axis('off')

plt.show()
```

6.6 4. Modify the top 100 × 100 pixels to a value of 210 and display the resulting image:

6.7 • Modify the pixel values of the top-left 100 × 100 region to have a value of 210 (which is a light gray color), and then display the modified image.

[4]:
```python
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np  # Import numpy

# Open the image with PIL
image_colored = Image.open("Lenna_(test_image).png")

# Convert PIL image to numpy array
image_array = np.array(image_colored)

# Modify the top-left 100x100 region to 210 (light gray)
image_array[:100, :100] = 210

# Convert back to image format and display
plt.imshow(image_array)
plt.axis('off')  # Hide axis
plt.show()
```



6.8 Exercise - 2:

6.9 Load and display a grayscale image.

6.10 Load a grayscale image using the Pillow library.

6.11 Display the grayscale image using matplotlib.

[8]:
```python
from PIL import Image
import matplotlib.pyplot as plt

# Load the image in grayscale mode
image_gray = Image.open("cameraman.png").convert("L")

# Display using matplotlib
plt.imshow(image_gray, cmap="gray")  # Ensure grayscale colormap
plt.axis("off")  # Hide axes
plt.title("Grayscale Image - Cameraman")
plt.show()
```

6.12 Extract and display the middle section of the image (150 pixels).

6.13 • Extract a 150 pixel section from the center of the image using NumPy array slicing.

6.14 • Display this cropped image using matplotlib.

[10]:

```
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np  # Import numpy

# Open the image in grayscale mode
image_gray = Image.open("cameraman.png").convert("L")

# Convert PIL image to numpy array
image_array = np.array(image_gray)

# Get image dimensions
height, width = image_array.shape

# Calculate the middle region (150 pixels in height)
start_y = (height - 150) // 2  # Starting Y-coordinate
end_y = start_y + 150  # Ending Y-coordinate

# Extract the middle section (150 pixels)
middle_section = image_array[start_y:end_y, :]

# Display the extracted section using matplotlib
plt.imshow(middle_section, cmap="gray")
plt.axis("off")  # Hide axis
plt.title("Middle Section (150 pixels)")
plt.show()
```

Middle Section (150 pixels)

6.15 Apply a simple threshold to the image (e.g., set all pixel values below 100 to 0).

6.16 Apply a threshold to the grayscale image: set all pixel values below 100 to 0, and all values above 100 to 255 (creating a binary image).

6.17 • Display the resulting binary image.

[11]:
```
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np  # Import numpy

# Open the image in grayscale mode
image_gray = Image.open("cameraman.png").convert("L")

# Convert PIL image to numpy array
image_array = np.array(image_gray)

# Apply thresholding: Set values < 100 to 0, and >= 100 to 255
threshold_value = 100
binary_image = np.where(image_array < threshold_value, 0, 255).astype(np.uint8)

# Display the binary image using matplotlib
plt.imshow(binary_image, cmap="gray")
plt.axis("off")  # Hide axis
plt.title("Binary Image (Threshold = 100)")
plt.show()
```

Binary Image (Threshold = 100)

6.18 Rotate the image 90 degrees clockwise and display the result.

6.19 Rotate the image by 90 degrees clockwise using the Pillow rotate method or by manipulating the image array.

6.20 Display the rotated image using matplotlib.

[12]:
```
from PIL import Image
import matplotlib.pyplot as plt

# Open the image in grayscale mode
image_gray = Image.open("cameraman.png").convert("L")

# Rotate 90 degrees clockwise using Pillow (-90 degrees counterclockwise)
rotated_image = image_gray.rotate(-90, expand=True)

# Display the rotated image
plt.imshow(rotated_image, cmap="gray")
plt.axis("off")  # Hide axis
plt.title("Rotated 90° Clockwise (Pillow)")
plt.show()
```

Rotated 90° Clockwise (Pillow)

6.21 **Convert the grayscale image to an RGB image.**

6.22 **Convert the grayscale image into an RGB image where the grayscale values are replicated across all three channels (R, G, and B).**

6.23 **Display the converted RGB image using matplotlib.**

[13]:
```python
from PIL import Image
import matplotlib.pyplot as plt

# Open the image in grayscale mode
image_gray = Image.open("cameraman.png").convert("L")

# Convert grayscale to RGB using Pillow
image_rgb = image_gray.convert("RGB")

# Display the converted RGB image
plt.imshow(image_rgb)
plt.axis("off")  # Hide axis
plt.title("Converted RGB Image (Pillow)")
plt.show()
```

Converted RGB Image (Pillow)

6.23.1 Image Compression and Decompression using PCA.

6.24 In this exercise, build a PCA from scratch using explained variance method for image compression task.

6.25 Load and Prepare Data:

7 • Fetch an image of you choice.{If colour convert to grayscale}

8 • Center the dataset - Standaridze the Data.

9 • Calculate the covaraince matrix of the Standaridze data.

[21]:
```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Load the image and convert it to grayscale
image_gray = Image.open("floyd.jpg").convert("L")  # Convert to grayscale

# Convert grayscale image to NumPy array
image_array = np.array(image_gray)
```

```python
# Step 2: Standardize the data (Center the dataset)
mean_pixel = np.mean(image_array, axis=0)  # Compute mean along each column␣
 ↪(pixel)
std_pixel = np.std(image_array, axis=0)    # Compute std along each column␣
 ↪(pixel)

standardized_image = (image_array - mean_pixel) / std_pixel  # Standardization

# Step 3: Reshape the standardized image for PCA (flatten the 2D image into 1D␣
 ↪vectors for each pixel row)
reshaped_image = standardized_image.reshape(-1, image_array.shape[1])  #␣
 ↪Flatten rows into columns

# Step 4: Compute the covariance matrix (rows are samples, columns are features)
cov_matrix = np.cov(reshaped_image, rowvar=False)

# Display grayscale image
plt.imshow(image_gray, cmap="gray")
plt.axis("off")
plt.title("Grayscale Image - Floyd")
plt.show()

# Print covariance matrix and top eigenvalues
print("Covariance Matrix:\n", cov_matrix)
```

Grayscale Image - Floyd

```
Covariance Matrix:
 [[1.0046729  0.99668958 0.98440611 … 0.84903837 0.84864559 0.82343495]
 [0.99668958 1.0046729  0.99859917 … 0.86167492 0.86057936 0.83075345]
 [0.98440611 0.99859917 1.0046729  … 0.86298486 0.86157475 0.83225358]
 …
 [0.84903837 0.86167492 0.86298486 … 1.0046729  0.9683979  0.91085806]
 [0.84864559 0.86057936 0.86157475 … 0.9683979  1.0046729  0.97034525]
 [0.82343495 0.83075345 0.83225358 … 0.91085806 0.97034525 1.0046729 ]]
```

9.0.1 Eigen Decomposition and Identifying Principal Components:

9.1 • Compute Eigen Values and Eigen Vectors.

9.2 • Sort the eigenvalues in descending order and choose the top k eigenvectors corresponding to the highest eigenvalues.

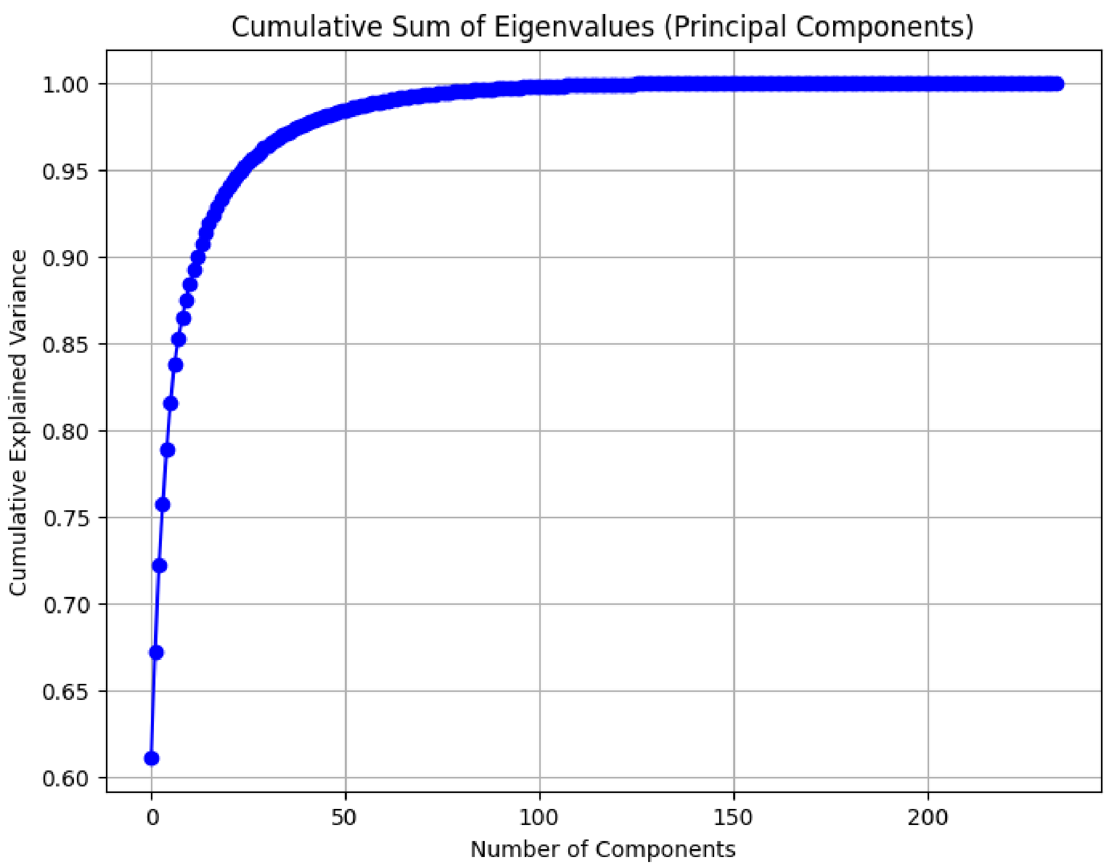
9.3 • Identify the Principal Components with the help of cumulative Sum plot.

[17]:
```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Load the image and convert it to grayscale
image_gray = Image.open("floyd.jpg").convert("L")  # Convert to grayscale
```

```python
# Convert grayscale image to NumPy array
image_array = np.array(image_gray)

# Step 2: Standardize the data (Center the dataset)
mean_pixel = np.mean(image_array, axis=0)  # Compute mean along each column (pixel)
std_pixel = np.std(image_array, axis=0)    # Compute std along each column (pixel)

standardized_image = (image_array - mean_pixel) / std_pixel  # Standardization

# Step 3: Reshape the standardized image for PCA (flatten the 2D image into 1D vectors for each pixel row)
reshaped_image = standardized_image.reshape(-1, image_array.shape[1])  # Flatten rows into columns

# Step 4: Compute the covariance matrix (rows are samples, columns are features)
cov_matrix = np.cov(reshaped_image, rowvar=False)

# Step 5: Compute eigenvalues and eigenvectors for PCA
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

# Step 6: Sort eigenvalues and eigenvectors in descending order of eigenvalue size
sorted_indices = np.argsort(eigenvalues)[::-1]
sorted_eigenvalues = eigenvalues[sorted_indices]
sorted_eigenvectors = eigenvectors[:, sorted_indices]

# Step 7: Calculate the cumulative sum of eigenvalues
cumulative_sum = np.cumsum(sorted_eigenvalues) / np.sum(sorted_eigenvalues)

# Step 8: Plot the cumulative sum of eigenvalues
plt.figure(figsize=(8, 6))
plt.plot(cumulative_sum, marker='o', linestyle='-', color='b')
plt.title("Cumulative Sum of Eigenvalues (Principal Components)")
plt.xlabel("Number of Components")
plt.ylabel("Cumulative Explained Variance")
plt.grid(True)
plt.show()

# Step 9: Print sorted eigenvalues and top components
print("Sorted Eigenvalues:\n", sorted_eigenvalues)
print("\nTop 5 Eigenvectors:\n", sorted_eigenvectors[:, :5])

# Optionally, choose the top k components where the cumulative variance is close to 1
```

```python
k = np.argmax(cumulative_sum >= 0.95)  # For example, choose components␣
↪explaining 95% of variance
print(f"Number of Components for 95% Variance: {k+1}")
```

```
/usr/local/lib/python3.11/dist-packages/matplotlib/cbook.py:1709:
ComplexWarning: Casting complex values to real discards the imaginary part
  return math.isfinite(val)
/usr/local/lib/python3.11/dist-packages/matplotlib/cbook.py:1345:
ComplexWarning: Casting complex values to real discards the imaginary part
  return np.asarray(x, float)
```




```
Sorted Eigenvalues:
 [ 1.43678487e+02+0.00000000e+00j  1.42687915e+01+0.00000000e+00j
  1.16881431e+01+0.00000000e+00j  8.31147079e+00+0.00000000e+00j
  7.52893745e+00+0.00000000e+00j  6.32197619e+00+0.00000000e+00j
  5.13056376e+00+0.00000000e+00j  3.49249403e+00+0.00000000e+00j
  2.80305873e+00+0.00000000e+00j  2.47003262e+00+0.00000000e+00j
  2.14210202e+00+0.00000000e+00j  1.93874605e+00+0.00000000e+00j
  1.78659786e+00+0.00000000e+00j  1.69551112e+00+0.00000000e+00j
  1.54955359e+00+0.00000000e+00j  1.25387187e+00+0.00000000e+00j
```

1.16150125e+00+0.00000000e+00j	1.05596611e+00+0.00000000e+00j
9.85206919e-01+0.00000000e+00j	9.24852755e-01+0.00000000e+00j
8.43189689e-01+0.00000000e+00j	7.52748936e-01+0.00000000e+00j
6.86335684e-01+0.00000000e+00j	6.29033618e-01+0.00000000e+00j
5.87596293e-01+0.00000000e+00j	5.77497520e-01+0.00000000e+00j
5.24319639e-01+0.00000000e+00j	4.98482219e-01+0.00000000e+00j
4.92323140e-01+0.00000000e+00j	4.50442956e-01+0.00000000e+00j
3.89938332e-01+0.00000000e+00j	3.72413521e-01+0.00000000e+00j
3.65391929e-01+0.00000000e+00j	3.29592378e-01+0.00000000e+00j
3.09313417e-01+0.00000000e+00j	2.95692940e-01+0.00000000e+00j
2.81596133e-01+0.00000000e+00j	2.69379351e-01+0.00000000e+00j
2.56668618e-01+0.00000000e+00j	2.48629509e-01+0.00000000e+00j
2.33263911e-01+0.00000000e+00j	2.19472259e-01+0.00000000e+00j
2.06252434e-01+0.00000000e+00j	1.99662922e-01+0.00000000e+00j
1.90255828e-01+0.00000000e+00j	1.81844765e-01+0.00000000e+00j
1.76055283e-01+0.00000000e+00j	1.70912623e-01+0.00000000e+00j
1.62562481e-01+0.00000000e+00j	1.59128934e-01+0.00000000e+00j
1.54465702e-01+0.00000000e+00j	1.48445197e-01+0.00000000e+00j
1.39829333e-01+0.00000000e+00j	1.35255680e-01+0.00000000e+00j
1.29531487e-01+0.00000000e+00j	1.27204610e-01+0.00000000e+00j
1.19763846e-01+0.00000000e+00j	1.12351722e-01+0.00000000e+00j
1.05303304e-01+0.00000000e+00j	1.01379611e-01+0.00000000e+00j
9.68711216e-02+0.00000000e+00j	9.52418629e-02+0.00000000e+00j
9.22116198e-02+0.00000000e+00j	8.60238557e-02+0.00000000e+00j
8.18519359e-02+0.00000000e+00j	7.98432462e-02+0.00000000e+00j
7.75913567e-02+0.00000000e+00j	7.43059070e-02+0.00000000e+00j
7.31327434e-02+0.00000000e+00j	7.15766999e-02+0.00000000e+00j
6.73410060e-02+0.00000000e+00j	6.42945077e-02+0.00000000e+00j
6.38980449e-02+0.00000000e+00j	6.08191152e-02+0.00000000e+00j
5.75436874e-02+0.00000000e+00j	5.39404648e-02+0.00000000e+00j
5.27807220e-02+0.00000000e+00j	5.19386560e-02+0.00000000e+00j
4.98919766e-02+0.00000000e+00j	4.67304443e-02+0.00000000e+00j
4.54671711e-02+0.00000000e+00j	4.29456253e-02+0.00000000e+00j
4.18263305e-02+0.00000000e+00j	3.83798851e-02+0.00000000e+00j
3.72779170e-02+0.00000000e+00j	3.59760322e-02+0.00000000e+00j
3.55451534e-02+0.00000000e+00j	3.48605078e-02+0.00000000e+00j
3.38795308e-02+0.00000000e+00j	3.19142240e-02+0.00000000e+00j
2.94937539e-02+0.00000000e+00j	2.92039107e-02+0.00000000e+00j
2.81537704e-02+0.00000000e+00j	2.74044036e-02+0.00000000e+00j
2.69556781e-02+0.00000000e+00j	2.57511417e-02+0.00000000e+00j
2.53024193e-02+0.00000000e+00j	2.39354011e-02+0.00000000e+00j
2.34622859e-02+0.00000000e+00j	2.31092771e-02+0.00000000e+00j
2.17146709e-02+0.00000000e+00j	2.09151778e-02+0.00000000e+00j
2.04842198e-02+0.00000000e+00j	1.93784131e-02+0.00000000e+00j
1.86056134e-02+0.00000000e+00j	1.82196418e-02+0.00000000e+00j
1.74290939e-02+0.00000000e+00j	1.59857346e-02+0.00000000e+00j
1.55433898e-02+0.00000000e+00j	1.51493370e-02+0.00000000e+00j
1.47908350e-02+0.00000000e+00j	1.45475240e-02+0.00000000e+00j

```
1.33866688e-02+0.00000000e+00j  1.31606438e-02+0.00000000e+00j
1.28765116e-02+0.00000000e+00j  1.27547874e-02+0.00000000e+00j
1.21291892e-02+0.00000000e+00j  1.14082506e-02+0.00000000e+00j
1.08734462e-02+0.00000000e+00j  1.07362105e-02+0.00000000e+00j
9.90259491e-03+0.00000000e+00j  9.75295997e-03+0.00000000e+00j
9.42898570e-03+0.00000000e+00j  8.85485933e-03+0.00000000e+00j
8.70959695e-03+0.00000000e+00j  8.17457014e-03+0.00000000e+00j
7.84805980e-03+0.00000000e+00j  7.79302900e-03+0.00000000e+00j
7.16213127e-03+0.00000000e+00j  6.93637575e-03+0.00000000e+00j
6.76155991e-03+0.00000000e+00j  6.29388944e-03+0.00000000e+00j
6.15652839e-03+0.00000000e+00j  6.04412275e-03+0.00000000e+00j
5.40921243e-03+0.00000000e+00j  5.30265549e-03+0.00000000e+00j
4.91984222e-03+0.00000000e+00j  4.85829745e-03+0.00000000e+00j
4.70445257e-03+0.00000000e+00j  4.47050993e-03+0.00000000e+00j
4.28756057e-03+0.00000000e+00j  4.15753785e-03+0.00000000e+00j
4.06321203e-03+0.00000000e+00j  3.65773273e-03+0.00000000e+00j
3.47090778e-03+0.00000000e+00j  3.42411694e-03+0.00000000e+00j
3.09913645e-03+0.00000000e+00j  3.00019514e-03+0.00000000e+00j
2.94544553e-03+0.00000000e+00j  2.78078288e-03+0.00000000e+00j
2.63227192e-03+0.00000000e+00j  2.55191299e-03+0.00000000e+00j
2.47612170e-03+0.00000000e+00j  2.33644530e-03+0.00000000e+00j
2.21690130e-03+0.00000000e+00j  2.10866820e-03+0.00000000e+00j
2.04483717e-03+0.00000000e+00j  1.99055457e-03+0.00000000e+00j
1.85506161e-03+0.00000000e+00j  1.76477833e-03+0.00000000e+00j
1.68413206e-03+0.00000000e+00j  1.65458525e-03+0.00000000e+00j
1.52942853e-03+0.00000000e+00j  1.45605071e-03+0.00000000e+00j
1.39395147e-03+0.00000000e+00j  1.32654817e-03+0.00000000e+00j
1.23190952e-03+0.00000000e+00j  1.15360943e-03+0.00000000e+00j
1.12338723e-03+0.00000000e+00j  1.06534012e-03+0.00000000e+00j
1.04222565e-03+0.00000000e+00j  9.64652419e-04+0.00000000e+00j
9.23656863e-04+0.00000000e+00j  8.26122105e-04+0.00000000e+00j
7.91175861e-04+0.00000000e+00j  7.38483330e-04+0.00000000e+00j
7.27911899e-04+0.00000000e+00j  6.34315109e-04+0.00000000e+00j
5.97313140e-04+0.00000000e+00j  5.31476544e-04+0.00000000e+00j
5.20447228e-04+0.00000000e+00j  4.92596453e-04+0.00000000e+00j
4.84375309e-04+0.00000000e+00j  4.38230848e-04+0.00000000e+00j
3.98090354e-04+0.00000000e+00j  3.83247949e-04+0.00000000e+00j
3.08381063e-04+0.00000000e+00j  3.02390224e-04+0.00000000e+00j
2.72706967e-04+0.00000000e+00j  2.47360087e-04+0.00000000e+00j
2.23167057e-04+0.00000000e+00j  2.11355979e-04+0.00000000e+00j
1.86490662e-04+0.00000000e+00j  1.69358306e-04+0.00000000e+00j
1.55962617e-04+0.00000000e+00j  1.26037895e-04+0.00000000e+00j
1.10532338e-04+0.00000000e+00j  1.03029986e-04+0.00000000e+00j
8.31576442e-05+0.00000000e+00j  7.49458662e-05+0.00000000e+00j
4.84487296e-05+0.00000000e+00j  4.40377683e-05+0.00000000e+00j
3.55187966e-05+0.00000000e+00j  3.20525450e-05+0.00000000e+00j
2.18837098e-05+0.00000000e+00j  1.91884940e-05+0.00000000e+00j
1.61636932e-05+0.00000000e+00j  1.48333949e-05+0.00000000e+00j
```

```
 1.05950787e-05+0.00000000e+00j  9.54932872e-06+0.00000000e+00j
 5.69828528e-06+0.00000000e+00j  3.81834727e-06+0.00000000e+00j
 2.46803211e-06+0.00000000e+00j  1.80296930e-06+0.00000000e+00j
 4.74220530e-15+4.56485622e-16j  4.74220530e-15-4.56485622e-16j
 2.93755943e-15+0.00000000e+00j  2.40930077e-15+0.00000000e+00j
 1.81961902e-15+3.17308397e-17j  1.81961902e-15-3.17308397e-17j
 1.05772796e-15+7.33190034e-16j  1.05772796e-15-7.33190034e-16j
 2.56825234e-16+0.00000000e+00j  1.38567687e-16+0.00000000e+00j
-4.04410019e-16+7.21925859e-16j -4.04410019e-16-7.21925859e-16j
-7.19070783e-16+5.35341541e-16j -7.19070783e-16-5.35341541e-16j
-1.55224760e-15+3.83819903e-16j -1.55224760e-15-3.83819903e-16j
-1.97715789e-15+0.00000000e+00j -2.88984063e-15+0.00000000e+00j
-4.55502561e-15+0.00000000e+00j -5.17719168e-15+0.00000000e+00j]

Top 5 Eigenvectors:
 [[ 0.07871314+0.j -0.03330886+0.j -0.02467622+0.j -0.01949597+0.j
   0.0460803 +0.j]
 [ 0.07870845+0.j -0.03190372+0.j -0.01578995+0.j -0.01882762+0.j
   0.04649685+0.j]
 [ 0.07908453+0.j -0.03372476+0.j -0.01697436+0.j -0.01537954+0.j
   0.03861279+0.j]
 …
 [ 0.07559936+0.j  0.01383237+0.j  0.04583133+0.j -0.02286798+0.j
  -0.03629745+0.j]
 [ 0.07546716+0.j  0.00470029+0.j  0.05598952+0.j -0.02758022+0.j
  -0.0411858 +0.j]
 [ 0.07295028+0.j -0.00867159+0.j  0.07191357+0.j -0.02132357+0.j
  -0.04859857+0.j]]
Number of Components for 95% Variance: 25
```

9.3.1 Reconstruction and Experiment:

9.4 • Reconstruction: Transform the original data by multiplying it with the selected eigenvectors(PCs) to obtain a lower-dimensional representation.

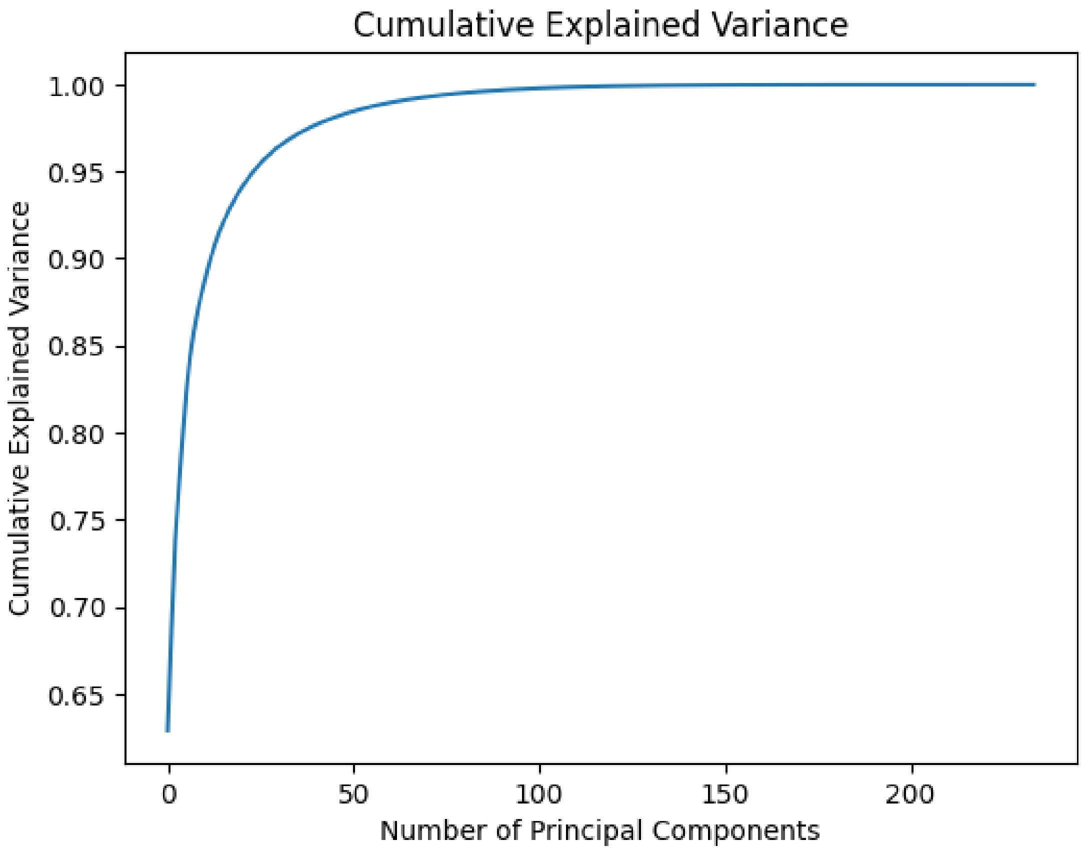
9.5 • Experiments: Pick Four different combination of principal components with various explained variance value and compare the result.

9.6 • Display the Results and Evaluate.

[20]:
```python
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Step 1: Load and Prepare Data
image = Image.open("floyd.jpg").convert("L")  # Convert image to grayscale
image_array = np.array(image)
```

```
# Flatten the image to 1D (each pixel becomes a feature)
flattened_image = image_array.flatten()

# Step 2: Standardize the Data
mean_pixel = np.mean(flattened_image)  # Mean of the flattened image
std_pixel = np.std(flattened_image)    # Standard deviation of the flattened␣
↪image

standardized_image = (flattened_image - mean_pixel) / std_pixel  # Standardize␣
↪the data

# Reshape the standardized image back to 2D (rows as samples, columns as␣
↪features)
standardized_image_2D = standardized_image.reshape(image_array.shape)

# Step 3: Calculate Covariance Matrix
cov_matrix = np.cov(standardized_image_2D, rowvar=False)

# Step 4: Eigen Decomposition
eig_vals, eig_vecs = np.linalg.eigh(cov_matrix)  # Eigenvalue decomposition

# Step 5: Sort Eigenvalues and Eigenvectors
sorted_indices = np.argsort(eig_vals)[::-1]  # Indices of eigenvalues in␣
↪descending order
eig_vals_sorted = eig_vals[sorted_indices]
eig_vecs_sorted = eig_vecs[:, sorted_indices]

# Step 6: Identify Principal Components
# Plot cumulative sum of eigenvalues to identify how many components to retain
cumulative_explained_variance = np.cumsum(eig_vals_sorted) / np.
↪sum(eig_vals_sorted)

# Plot the cumulative explained variance
plt.plot(cumulative_explained_variance)
plt.title("Cumulative Explained Variance")
plt.xlabel("Number of Principal Components")
plt.ylabel("Cumulative Explained Variance")
plt.show()

# Step 7: Reconstruction with Different Number of Principal Components
# Pick top k components (for example, k = 10, 50, 100, 200)
k_values = [10, 50, 100, 200]
reconstructed_images = []

for k in k_values:
    # Select the first k eigenvectors
    top_k_eigenvectors = eig_vecs_sorted[:, :k]
```

```
    # Project original data onto the k eigenvectors
    projected_data = np.dot(standardized_image_2D, top_k_eigenvectors)

    # Reconstruct the image from the projection
    reconstructed_image = np.dot(projected_data, top_k_eigenvectors.T)

    # Append reconstructed image
    reconstructed_images.append(reconstructed_image)

    # Display the result
    plt.imshow(reconstructed_image, cmap="gray")
    plt.title(f"Reconstructed Image with {k} Principal Components")
    plt.axis("off")
    plt.show()

# Step 8: Evaluation
# Compute and compare the reconstructed images' PSNR (Peak Signal-to-Noise␣
↪Ratio)
def psnr(original, reconstructed):
    mse = np.mean((original - reconstructed) ** 2)
    if mse == 0:
        return 100  # Perfect match
    max_pixel = 255.0
    return 20 * np.log10(max_pixel / np.sqrt(mse))

# Evaluate PSNR for each reconstructed image
original_image = image_array.astype(np.float32)
psnr_values = [psnr(original_image, recon) for recon in reconstructed_images]

# Print PSNR values for each k
for k, psnr_value in zip(k_values, psnr_values):
    print(f"PSNR for {k} Principal Components: {psnr_value:.2f} dB")
```


Cumulative Explained Variance
Cumulative Explained Variance
Number of Principal Components
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0
50
100
150
200

Reconstructed Image with 10 Principal Components

Reconstructed Image with 50 Principal Components

Reconstructed Image with 100 Principal Components

Reconstructed Image with 200 Principal Components

```
PSNR for 10 Principal Components: 5.94 dB
PSNR for 50 Principal Components: 5.94 dB
PSNR for 100 Principal Components: 5.94 dB
PSNR for 200 Principal Components: 5.94 dB
```