**Experiment No 8**

**Problem 1.1.** Program to compute the average of numbers in the list.

l=[5,7,15,17,12.5,9,16,18]

sum=0

for i in range(len(l)):

sum=sum+l[i]

print(sum/len(l))



**Alternate Solution**

l=[5,7,15,17,12.5,9,16,18]

sum=0

for i in l:

sum=sum+i

print(sum/len(l))



**Problem 1.2.** Program to read an image from the file and output it

import cv2

img = cv2.imread(‘filename’)

cv2.show(‘img’,img)

**Problem 1.3.** Program to create a two-dimensional array

import numpy as np

arr=np.array([[1,2,3],[4,5,6],[7,8,9]])

print(arr.ndim)

print(arr.shape)



**Experiment No 9**

**Program to add Gaussian Noise in the image**

**Figure 1:** Image with gaussian noise looks like an image given above.

import cv2

import numpy as np

from PIL import Image, ImageFilter

from numpy import sqrt

img = cv2.imread(’filename’)

for fil in [0.01,0.02,0.03,0.04,0.05,0.1,0.15,0.2,0.25]:

mean = 0

var=fil # var means variance

stddev = sqrt(var)

noise = np.zeros(img.shape, np.uint8)

cv2.randn(noise, mean, stddev)

noisy\_img = cv2.add(img, noise)

cv2.imwrite(’G-Noise’+str(fil)+’.jpg’, noisy\_img)

**Experiment No-9**

**Program to read images from the zip file**

import zipfile

from PIL import Image

imgzip = zipfile.ZipFile("100-Test.zip")

inflist = imgzip.infolist()

for f in inflist:

ifile = imgzip.open(f)

img = Image.open(ifile)

print(img)

**Program to apply Gaussian Blur on the single image**

from PIL import Image, ImageFilter

im=Image.open(’1.jpg’)

im=im.filter(ImageFilter.GaussianBlur(30))

im.show()



**To apply operations on images using PIL library**

from PIL import Image, ImageFilter

import os

size=(224,224)

for f in os.listdir(’.’):

if f.endswith(’.jpg’):

im1=Image.open(f)

fn,f1=os.path.splitext(f)

print(fn)

im1.thumbnail(size)

im1=im1.rotate(45)

im1=im1.convert(mode=’L’)

im1=im1.filter(ImageFilter.GaussianBlur(4))

im1.save(‘ping/{}.png’.format(fn))



**Experiment No-10**

**Plot a scatter plot on Iris Dataset.**

import pandas as pd

import matplotlib.pyplot as plt

data = pd.read\_csv("Iris.csv")

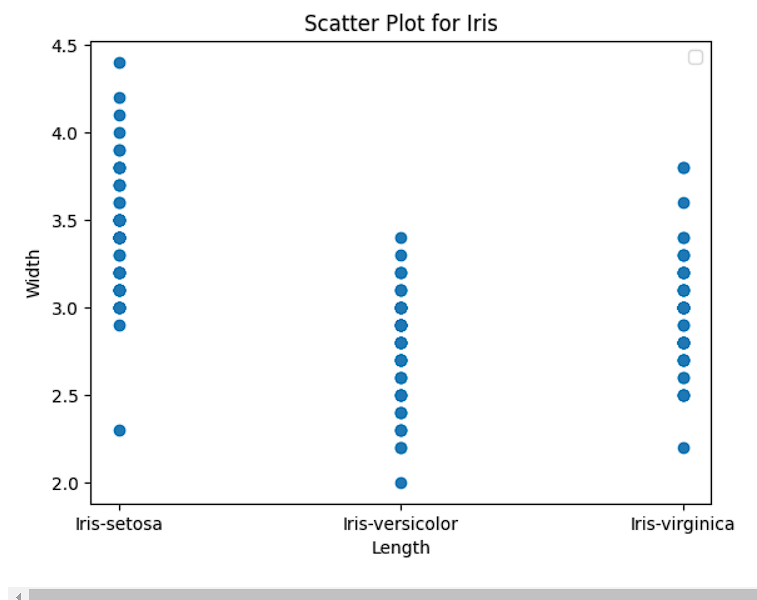
plt.scatter(data['Species'], data['SepalLengthCm'])

plt.title("Scatter Plot")

plt.xlabel('Species')

plt.ylabel('SepalLengthCm')

plt.show()



**Plot a scatter plot on Tips Dataset.**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

data = sns.load\_dataset("tips")

plt.scatter(data['day'], data['tip'], c=data['size'],s=data['total\_bill'])

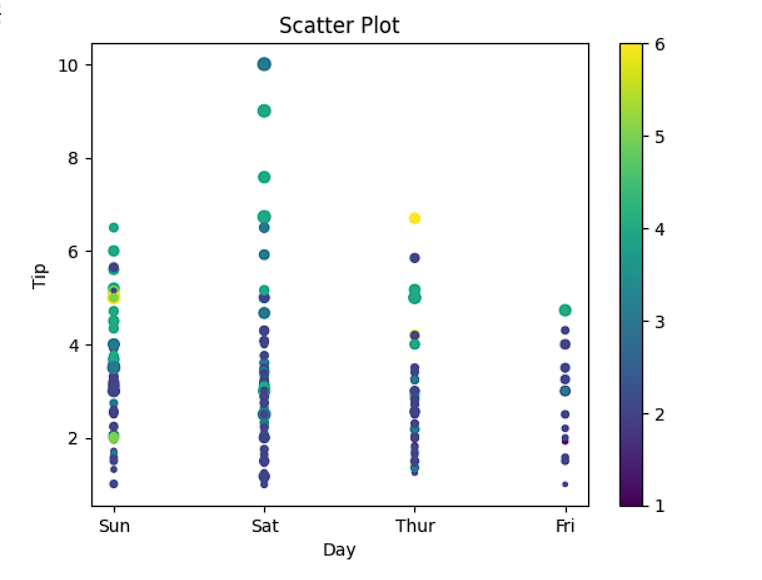
plt.title("Scatter Plot")

plt.xlabel('Day')

plt.ylabel('Tip')

plt.colorbar()

plt.show()



**Plot a Bar plot on Tips Dataset.**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

data = sns.load\_dataset("tips")

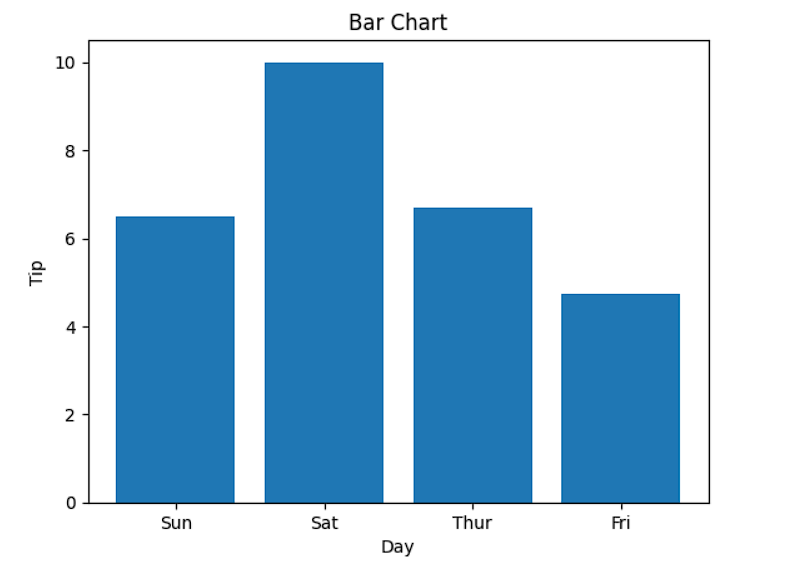
plt.bar(data['day'], data['tip'])

plt.title("Bar Chart")

plt.xlabel('Day')

plt.ylabel('Tip')

plt.show()



**A program to print the number of samples and number of features.**

from sklearn.datasets import load\_wine

wdata = load\_wine()

X, y = wdata[’data’], wdata[’target’]

print(X.shape)

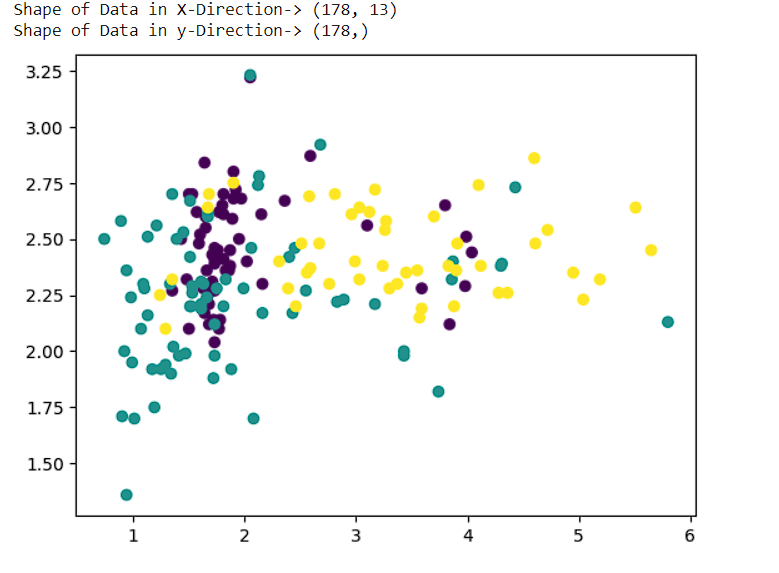
print(y.shape)

Print the data in two dimension.

import matplotlib.pyplot as plt

plt.scatter(X[:,1], X[:,2], c=y)

plt.show()

****

**Experiment No-11**

**A program to print the principal components for the given data.**

from sklearn.decomposition import PCA

pca = PCA()

Xt = pca.fit\_transform(X)

plot = plt.scatter(Xt[:,0], Xt[:,1], c=y)

plt.legend(handles=plot.legend\_elements()[0], labels=list(winedata[’target\_names’]))

plt.show()



**Experiment No-12**

import os

import sys

from PIL import Image

import imagehash

from imagehash import phash

import cv2

import glob

import os

from os import listdir

# get the path/directory

l=[]

m=[]

n=[]

for img1 in sorted(glob.glob("n/\*.jpg")):

img1 = cv2.imread(img1)

img1 = Image.fromarray(img1)

l.append(phash(img1))

for i in range(len(l)):

for j in range(len(l)):

print((l[i]-l[j]))

import numpy as np

from PIL import Image

import imagehash

from imagehash import phash,whash

import cv2

import glob

import os

l=[]

for img1 in glob.glob("da1/\*.png"):

img1 = cv2.imread(img1)

img1 = Image.fromarray(img1)

l.append(phash(img1))

count=0

for i in range(len(l)):

for j in range(len(l)):

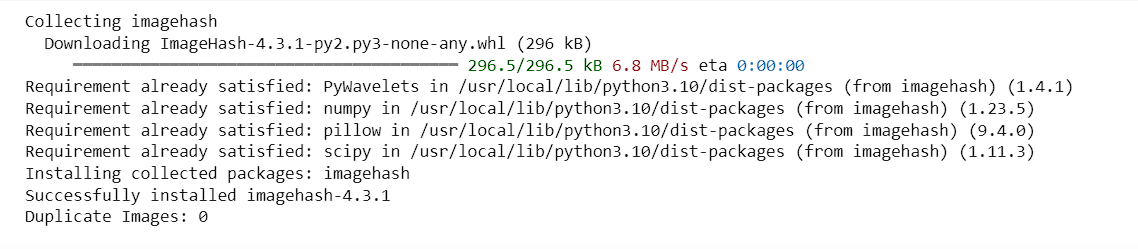
if i!=j:

d=l[i]-l[j]

if d<10:

count=count+1

print("Duplicate Images:",count)



from PIL import Image import glob

count=0

images=glob.glob("da/\*.png")

for image in images:

count=count+1

im=Image.open(image)

im.save('da1/'+str(count)+'.png',

optimize = True,

quality = 80)

print("done")

**Experiment No-13**

**Implementing the Progran to calculate Average of distance of images(Phash)**

import numpy as np

from PIL import Image

!pip install imagehash

import imagehash

from imagehash import phash,whash

import cv2

import glob

import os

l=[]

i=0;

for img1 in glob.glob("/content/drive/MyDrive/test/\*.jpg"):

    img1 = cv2.imread(img1)

    img1 = Image.fromarray(img1)

    l.append(phash(img1))

l2=[]

for img1 in glob.glob("/content/drive/MyDrive/test/\*.jpg"):

    img1 = cv2.imread(img1)

    img1 = Image.fromarray(img1)

    l2.append(phash(img1))

count=0

avg=[]

for i in range(len(l)):

    for j in range(len(l2)):

        d=(l[i]-l2[j])/64

        avg.append(d)

print("Average of distance of images:", sum(avg)/len(avg))



**Implementing the Progran to calculate Average of distance of images(Whash)**

l=[]

i=0;

for img1 in glob.glob("/content/drive/MyDrive/test/\*.jpg"):

    img1 = cv2.imread(img1)

    img1 = Image.fromarray(img1)

    l.append(whash(img1))

l2=[]

for img1 in glob.glob("/content/drive/MyDrive/test/\*.jpg"):

    img1 = cv2.imread(img1)

    img1 = Image.fromarray(img1)

    l2.append(whash(img1))

count=0

avg=[]

for i in range(len(l)):

    for j in range(len(l2)):

        d=(l[i]-l2[j])/64

        avg.append(d)

print("Average of distance of images:", sum(avg)/len(avg))



**Plot the Histogram of Average Distance**

import matplotlib.pyplot as plt

plt.hist(avg, bins=20, color='blue', alpha=0.7)

plt.xlabel('Average Distance')

plt.ylabel('Frequency')

plt.title('Histogram of Average Distances')

plt.show()

****

**Plot the Histogram with Gaussian Distribution**

plt.figure(figsize=(8, 6))

plt.hist(avg, bins=40, range=(0, 1.0), alpha=0.5, color='blue', label='Average Hash Differences')

mu = np.mean(avg)

sigma = np.std(avg)

print("Avrage of:",mu)

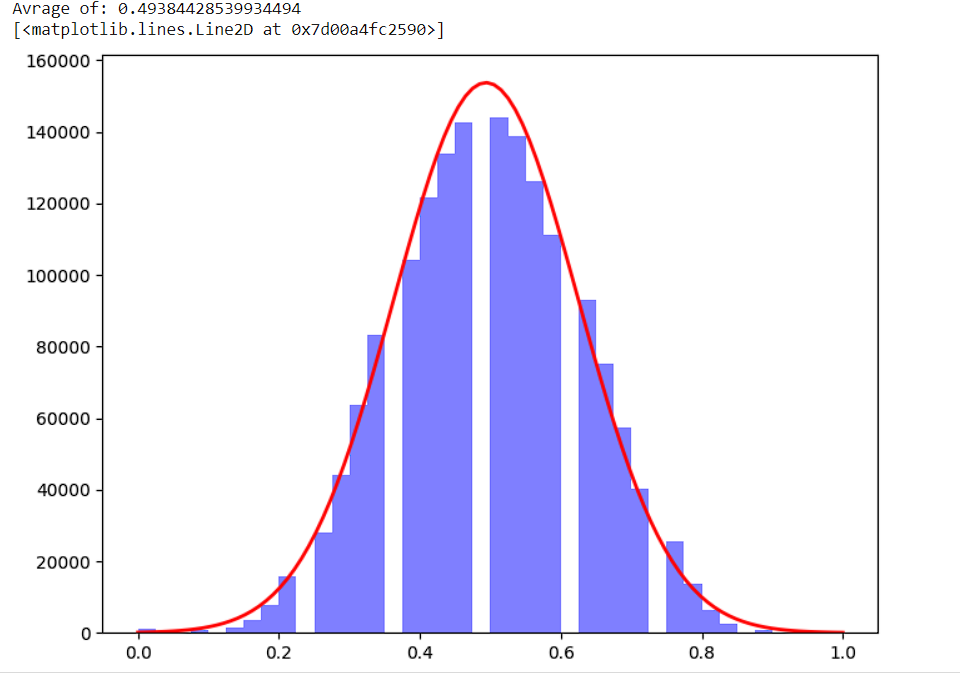
x = np.linspace(0, 1.0, 100)

pdf = 1 / (sigma \* np.sqrt(2 \* np.pi)) \* np.exp(-(x - mu) \*\* 2 / (2 \* sigma \*\* 2))

pdf = pdf \* (len(avg) / 31.5)

# Plot the Gaussian curve

plt.plot(x, pdf, 'r', linewidth=2, label='Gaussian Curve')



**Matlab:**

**Code:**

1) Image importation

temp = imread('name of the image;') %This will assign the image to i

imshow(temp);

A screenshot of a computer

Description automatically generated

2) **Enhancement of an image**

Image filtering

i = imread('image\_name.jpg');

imshow(i)

Red = i(:,:,1);

Green = i(:,:,2);

Blue = i(:,:,3);

temp = i;

imhist(Red);

A screenshot of a computer

Description automatically generated

Example 2:

figure;

temp = i;

temp(:,:,1) = temp(:,:,1) + 100;

imshow(temp);

A screenshot of a computer

Description automatically generated

3) **Image deblurring**

Deblurring an image increases the clarity of the image by making blurry pixels on the image more

visible

temp = imread('image\_name.jpg'); %this code imports the image

imshow(temp)

Create a blurred image from the original image i. This image will be used as a simulator for the

deblurring processes.

Begin by making a point spread function (PSF) by using the fspecial function with specified linear

motion (for my case I will use 50 pixels at an angle of 10 degrees) then convolve the PSF with the

image by using imfilter function.

PSF = fspecial('motion',50,10);

Idouble = im2double(i);

blur= imfilter(Idouble,PSF,'conv','circular');

imshow(blur)

A screenshot of a computer

Description automatically generated

clr1 = deconvwnr(blur,PSF);

imshow(clr1)

A screenshot of a computer

Description automatically generated

**Image segmentation with 1st order derivative**

**Code:**

% Read the input image

inputImage = imread('C:\Users\akhil\Downloads\3.jpg');

% Convert the image to grayscale (if it's not already)

grayImage = rgb2gray(inputImage);

% Apply Gaussian smoothing to reduce noise (optional but recommended)

sigma = 1; % Adjust the value according to your image

smoothedImage = imgaussfilt(grayImage, sigma);

% Perform edge detection using the Canny edge detector

edgeImage = edge(smoothedImage, 'Canny');

% Display the original image and the segmented edges

subplot(1, 2, 1);

imshow(inputImage);

title('Original Image');

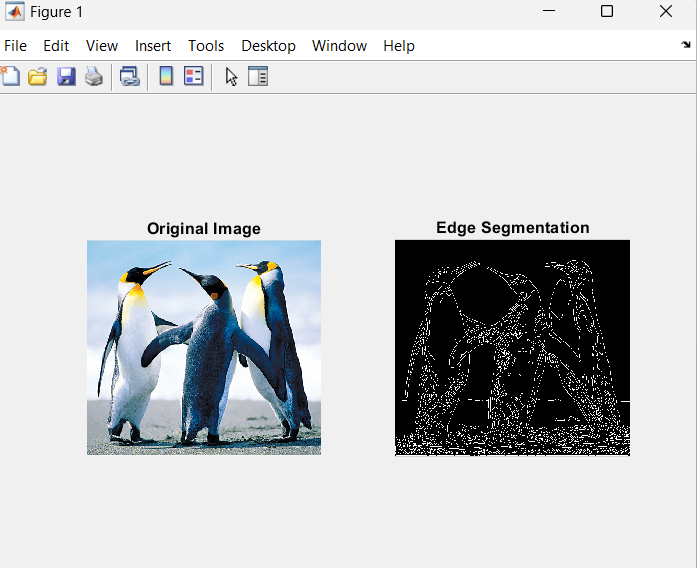
subplot(1, 2, 2);

imshow(edgeImage);

title('Edge Segmentation');

% You can save the edge image to a file if needed

% imwrite(edgeImage, 'edge\_segmentation.jpg');



**Image segmentation with 2nd order dewrivative**

**Code:**

% Read the input image

inputImage = imread('C:\Users\akhil\Downloads\3.jpg');

% Convert the image to grayscale (if it's not already)

grayImage = rgb2gray(inputImage);

% Apply Gaussian smoothing to reduce noise

sigma = 1.5; % Adjust the value according to your image

smoothedImage = imgaussfilt(double(grayImage), sigma);

% Compute the Laplacian of Gaussian

logImage = -del2(smoothedImage);

% Threshold the Laplacian of Gaussian to create a binary mask

threshold = 0.1; % Adjust the threshold as needed

binaryMask = logImage > threshold;

% Display the original image and the segmented edges

subplot(1, 2, 1);

imshow(inputImage);

title('Original Image');

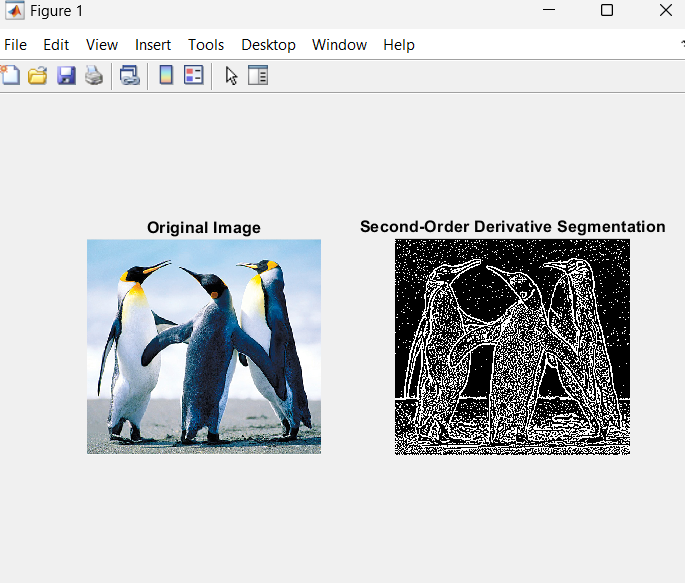
subplot(1, 2, 2);

imshow(binaryMask);

title('Second-Order Derivative Segmentation');

% You can save the binary mask to a file if needed

% imwrite(binaryMask, 'second\_order\_derivative\_segmentation.jpg');

****

**Image Clustering from Clustering**

% Load an image

image = imread('C:\Users\akhil\Downloads\3.jpg'); % Replace 'your\_image.jpg' with the path to your image

% Convert the image to double precision for processing

image = double(image);

% Reshape the image to a 2D matrix (rows x columns) x 3 (RGB channels)

[m, n, ~] = size(image);

image\_reshaped = reshape(image, m \* n, 3);

% Number of clusters (you can adjust this)

num\_clusters = 5;

% Perform k-means clustering

[cluster\_indices, cluster\_centers] = kmeans(image\_reshaped, num\_clusters);

% Reshape the cluster indices back to the original image size

segmented\_image = reshape(cluster\_indices, m, n);

% Display the segmented image

figure;

imshow(uint8(segmented\_image), []); % Convert to uint8 for display

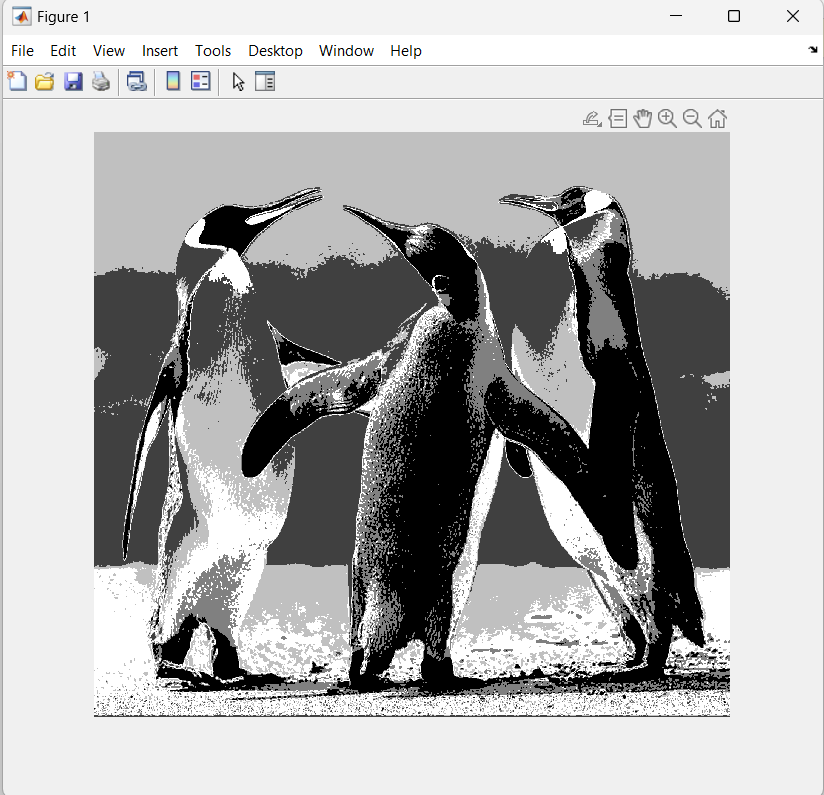
% You can also color the segments using cluster\_centers

colored\_image = cluster\_centers(segmented\_image, :);

% Display the colored image

figure;

imshow(uint8(colored\_image), []); % Convert to uint8 for display



% Load an image

image = imread('C:\Users\akhil\Downloads\Xray\_share.jpg'); % Replace 'your\_image.jpg' with the path to your image

% Convert the image to double precision for processing

image = double(image);

% Reshape the image to a 2D matrix (rows x columns) x 3 (RGB channels)

[m, n, ~] = size(image);

image\_reshaped = reshape(image, m \* n, 3);

% Number of clusters (you can adjust this)

num\_clusters = 5;

% Perform k-means clustering

[cluster\_indices, cluster\_centers] = kmeans(image\_reshaped, num\_clusters);

% Reshape the cluster indices back to the original image size

segmented\_image = reshape(cluster\_indices, m, n);

% Display the segmented image

figure;

imshow(uint8(segmented\_image), []); % Convert to uint8 for display

% You can also color the segments using cluster\_centers

colored\_image = cluster\_centers(segmented\_image, :);

% Display the colored image

figure;

imshow(uint8(colored\_image), []); % Convert to uint8 for display