In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as  plt
from math import sqrt
import math
```

Question 1.2

In [2]:
```
df = pd.read_csv('C:/Users/nikhi/Desktop/Machine Learning/iris dataset kaggle.csv'
```

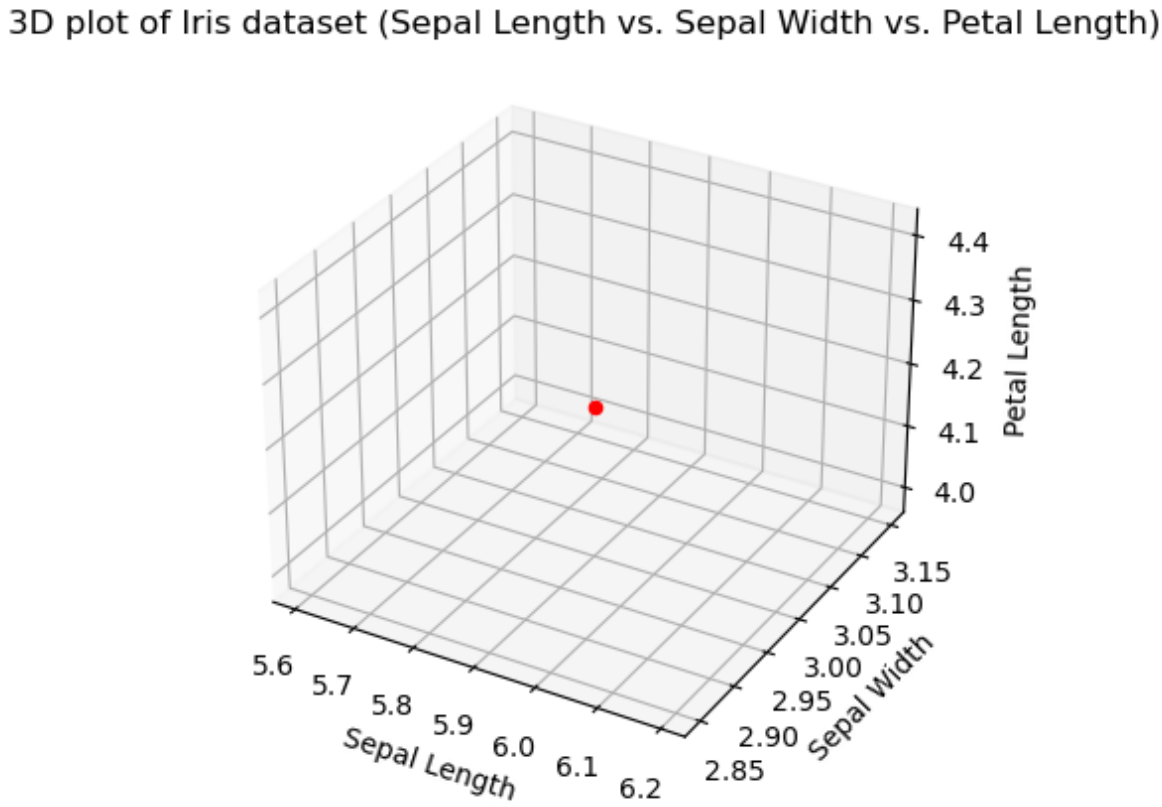
In [3]:
```
df.head()
```

Out[3]:

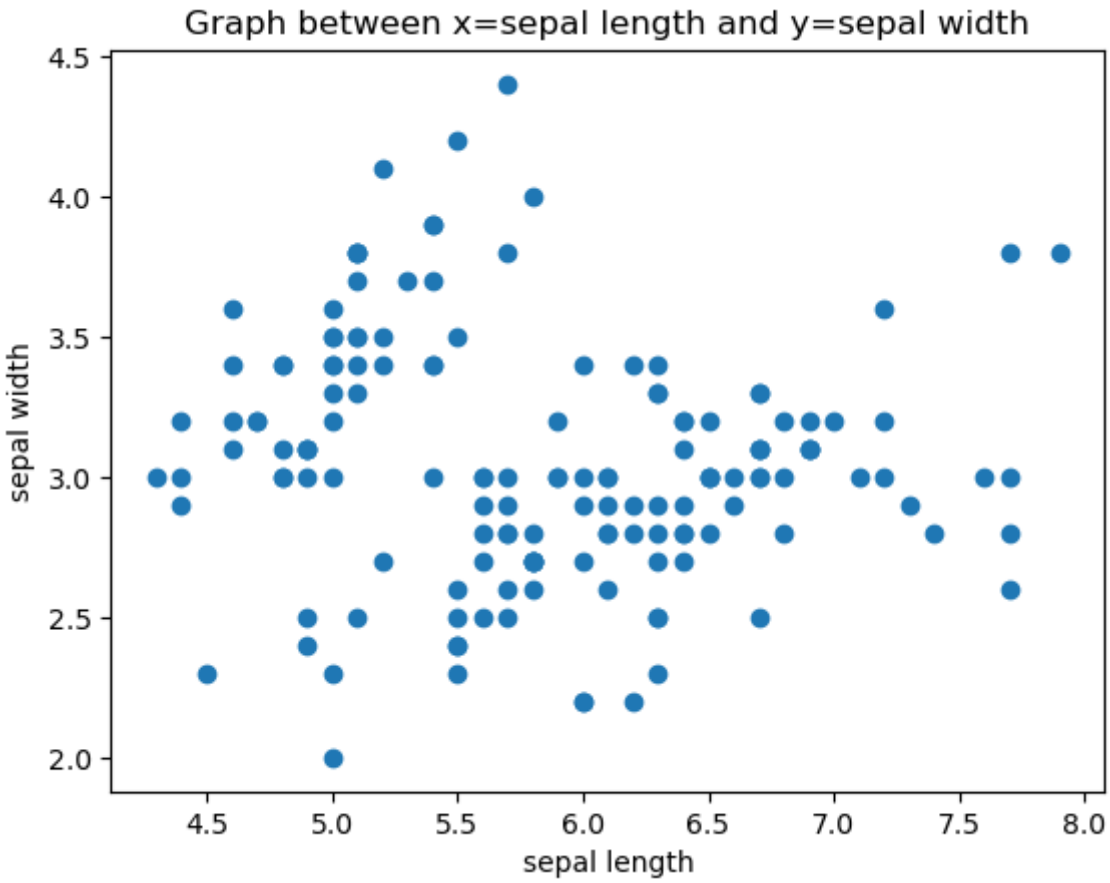
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In [4]:
```
data2 = df.drop(['petal_length','petal_width','species'],axis= 1)
data2.head()
data1 = df.drop(['species'],axis = 1)
data1.head()
data3 = data2.drop(['sepal_width'], axis = 1)
```

In [5]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Create a 3D plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Define the point
x, y, z = 5.9, 3.0, 4.2

# Plot the point
ax.scatter(x, y, z, c='r', marker='o')

# Add axis labels and a title
ax.set_xlabel('Sepal Length')
ax.set_ylabel('Sepal Width')
ax.set_zlabel('Petal Length')
ax.set_title('3D plot of Iris dataset (Sepal Length vs. Sepal Width vs. Petal Lengt

# Show the plot
plt.show()
```



In [6]:
```
plt.scatter(df['sepal_length'], df['sepal_width'])
plt.xlabel('sepal length')
plt.ylabel('sepal width')
plt.title('Graph between x=sepal length and y=sepal width')
plt.show()
```



Question 1.3

In [7]:
```
df.max
```

Out[7]:
```
<bound method NDFrame._add_numeric_operations.<locals>.max of      sepal_length  s
epal_width  petal_length  petal_width         species
0             5.1          3.5           1.4          0.2     Iris-setosa
1             4.9          3.0           1.4          0.2     Iris-setosa
2             4.7          3.2           1.3          0.2     Iris-setosa
3             4.6          3.1           1.5          0.2     Iris-setosa
4             5.0          3.6           1.4          0.2     Iris-setosa
..            ...          ...           ...          ...             ...
145           6.7          3.0           5.2          2.3  Iris-virginica
146           6.3          2.5           5.0          1.9  Iris-virginica
147           6.5          3.0           5.2          2.0  Iris-virginica
148           6.2          3.4           5.4          2.3  Iris-virginica
149           5.9          3.0           5.1          1.8  Iris-virginica

[150 rows x 5 columns]>
```

In [8]:
```
df.describe()
```

Out[8]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In [9]:
```
import numpy as np

vector_1 = np.array([5,3])
vector_2 = np.array([1,4])
difference = vector_1 - vector_2
dot_product = np.dot(difference.T, difference)
print(np.sqrt(dot_product))
```

```
4.123105625617661
```

In [10]:
```
#Angle between two vectors
vector_1 = np.array([5,3])
vector_2 = np.array([1,4])
k = np.dot(vector_1,vector_2)
print('dot_product = ',k)
mod_vector_1 = np.linalg.norm(vector_1)
print('norm of vector_1 = ',mod_vector_1)
mod_vector_2 = np.linalg.norm(vector_2)
print('norm of vector_2 = ',mod_vector_2)
cos_tita = k/(mod_vector_1*mod_vector_2)
print( 'cos_tita = ', cos_tita)
tita_radians = np.arccos(cos_tita)
print('tita in radians=', tita_radians)
#radian to degree
```

```
tita_degrees = math.degrees(tita_radians)
print('tita_degrees =' ,tita_degrees)
#tita_Degree = ((180*0.7853)/3.14)
#print(tita_Degree)
```

```
dot_product =  17
norm of vector_1 =  5.830951894845301
norm of vector_2 =  4.123105625617661
cos_tita =  0.7071067811865475
tita in radians= 0.7853981633974484
tita_degrees = 45.00000000000001
```

In [11]:
```
#l3 norm for vector A
l3_norm = np.linalg.norm(vector_1,ord = 3)
print('the l3 norm for vector 1 is',l3_norm)
```

```
the l3 norm for vector 1 is 5.336803297443889
```

In [12]:
```
l3_norm = np.linalg.norm(vector_1-vector_2, ord = 3)
print(l3_norm)
```

```
4.020725758589058
```

Question 2.1

In [13]:
```
np.mean(df['sepal_length'])
```

Out[13]:
```
5.843333333333335
```

In [14]:
```
np.median(df['sepal_length'])
```

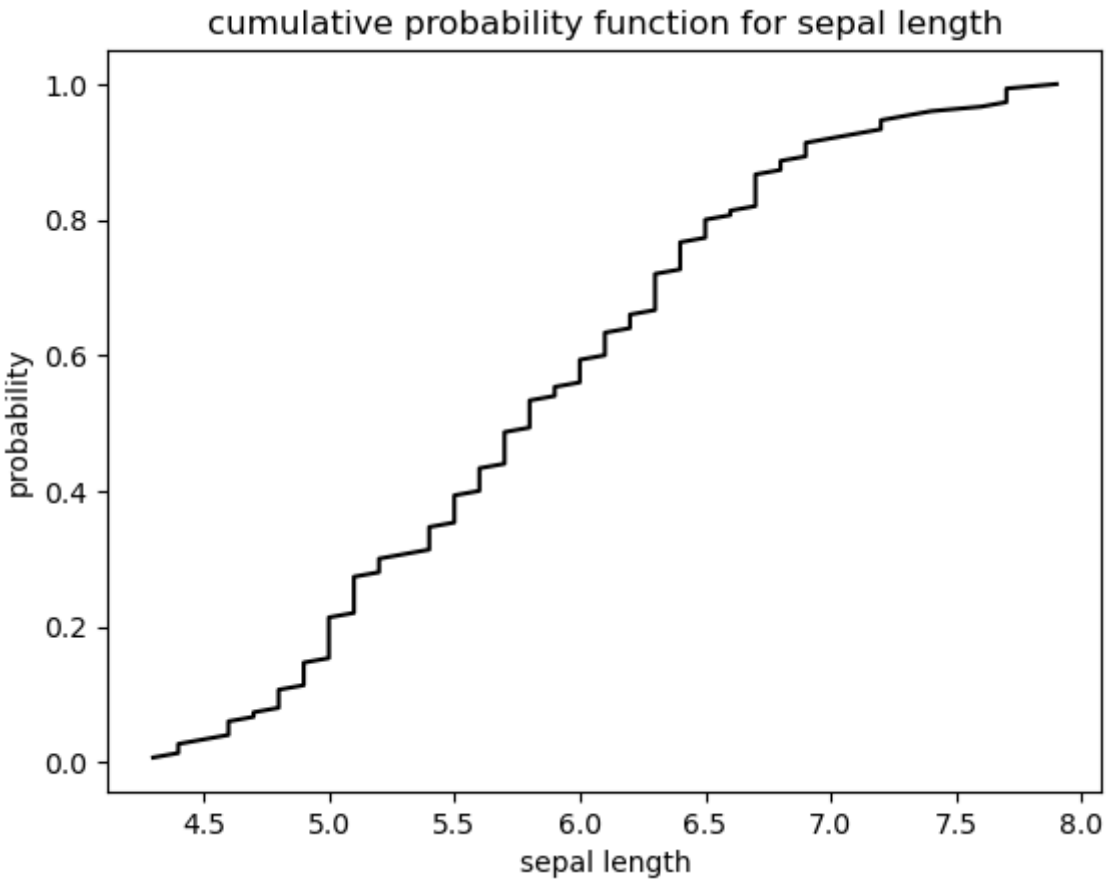
Out[14]:
```
5.8
```

In [15]:
```
import statistics
set1 =df['sepal_length']
print("Mode of given data set is % s" % (statistics.mode(set1)))
```

```
Mode of given data set is 5.0
```

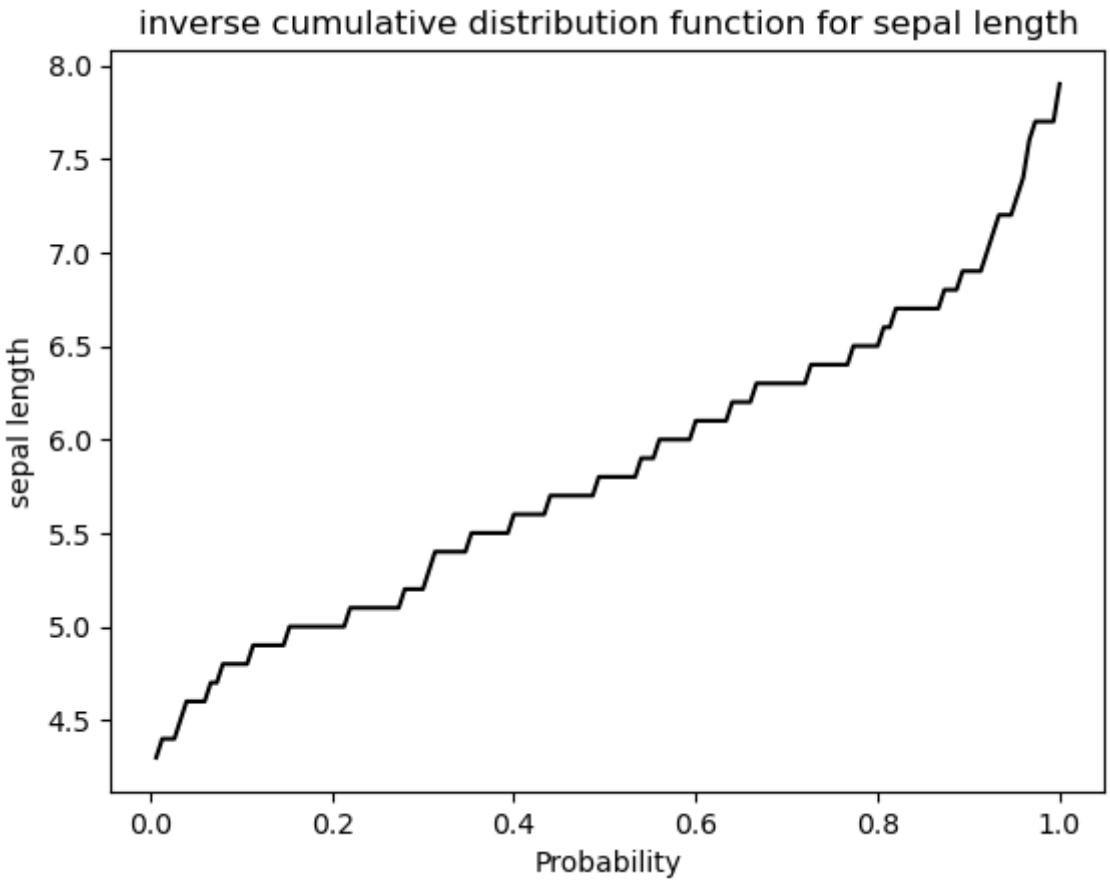
In [16]:
```
df = pd.read_csv('C:/Users/nikhi/Desktop/Machine Learning/iris dataset kaggle.csv'
df_new = np.array(df['sepal_length'])
df_new.sort()
yvals = np.zeros(len(df_new))
for i in range(len(df_new)):
    yvals[i] = (i+1)/len(yvals)
plt.xlabel('sepal length')
plt.ylabel('probability')
plt.title('cumulative probability function for sepal length')
plt.plot(df_new,yvals,'k-')
```

Out[16]:
```
[<matplotlib.lines.Line2D at 0x17d1f646ee0>]
```



In [17]:
```
df = pd.read_csv('C:/Users/nikhi/Desktop/Machine Learning/iris dataset kaggle.csv'
df_new = np.array(df['sepal_length'])
df_new.sort()
yvals = np.zeros(len(df_new))
for i in range(len(df_new)):
    yvals[i] = (i+1)/len(yvals)
plt.xlabel('Probability')
plt.ylabel('sepal length')
plt.title('inverse cumulative distribution function for sepal length')
plt.plot(yvals,df_new,'k-')
```

Out[17]:
```
[<matplotlib.lines.Line2D at 0x17d1f6b2c40>]
```



Question 2.3

In [18]:
```
vector_1 = np.array([5,3])
vector_2 = np.array([1,4])
np.dot(vector_1,vector_2)
```

Out[18]:
```
17
```

In [19]:
```
sample_mean_vector = np.array([5.843,3.054])
sample_covariance_matrix = np.array([[0.681, -0.039],[-0.039, 0.187]])
correlation = sample_covariance_matrix[0,1]/sqrt(sample_covariance_matrix[0,0]*sam
print(correlation)
```

```
-0.1092874387009562
```

In [20]:
```
#doubttful question
```

In [21]:
```
sample_covariance_vector1 = np.array([0.681,-0.039])
sample_covariance_vector2 = np.array([-0.039, 0.187])
dot_product = np.dot(sample_covariance_vector1,sample_covariance_vector2)
costita = dot_product/(np.linalg.norm(sample_covariance_vector1)*np.linalg.norm(sa
print(costita)
```

```
-0.2598000765469434
```

In [22]:
```
# in 2.3 costita not done
```

In [23]:
```
np.trace(sample_covariance_matrix)
```

Out[23]:
```
0.8680000000000001
```

In [24]:
```
np.linalg.det(sample_covariance_matrix)
```

Out[24]:
```
0.12582600000000002
```

In [ ]:
```
```

In [25]:
```
df.head()
```

Out[25]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In [26]:
```
data = df.drop(['petal_length','petal_width','species'],axis = 1)
data.head()
```

Out[26]:

	sepal_length	sepal_width
0	5.1	3.5
1	4.9	3.0
2	4.7	3.2
3	4.6	3.1
4	5.0	3.6

In [27]:
```
data.cov()
```

Out[27]:

	sepal_length	sepal_width
sepal_length	0.685694	-0.039268
sepal_width	-0.039268	0.188004

In [28]:
```
correlation_sepal_length_width = data.corr()
print(correlation_sepal_length_width)
```

```
              sepal_length  sepal_width
sepal_length      1.000000    -0.109369
sepal_width      -0.109369     1.000000
```

In [29]:
```
matrix = np.array([data.mean()])
print(matrix)
```

```
[[5.84333333 3.054     ]]
```

In [30]:
```
tita = np.arccos(-0.109369)
print(tita)
```

```
1.6803845464935199
```

In [31]:
```
tita_degrees = np.degrees(tita)
print(tita_degrees)
```

```
96.27894247308355
```

Question 2.4

In [32]:
```
#example 2.4
df.head()
```

Out[32]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In [33]:
```
data1 = df.drop(['species'],axis = 1)
data1.head()
```

Out[33]:

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [34]:
```
covariance_for_all_data = data1.cov()
```

In [35]:
```
data1.corr()
```

Out[35]:

	sepal_length	sepal_width	petal_length	petal_width
sepal_length	1.000000	-0.109369	0.871754	0.817954
sepal_width	-0.109369	1.000000	-0.420516	-0.356544
petal_length	0.871754	-0.420516	1.000000	0.962757
petal_width	0.817954	-0.356544	0.962757	1.000000

In [36]:
```
#Total Variance is the trace of the covariance matrix
np.trace(covariance_for_all_data)
```

Out[36]:
```
4.569291275167785
```

In [37]:
```
np.linalg.det(covariance_for_all_data)
```

Out[37]:
```
0.0019032757967392521
```

Question 2.5

In [38]:
```
D = np.array([[1,0.8], [5, 2.4],[9,5.5]])
I = np.array([[1],[1],[1]])
print(D)
print(I)
mean1 = D[:,0].mean()
mean2 = D[:,1].mean()
```

```
mean = np.array([[mean1],[mean2]])
print(mean1)
print(mean2)
print(mean)
print(len(D))
k = D.transpose()
D_mean = D-I*mean.transpose()
print(D_mean)
```

```
[[1.  0.8]
 [5.  2.4]
 [9.  5.5]]
[[1]
 [1]
 [1]]
5.0
2.9
[[5. ]
 [2.9]]
3
[[-4.  -2.1]
 [ 0.  -0.5]
 [ 4.   2.6]]
```

In [39]:
```
n = 3
Inner_D_mean =1/n*(np.dot(D_mean.T,D_mean))
print(Inner_D_mean)
outer_D_mean = 1/n*(np.outer(D_mean,D_mean))
print(outer_D_mean)
```

```
[[10.66666667  6.26666667]
 [ 6.26666667  3.80666667]]
[[ 5.33333333  2.8        -0.          0.66666667 -5.33333333 -3.46666667]
 [ 2.8         1.47       -0.          0.35       -2.8        -1.82      ]
 [-0.         -0.          0.         -0.          0.          0.        ]
 [ 0.66666667  0.35       -0.          0.08333333 -0.66666667 -0.43333333]
 [-5.33333333 -2.8         0.         -0.66666667  5.33333333  3.46666667]
 [-3.46666667 -1.82        0.         -0.43333333  3.46666667  2.25333333]]
```

In [40]:
```
df.head()
```

Out[40]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In [41]:
```
data2 = df.drop(['petal_length','petal_width','species'],axis= 1)
data2.head()
```

Out[41]:

	sepal_length	sepal_width
0	5.1	3.5
1	4.9	3.0
2	4.7	3.2
3	4.6	3.1
4	5.0	3.6

Question 2.8

In [42]:
```
k = np.array([data2['sepal_length'],data2['sepal_width']])
print(k)
sepal_length_mean = np.mean(data2['sepal_length'])
sepal_width_mean = np.mean(data2['sepal_width'])
print('/t')
mean = np.array([[sepal_length_mean],[sepal_width_mean]])
print(mean)
```

```
[[5.1 4.9 4.7 4.6 5.  5.4 4.6 5.  4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
  5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.  5.  5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.
  5.5 4.9 4.4 5.1 5.  4.5 4.4 5.  5.1 4.8 5.1 4.6 5.3 5.  7.  6.4 6.9 5.5
  6.5 5.7 6.3 4.9 6.6 5.2 5.  5.9 6.  6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
  6.3 6.1 6.4 6.6 6.8 6.7 6.  5.7 5.5 5.5 5.8 6.  5.4 6.  6.7 6.3 5.6 5.5
  5.5 6.1 5.8 5.  5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
  6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.  6.9 5.6 7.7 6.3 6.7 7.2
  6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.  6.9 6.7 6.9 5.8 6.8
  6.7 6.7 6.3 6.5 6.2 5.9]
 [3.5 3.  3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 3.4 3.  3.  4.  4.4 3.9 3.5
  3.8 3.8 3.4 3.7 3.6 3.3 3.4 3.  3.4 3.5 3.4 3.2 3.1 3.4 4.1 4.2 3.1 3.2
  3.5 3.1 3.  3.4 3.5 2.3 3.2 3.5 3.8 3.  3.8 3.2 3.7 3.3 3.2 3.2 3.1 2.3
  2.8 2.8 3.3 2.4 2.9 2.7 2.  3.  2.2 2.9 2.9 3.1 3.  2.7 2.2 2.5 3.2 2.8
  2.5 2.8 2.9 3.  2.8 3.  2.9 2.6 2.4 2.4 2.7 2.7 3.  3.4 3.1 2.3 3.  2.5
  2.6 3.  2.6 2.3 2.7 3.  2.9 2.9 2.5 2.8 3.3 2.7 3.  2.9 3.  3.  2.5 2.9
  2.5 3.6 3.2 2.7 3.  2.5 2.8 3.2 3.  3.8 2.6 2.2 3.2 2.8 2.8 2.7 3.3 3.2
  2.8 3.  2.8 3.  2.8 3.8 2.8 2.8 2.6 3.  3.4 3.1 3.  3.1 3.1 3.1 2.7 3.2
  3.3 3.  2.5 3.  3.4 3. ]]
/t
[[5.84333333]
 [3.054     ]]
```

Question 2.8

In [43]:
```
covariance_matrix = np.cov(k)
print('covariance matrix is',covariance_matrix)
x2 = np.array([[6.9],[3.1]])#consider a point (6.9,3.1)
variance_x2 = x2-mean
print('variance for x2 is',variance_x2)
covariance_inv = np.linalg.inv(covariance_matrix)
print('The covariance inverse matrix is = ', covariance_inv)
```

```
covariance matrix is [[ 0.68569351 -0.03926846]
 [-0.03926846  0.18800403]]
variance for x2 is [[1.05666667]
 [0.046     ]]
The covariance inverse matrix is =  [[1.47603328 0.30829951]
 [0.30829951 5.38342961]]
```

In [44]:
```
d = np.dot(variance_x2.T,covariance_inv)
mahalanobis_distance = np.dot(d,variance_x2)
print('the mahanalobis distance is',mahalanobis_distance)
```

```
the mahanalobis distance is [[1.68941892]]
```

In [45]:
```
l2 = np.linalg.norm(variance_x2)
l2_square = l2**2
print('the squared distance of x2 from mean is ',l2_square)
```

```
the squared distance of x2 from mean is  1.1186604444444428
```

In [46]:
```
l2_norm =np.linalg.norm(variance_x2)
print('the l2 norm of x2 vector is',l2_norm)
l2_norm_square = pow(l2_norm,2)
print('the euclidian square distance is',l2_norm_square)
```

```
the l2 norm of x2 vector is 1.0576674545642608
the euclidian square distance is 1.1186604444444428
```

In [47]:
```
eig_values,eigen_vector = np.linalg.eig(covariance_matrix)
print('the eigen vector for the covariance matrix is ',eigen_vector)
print('the eigen values for the covariance matrix is ',eig_values)
#covariance_matrix_new_axis = np.array([[eig_values[0,0],0],[0,eig_values[0,1]]])
covariance_matrix_new_axis = np.array([[0.6887728,0],[0,0.18492474]])
e1 = np.array([[1],[0]])
dot_product1 = np.dot(e1.T,covariance_matrix_new_axis)
print('the dot product is',dot_product1)
costita1 = dot_product1/(np.linalg.norm(e1.T)*np.linalg.norm(covariance_matrix_new_
print('the COSangle between the old axis and the new axis is',costita1)
tita_radians1 = np.arccos(0.96579)
print('tita_radians',tita_radians1)
tita_degrees1 = math.degrees(tita_radians1)
print('tita in degrees',tita_degrees1)
```

```
the eigen vector for the covariance matrix is  [[ 0.99693955  0.07817635]
 [-0.07817635  0.99693955]]
the eigen values for the covariance matrix is  [0.6887728  0.18492474]
the dot product is [[0.6887728 0.       ]]
the COSangle between the old axis and the new axis is [[0.96579648 0.        ]]
tita_radians 0.262323667392471
tita in degrees 15.030039007982161
```

In [ ]:
```
```

Question 2

In [48]:
```
import seaborn as  sns
from sklearn.datasets import load_iris
iris  = load_iris()
iris
np.mean(iris.data)
x1 = iris.data[1,:]
x1
mean = np.mean(iris.data,0)
mean
x1 = iris.data[1,:]
x1-mean
cov = np.cov(iris.data.T)
cov_inv = np.linalg.inv(cov)
cov_inv
d = np.dot((x1-mean).T,cov_inv)
mahalanobis_distance = np.dot(d,x1-mean)
mahalanobis_distance
```

Out[48]:
```
2.849118686158605
```

In [49]:

```
mahalanobis_distance = []
for i in range(len(iris.data)):
    x = iris.data[i,1]
    d = np.dot((x-mean).T,cov_inv)
    md = np.dot(d,cov)
    mahalanobis_distance.append(md)

print(mahalanobis_distance)

k = np.max(mahalanobis_distance)
k
sns.distplot(mahalanobis_distance)
```

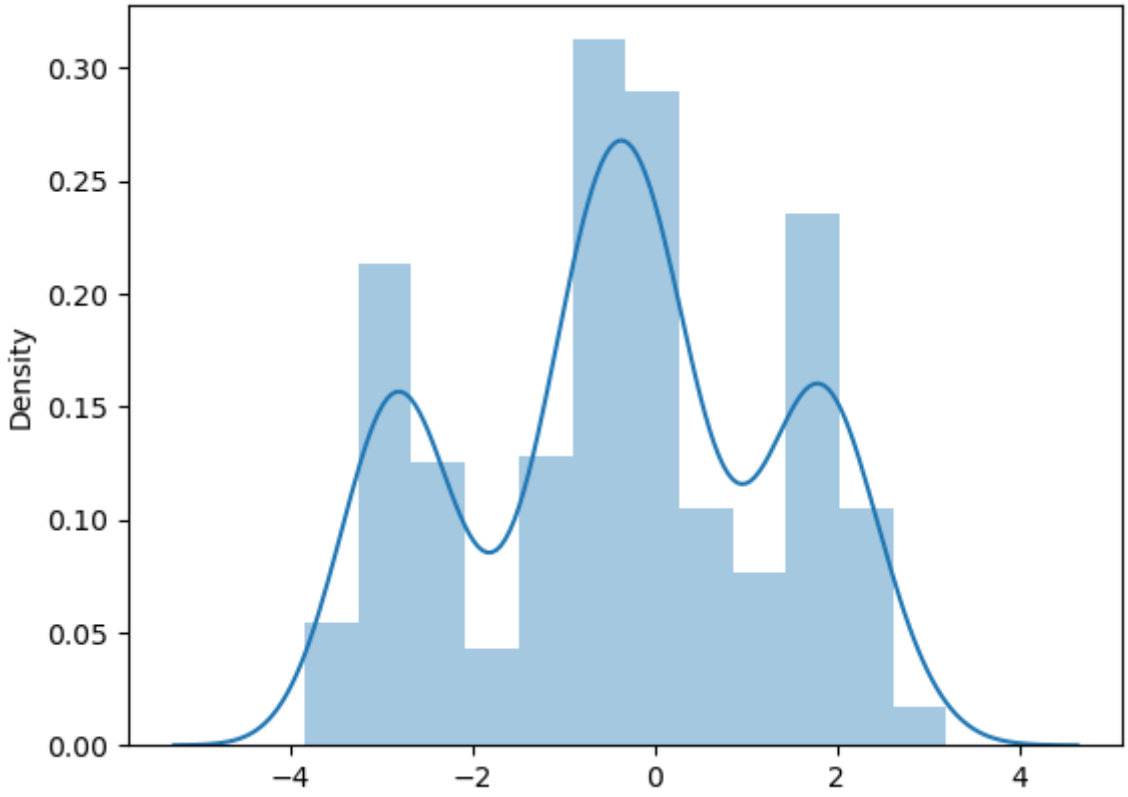
[array([-2.34333333, 0.44266667, -0.258   , 2.30066667]), array([-2.84333333,
-0.05733333, -0.758   , 1.80066667]), array([-2.64333333, 0.14266667, -0.558
, 2.00066667]), array([-2.74333333, 0.04266667, -0.658   , 1.90066667]), arra
y([-2.24333333, 0.54266667, -0.158   , 2.40066667]), array([-1.94333333, 0.84
266667, 0.142   , 2.70066667]), array([-2.44333333, 0.34266667, -0.358   ,
2.20066667]), array([-2.44333333, 0.34266667, -0.358   , 2.20066667]), array
([-2.94333333, -0.15733333, -0.858   , 1.70066667]), array([-2.74333333, 0.042
66667, -0.658   , 1.90066667]), array([-2.14333333, 0.64266667, -0.058   ,
2.50066667]), array([-2.44333333, 0.34266667, -0.358   , 2.20066667]), array
([-2.84333333, -0.05733333, -0.758   , 1.80066667]), array([-2.84333333, -0.057
33333, -0.758   , 1.80066667]), array([-1.84333333, 0.94266667, 0.242   ,
2.80066667]), array([-1.44333333, 1.34266667, 0.642   , 3.20066667]), array
([-1.94333333, 0.84266667, 0.142   , 2.70066667]), array([-2.34333333, 0.442
66667, -0.258   , 2.30066667]), array([-2.04333333, 0.74266667, 0.042   ,
2.60066667]), array([-2.04333333, 0.74266667, 0.042   , 2.60066667]), array
([-2.44333333, 0.34266667, -0.358   , 2.20066667]), array([-2.14333333, 0.642
66667, -0.058   , 2.50066667]), array([-2.24333333, 0.54266667, -0.158   ,
2.40066667]), array([-2.54333333, 0.24266667, -0.458   , 2.10066667]), array
([-2.44333333, 0.34266667, -0.358   , 2.20066667]), array([-2.84333333, -0.057
33333, -0.758   , 1.80066667]), array([-2.44333333, 0.34266667, -0.358   ,
2.20066667]), array([-2.34333333, 0.44266667, -0.258   , 2.30066667]), array
([-2.44333333, 0.34266667, -0.358   , 2.20066667]), array([-2.64333333, 0.142
66667, -0.558   , 2.00066667]), array([-2.74333333, 0.04266667, -0.658   ,
1.90066667]), array([-2.44333333, 0.34266667, -0.358   , 2.20066667]), array
([-1.74333333, 1.04266667, 0.342   , 2.90066667]), array([-1.64333333, 1.142
66667, 0.442   , 3.00066667]), array([-2.74333333, 0.04266667, -0.658   ,
1.90066667]), array([-2.64333333, 0.14266667, -0.558   , 2.00066667]), array
([-2.34333333, 0.44266667, -0.258   , 2.30066667]), array([-2.24333333, 0.542
66667, -0.158   , 2.40066667]), array([-2.84333333, -0.05733333, -0.758   ,
1.80066667]), array([-2.44333333, 0.34266667, -0.358   , 2.20066667]), array
([-2.34333333, 0.44266667, -0.258   , 2.30066667]), array([-3.54333333, -0.757
33333, -1.458   , 1.10066667]), array([-2.64333333, 0.14266667, -0.558   ,
2.00066667]), array([-2.34333333, 0.44266667, -0.258   , 2.30066667]), array
([-2.04333333, 0.74266667, 0.042   , 2.60066667]), array([-2.84333333, -0.057
33333, -0.758   , 1.80066667]), array([-2.04333333, 0.74266667, 0.042   ,
2.60066667]), array([-2.64333333, 0.14266667, -0.558   , 2.00066667]), array
([-2.14333333, 0.64266667, -0.058   , 2.50066667]), array([-2.54333333, 0.242
66667, -0.458   , 2.10066667]), array([-2.64333333, 0.14266667, -0.558   ,
2.00066667]), array([-2.64333333, 0.14266667, -0.558   , 2.00066667]), array
([-2.74333333, 0.04266667, -0.658   , 1.90066667]), array([-3.54333333, -0.757
33333, -1.458   , 1.10066667]), array([-3.04333333, -0.25733333, -0.958   ,
1.60066667]), array([-3.04333333, -0.25733333, -0.958   , 1.60066667]), array
([-2.54333333, 0.24266667, -0.458   , 2.10066667]), array([-3.44333333, -0.657
33333, -1.358   , 1.20066667]), array([-2.94333333, -0.15733333, -0.858   ,
1.70066667]), array([-3.14333333, -0.35733333, -1.058   , 1.50066667]), array
([-3.84333333, -1.05733333, -1.758   , 0.80066667]), array([-2.84333333, -0.057
33333, -0.758   , 1.80066667]), array([-3.64333333, -0.85733333, -1.558   ,
1.00066667]), array([-2.94333333, -0.15733333, -0.858   , 1.70066667]), array
([-2.94333333, -0.15733333, -0.858   , 1.70066667]), array([-2.74333333, 0.042
66667, -0.658   , 1.90066667]), array([-2.84333333, -0.05733333, -0.758   ,
1.80066667]), array([-3.14333333, -0.35733333, -1.058   , 1.50066667]), array
([-3.64333333, -0.85733333, -1.558   , 1.00066667]), array([-3.34333333, -0.557
33333, -1.258   , 1.30066667]), array([-2.64333333, 0.14266667, -0.558   ,
2.00066667]), array([-3.04333333, -0.25733333, -0.958   , 1.60066667]), array
([-3.34333333, -0.55733333, -1.258   , 1.30066667]), array([-3.04333333, -0.257
33333, -0.958   , 1.60066667]), array([-2.94333333, -0.15733333, -0.858   ,
1.70066667]), array([-2.84333333, -0.05733333, -0.758   , 1.80066667]), array
([-3.04333333, -0.25733333, -0.958   , 1.60066667]), array([-2.84333333, -0.057
33333, -0.758   , 1.80066667]), array([-2.94333333, -0.15733333, -0.858   ,
1.70066667]), array([-3.24333333, -0.45733333, -1.158   , 1.40066667]), array
([-3.44333333, -0.65733333, -1.358   , 1.20066667]), array([-3.44333333, -0.657
33333, -1.358   , 1.20066667]), array([-3.14333333, -0.35733333, -1.058   ,
1.50066667]), array([-3.14333333, -0.35733333, -1.058   , 1.50066667]), array
([-2.84333333, -0.05733333, -0.758   , 1.80066667]), array([-2.44333333, 0.342

```
66667, -0.358     ,  2.20066667]), array([-2.74333333,  0.04266667, -0.658     ,
1.90066667]), array([-3.54333333, -0.75733333, -1.458     ,  1.10066667]), array
([-2.84333333, -0.05733333, -0.758     ,  1.80066667]), array([-3.34333333, -0.557
33333, -1.258     ,  1.30066667]), array([-3.24333333, -0.45733333, -1.158     ,
1.40066667]), array([-2.84333333, -0.05733333, -0.758     ,  1.80066667]), array
([-3.24333333, -0.45733333, -1.158     ,  1.40066667]), array([-3.54333333, -0.757
33333, -1.458     ,  1.10066667]), array([-3.14333333, -0.35733333, -1.058     ,
1.50066667]), array([-2.84333333, -0.05733333, -0.758     ,  1.80066667]), array
([-2.94333333, -0.15733333, -0.858     ,  1.70066667]), array([-2.94333333, -0.157
33333, -0.858     ,  1.70066667]), array([-3.34333333, -0.55733333, -1.258     ,
1.30066667]), array([-3.04333333, -0.25733333, -0.958     ,  1.60066667]), array
([-2.54333333,  0.24266667, -0.458     ,  2.10066667]), array([-3.14333333, -0.357
33333, -1.058     ,  1.50066667]), array([-2.84333333, -0.05733333, -0.758     ,
1.80066667]), array([-2.94333333, -0.15733333, -0.858     ,  1.70066667]), array
([-2.84333333, -0.05733333, -0.758     ,  1.80066667]), array([-2.84333333, -0.057
33333, -0.758     ,  1.80066667]), array([-3.34333333, -0.55733333, -1.258     ,
1.30066667]), array([-2.94333333, -0.15733333, -0.858     ,  1.70066667]), array
([-3.34333333, -0.55733333, -1.258     ,  1.30066667]), array([-2.24333333,  0.542
66667, -0.158     ,  2.40066667]), array([-2.64333333,  0.14266667, -0.558     ,
2.00066667]), array([-3.14333333, -0.35733333, -1.058     ,  1.50066667]), array
([-2.84333333, -0.05733333, -0.758     ,  1.80066667]), array([-3.34333333, -0.557
33333, -1.258     ,  1.30066667]), array([-3.04333333, -0.25733333, -0.958     ,
1.60066667]), array([-2.64333333,  0.14266667, -0.558     ,  2.00066667]), array
([-2.84333333, -0.05733333, -0.758     ,  1.80066667]), array([-2.04333333,  0.742
66667,  0.042     ,  2.60066667]), array([-3.24333333, -0.45733333, -1.158     ,
1.40066667]), array([-3.64333333, -0.85733333, -1.558     ,  1.00066667]), array
([-2.64333333,  0.14266667, -0.558     ,  2.00066667]), array([-3.04333333, -0.257
33333, -0.958     ,  1.60066667]), array([-3.04333333, -0.25733333, -0.958     ,
1.60066667]), array([-3.14333333, -0.35733333, -1.058     ,  1.50066667]), array
([-2.54333333,  0.24266667, -0.458     ,  2.10066667]), array([-2.64333333,  0.142
66667, -0.558     ,  2.00066667]), array([-3.04333333, -0.25733333, -0.958     ,
1.60066667]), array([-2.84333333, -0.05733333, -0.758     ,  1.80066667]), array
([-3.04333333, -0.25733333, -0.958     ,  1.60066667]), array([-2.84333333, -0.057
33333, -0.758     ,  1.80066667]), array([-3.04333333, -0.25733333, -0.958     ,
1.60066667]), array([-2.04333333,  0.74266667,  0.042     ,  2.60066667]), array
([-3.04333333, -0.25733333, -0.958     ,  1.60066667]), array([-3.04333333, -0.257
33333, -0.958     ,  1.60066667]), array([-3.24333333, -0.45733333, -1.158     ,
1.40066667]), array([-2.84333333, -0.05733333, -0.758     ,  1.80066667]), array
([-2.44333333,  0.34266667, -0.358     ,  2.20066667]), array([-2.74333333,  0.042
66667, -0.658     ,  1.90066667]), array([-2.84333333, -0.05733333, -0.758     ,
1.80066667]), array([-2.74333333,  0.04266667, -0.658     ,  1.90066667]), array
([-2.74333333,  0.04266667, -0.658     ,  1.90066667]), array([-2.74333333,  0.042
66667, -0.658     ,  1.90066667]), array([-3.14333333, -0.35733333, -1.058     ,
1.50066667]), array([-2.64333333,  0.14266667, -0.558     ,  2.00066667]), array
([-2.54333333,  0.24266667, -0.458     ,  2.10066667]), array([-2.84333333, -0.057
33333, -0.758     ,  1.80066667]), array([-3.34333333, -0.55733333, -1.258     ,
1.30066667]), array([-2.84333333, -0.05733333, -0.758     ,  1.80066667]), array
([-2.44333333,  0.34266667, -0.358     ,  2.20066667]), array([-2.84333333, -0.057
33333, -0.758     ,  1.80066667])]
```

```
C:\Users\nikhi\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWa
rning: `distplot` is a deprecated function and will be removed in a future versio
n. Please adapt your code to use either `displot` (a figure-level function with si
milar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

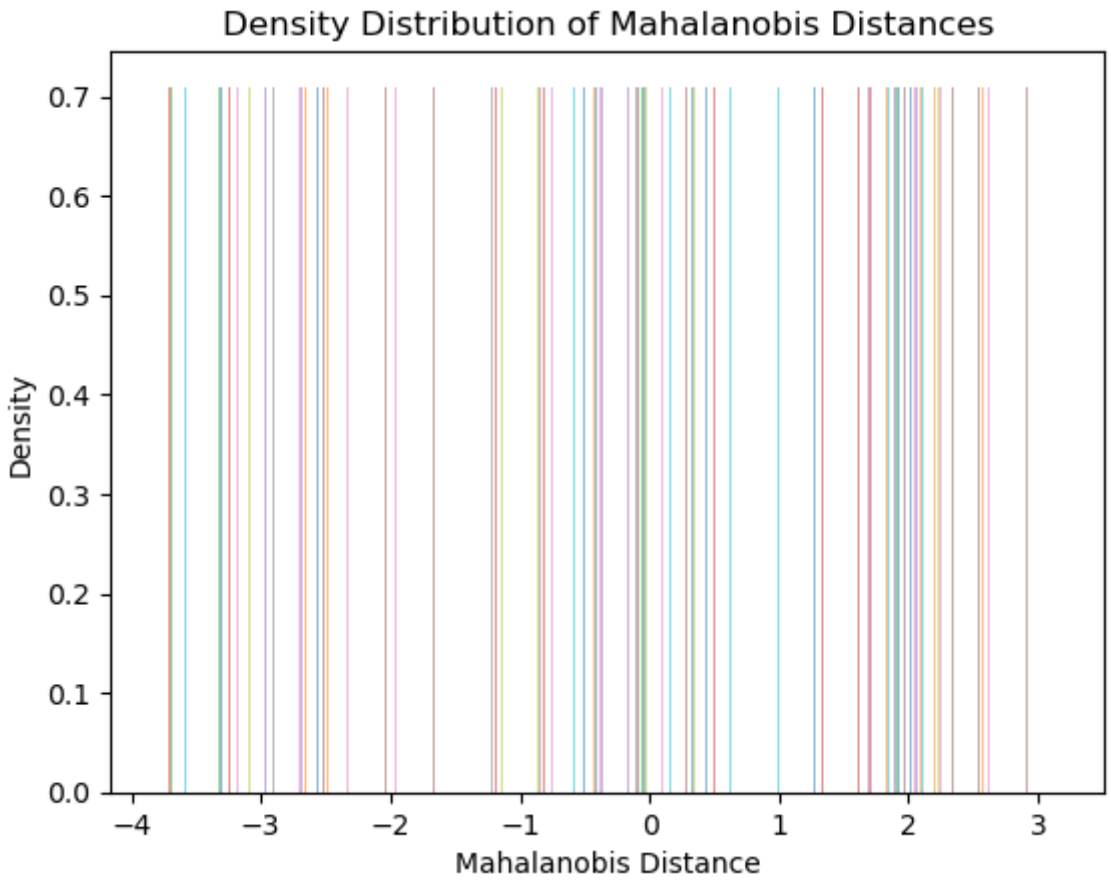
Out[49]:

```
<AxesSubplot:ylabel='Density'>
```



In [50]:
```
plt.hist(mahalanobis_distance, bins=20, density=True, alpha=0.5)
plt.xlabel('Mahalanobis Distance')
plt.ylabel('Density')
plt.title('Density Distribution of Mahalanobis Distances')
```

Out[50]:
```
Text(0.5, 1.0, 'Density Distribution of Mahalanobis\xa0Distances')
```



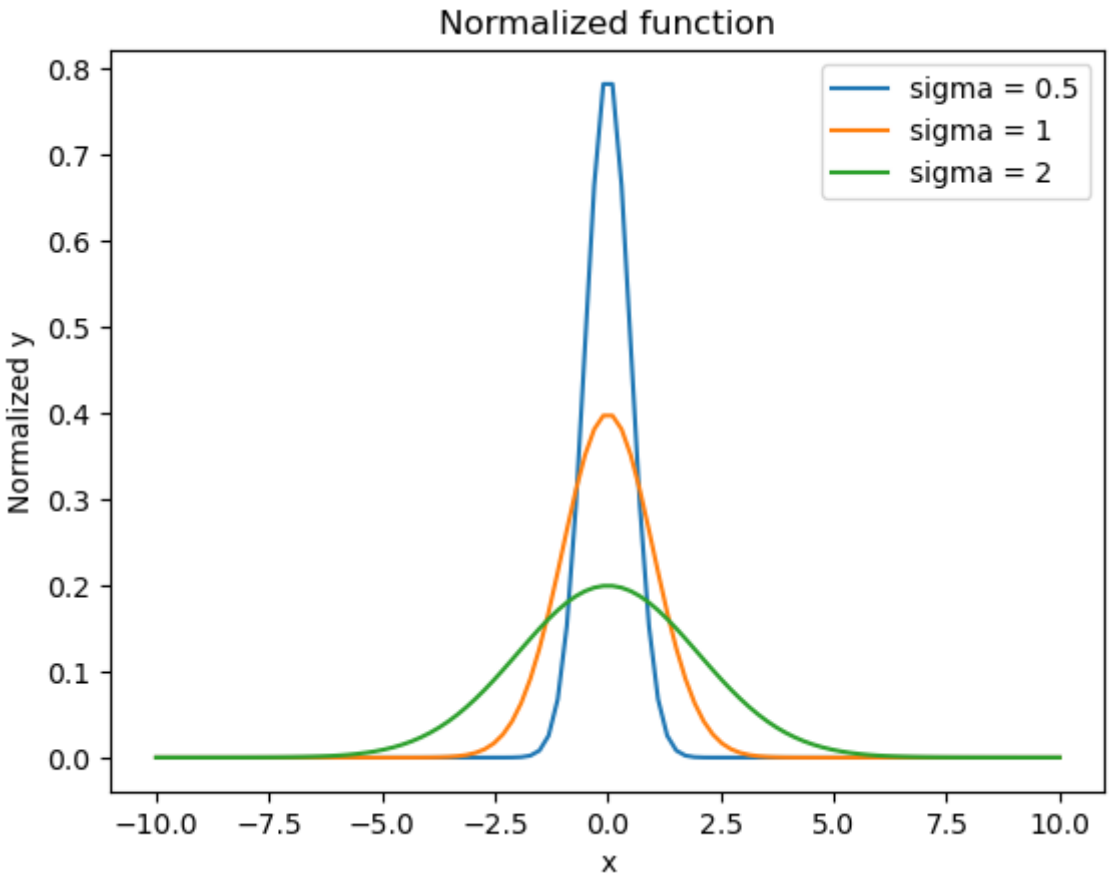
In [ ]:

In [51]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
iris = load_iris()
mean = np.mean(iris.data)
sigma = np.std(iris.data)
print(mean)
print(sigma)
def normalize(iris):
    return (iris-mean)/sigma
z = normalize(iris.data)
z_mean = np.mean(z)
print(z_mean)
z_std = np.std(z)
print(z_std)
```

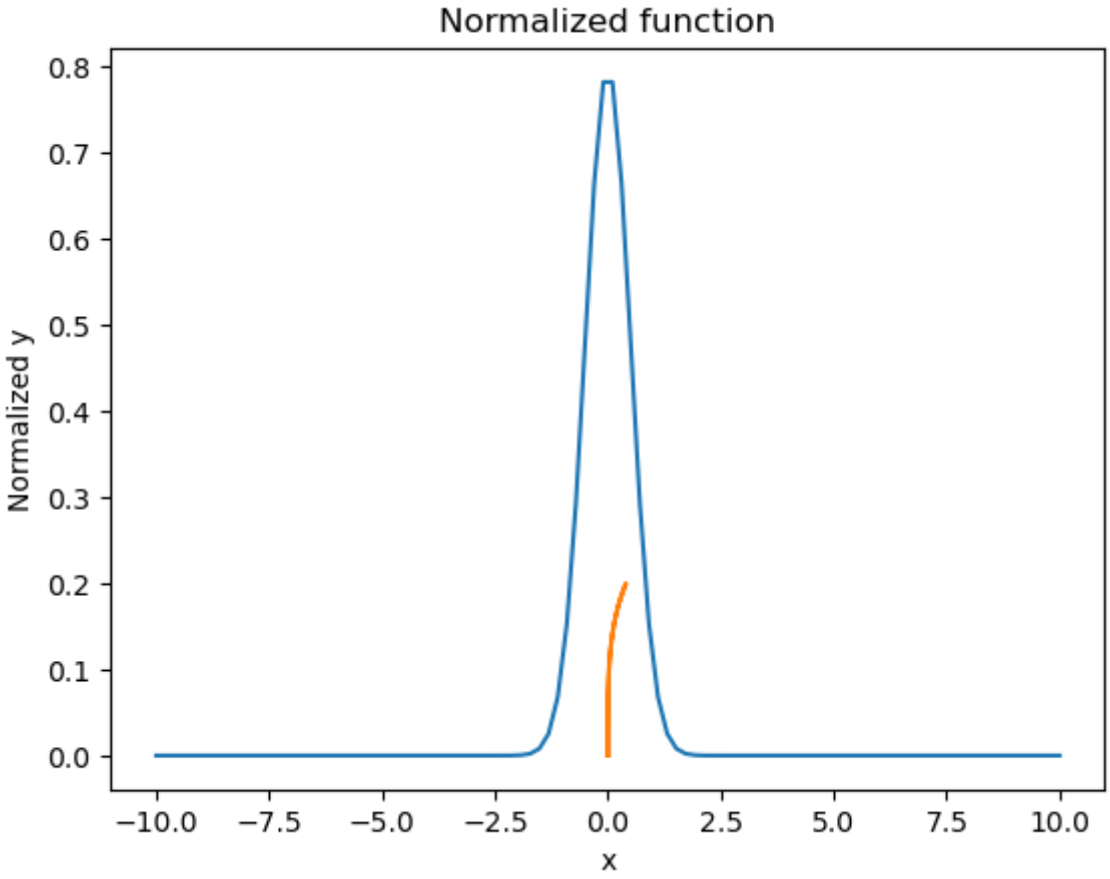
```
3.4644999999999997
1.9738430577598278
1.1842378929335003e-16
1.0
```

Question 2.7 Plotting the normalized function

In [52]:
```
import numpy as np
import matplotlib.pyplot as plt

def normalize(x, mu, sigma):
    return np.exp(-(x - mu)**2 / (2 * sigma**2)) / (sigma * np.sqrt(2 * np.pi))

x = np.linspace(-10, 10, 100)
y1 = normalize(x, 0, 0.5)
y2 = normalize(x, 0, 1)
y3 = normalize(x, 0, 2)

plt.plot(x, y1, label='sigma = 0.5')
plt.plot(x, y2, label='sigma = 1')
plt.plot(x, y3, label='sigma = 2')
plt.xlabel('x')
plt.ylabel('Normalized y')
plt.title('Normalized function')
plt.legend()
plt.show()
```



In [53]:
```
def normalize(x, mu, sigma):
    return np.exp(-(x - mu)**2 / (2 * sigma**2)) / (sigma * np.sqrt(2 * np.pi))
x = np.linspace(-10, 10, 100)
y1 = normalize(x, 0, 0.5)
y2 = normalize(x,0,1)
y3 = normalize(x,0,2)
plt.plot(x, y1,y2,y3)
plt.xlabel('x')
plt.ylabel('Normalized y')
plt.title('Normalized function')
plt.show()
```


Normalized function
Normalized y
x
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0