Pattern Processing using AI Practical File



COSCE60

Nikhil Gupta

2019UCO1719

COE Section 3

**Q1 Write a python program to implement a simple Chatbot.**

Code:

# %%

import json

import requests

# %%

def chatbot():

print("Hi! I'm a weather prediction chatbot")

api\_key = "f37d0c3cb6c045218e1152633232504"

while(1):

print("Which city would you like the weather forcast for? Type 'exit' to quit")

city = input().lower()

if(city=="exit"):

break

url = f"https://api.weatherapi.com/v1/forecast.json?key={api\_key}&q={city}"

response = requests.get(url)

data = json.loads(response.text)

if len(data)>1:

temp = data["current"]["temp\_c"]

description = data["current"]["condition"]["text"]

humidity = data["current"]["humidity"]

wind\_speed = data["current"]["wind\_kph"]

print(f"The weather in {city.title()} is {description}, with a temperature of {temp}°C , humidity of {humidity}%, and wind speed of {wind\_speed} km/h.")

print()

else:

print("City not found. Please try again.")

print()

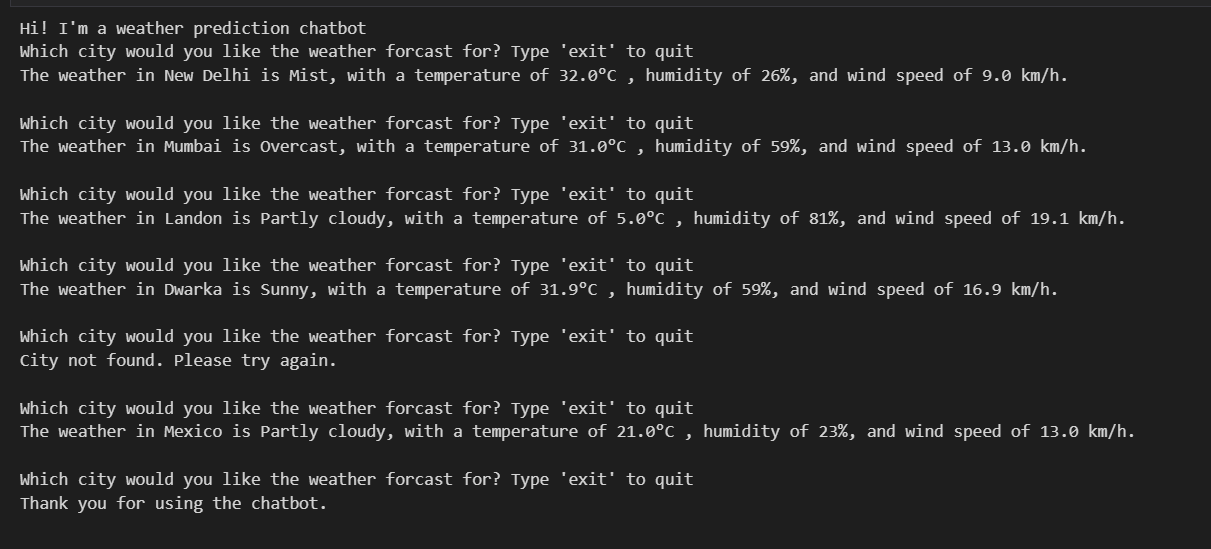
print("Thank you for using the chatbot.")

# %%

chatbot()

# %%

Output:



Q2 Write a program to implement k-means clustering from scratch.

Code:

# %%

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import make\_blobs

# %%

X,Y = make\_blobs(n\_samples=500,n\_features=2,centers=5,random\_state=3)

# %%

plt.figure(0)

plt.scatter(X[:,0],X[:,1],c=Y)

plt.show()

# %%

k = 5

color = ["green","red","yellow","blue","orange"]

clusters = {}

for i in range(k):

center = 10\*(2\*np.random.random((X.shape[1],))-1)

points = []

cluster = {

"center":center,

"points":points,

"color":color[i]

}

clusters[i] = cluster

# %%

print(clusters)

# %%

def distance(x1,x2):

return np.sqrt(np.sum((x1-x2)\*\*2))

def assignPointsToCluster(clusters):

for i in range(X.shape[0]):

clust\_x = X[i]

dist = []

for kx in range(k):

d = distance(clust\_x,clusters[kx]['center'])

dist.append(d)

idx = np.argmin(dist)

clusters[idx]['points'].append(clust\_x)

def updateCluster(clusters):

for kx in range(k):

pts = np.array(clusters[kx]['points'])

if(pts.shape[0]>0):

new\_centers = np.mean(pts,axis=0)

clusters[kx]['center'] = new\_centers

clusters[kx]['points'] = []

def plotClusters(clusters):

plt.figure()

for kx in range(k):

pts = np.array(clusters[kx]['points'])

try:

plt.scatter(pts[:,0],pts[:,1],color=clusters[kx]['color'])

except:

pass

cent = clusters[kx]['center']

plt.scatter(cent[0],cent[1],color='black',marker="\*")

# %%

epoch = 5

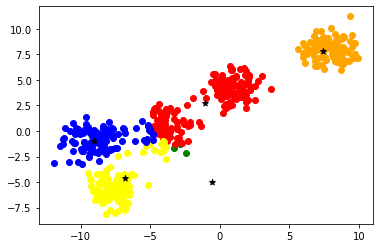
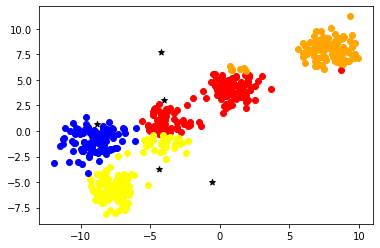
for i in range(epoch):

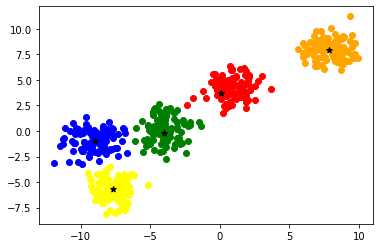
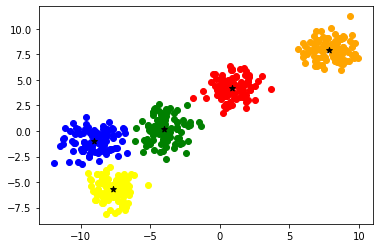
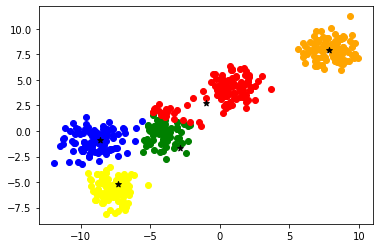
assignPointsToCluster(clusters)

plotClusters(clusters)

updateCluster(clusters)

# %%

Output:



Q3 Generating samples of Gaussian (normal) distributions and plotting them for visualization

Code:

# %%

import numpy as np

import matplotlib.pyplot as plt

# %%

x\_axis = np.arange(-100,100,0.1)

print(x\_axis)

# %%

mean = np.mean(x\_axis)

std = np.std(x\_axis)

print(mean,std)

# %%

y\_axis = 1/(std \* np.sqrt(2 \* np.pi)) \* np.exp( - (x\_axis - mean)\*\*2 / (2 \* std\*\*2) )

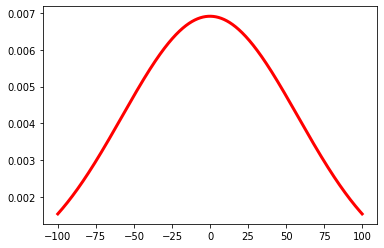
# %%

plt.plot(x\_axis,y\_axis,linewidth=3, color='r')

plt.show()

# %%

Output:



Q4 Implement Decision Tree algorithms.

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import export\_graphviz

import graphviz

from sklearn.tree import DecisionTreeClassifier

iris = datasets.load\_iris()

X = iris.data

y = iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state = 0, test\_size = 0.2)

tree = DecisionTreeClassifier(max\_depth = 5)

tree.fit(X\_train, y\_train)

tree\_predictions = tree.predict(X\_test)

cm = confusion\_matrix(y\_test, tree\_predictions)

print(cm)

score =accuracy\_score(tree\_predictions, y\_test)

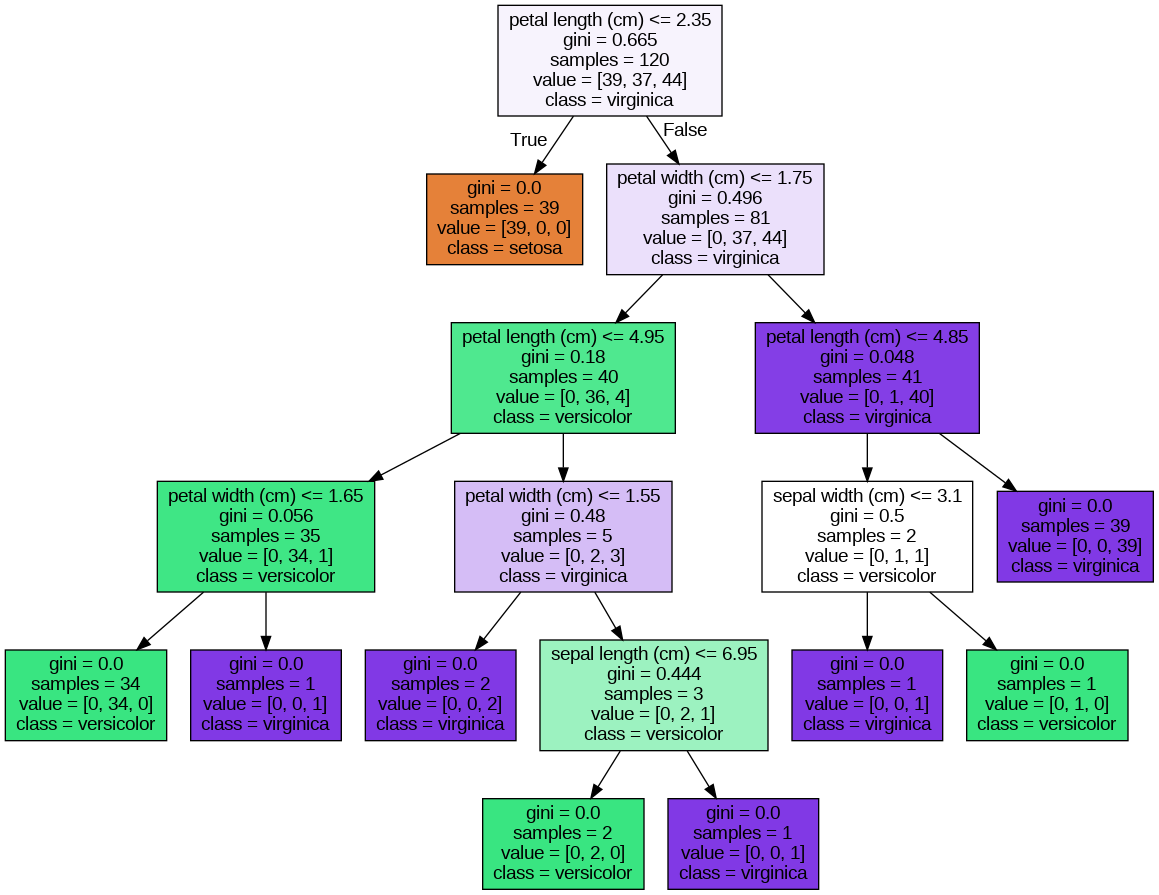
print(score)

tree\_formed = export\_graphviz(tree, out\_file = None, feature\_names = iris.feature\_names, class\_names = iris.target\_names, filled = True)

graph = graphviz.Source(tree\_formed, format="png")

graph

Output:



Q5 Implement SVM.

Code:

# %% [markdown]

# ### Generate Dataset

# %%

from sklearn.datasets import make\_classification

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

import numpy as np

# %%

X,Y = make\_classification(n\_classes=2,n\_samples=1000,n\_clusters\_per\_class=1,random\_state=3,n\_features=2,n\_informative=2,n\_redundant=0)

# %%

plt.scatter(X[:,0],X[:,1],c=Y)

plt.show()

# %%

plt.scatter(X[:,0],X[:,1],c=Y)

plt.show()

# %%

def make\_meshgrid(x, y, h=.02):

x\_min, x\_max = x.min() - 1, x.max() + 1

y\_min, y\_max = y.min() - 1, y.max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h), np.arange(y\_min, y\_max, h))

return xx, yy

def plot\_contours(ax, clf, xx, yy, \*\*params):

Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

out = ax.contourf(xx, yy, Z, \*\*params)

return out

def plot(model, X,Y,title):

fig, ax = plt.subplots()

xx, yy = make\_meshgrid(X[:,0],X[:,1])

plot\_contours(ax, model, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)

ax.scatter(X[:,0], X[:,1], c=Y, cmap=plt.cm.coolwarm, s=20, edgecolors='k')

ax.set\_title(title)

plt.show()

# %%

from sklearn import svm

# %%

svc = svm.SVC(kernel='linear')

svc.fit(X,Y)

print(svc.score(X,Y))

plot(svc,X,Y,"Linear kernal")

# %%

# RBF Kernel

svc = svm.SVC()

svc.fit(X,Y)

print(svc.score(X,Y))

plot(svc,X,Y,"RBF kernal")

# %%

# Polynomial Kernel

svc = svm.SVC(kernel='poly')

svc.fit(X,Y)

print(svc.score(X,Y))

plot(svc,X,Y,"Poly kernal")

# %%

def custom\_kernel(x1,x2):

return np.square(np.dot(x1,x2.T))

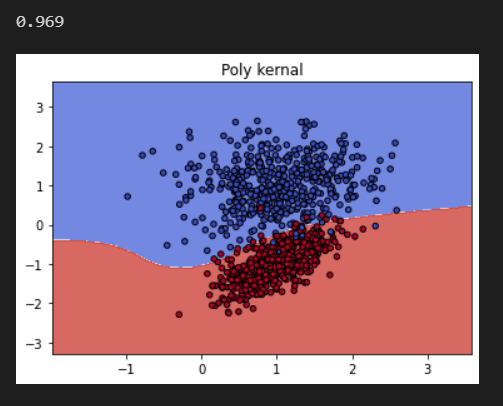
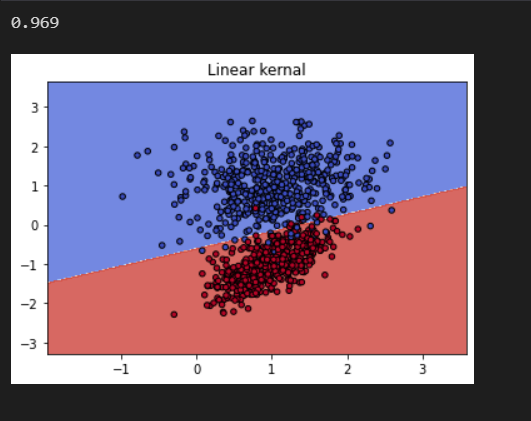
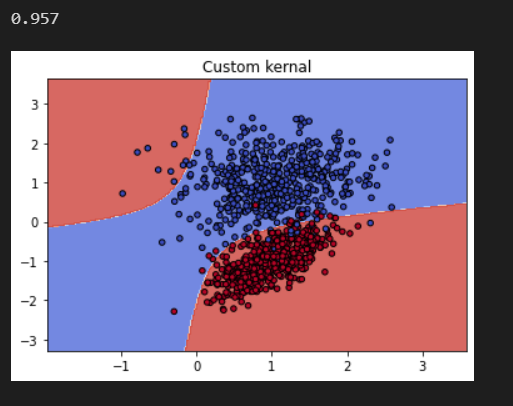
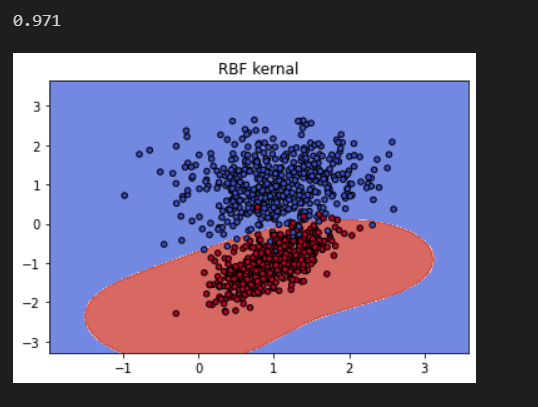
svc = svm.SVC(kernel=custom\_kernel)

svc.fit(X,Y)

print(svc.score(X,Y))

plot(svc,X,Y,"Custom kernal")

# %%

Output:

Q6 Implement agglomerative Hierarchical clustering.

Code:

from clustimage import Clustimage

cl = Clustimage()

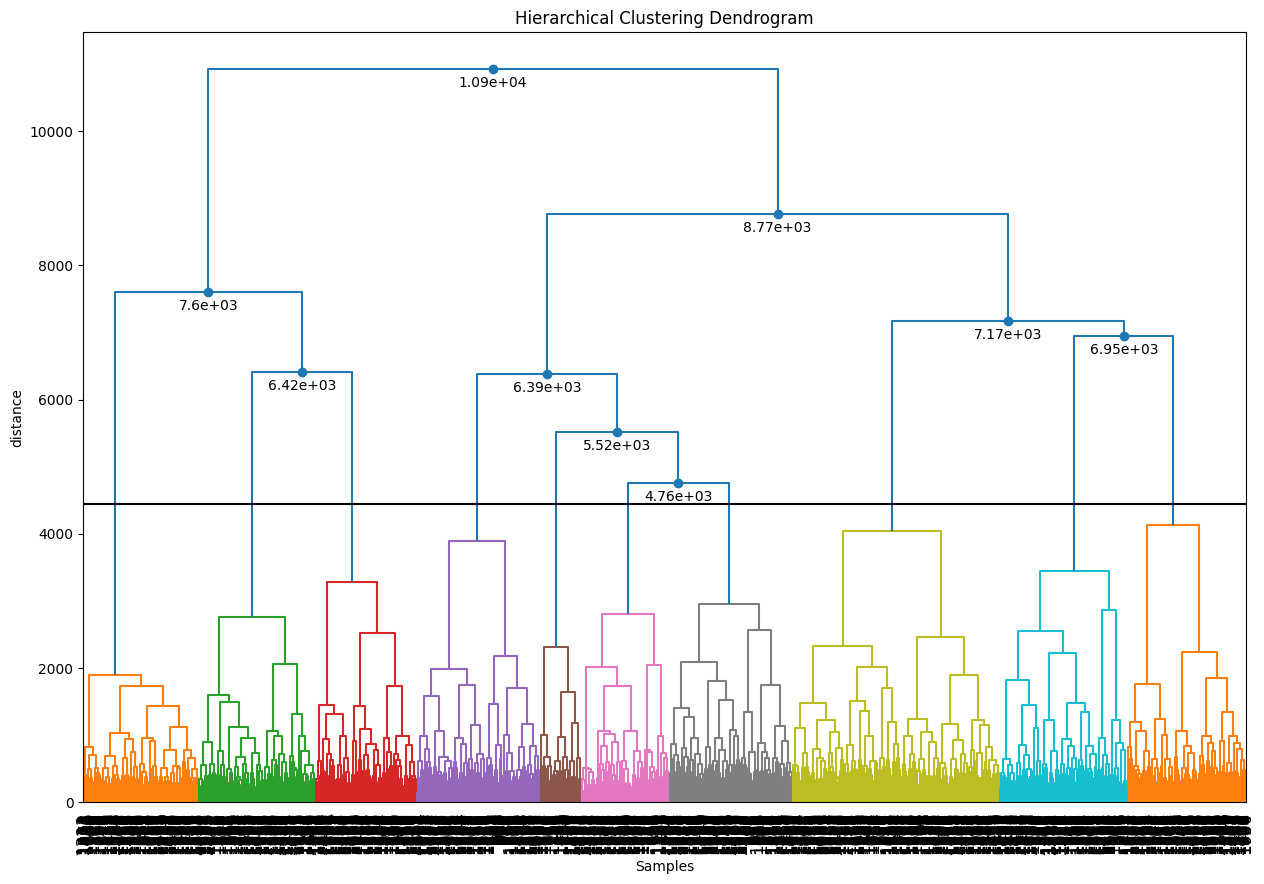
X = cl.import\_example(data='mnist')

result = cl.fit\_transform(X, cluster='agglomerative')

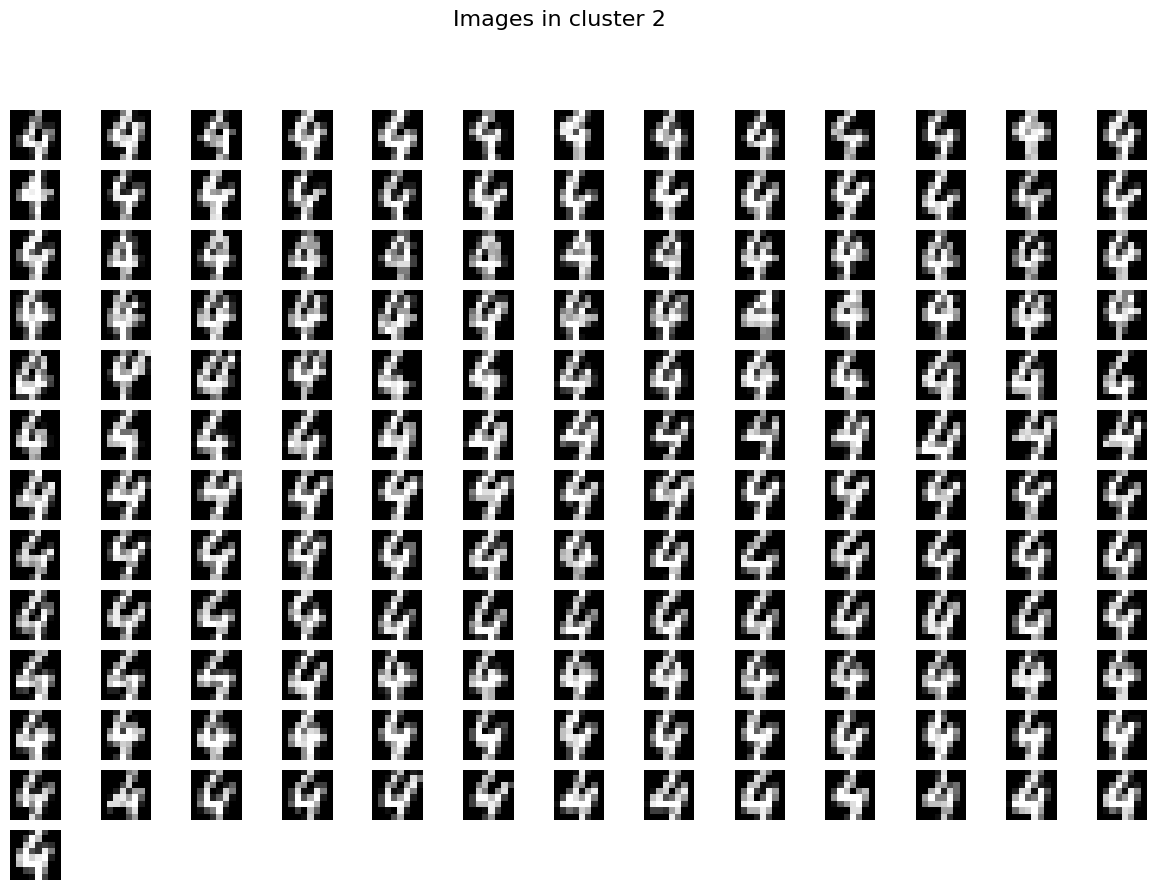
cl.dendrogram()

cl.plot(cmap='binary', labels=[1,2])

Output:







Q7 Implement Maximum-Likelihood estimation.

Code:

# %%

import numpy as np

import math

from scipy.optimize import minimize

# %%

mean = 10

std = 20

# %%

s = np.random.normal(mean, std, 3000)

# %%

def likelihood(mean, std, x):

return (1 / math.sqrt(2 \* math.pi \* std\*\*2)) \* np.exp(-(x - mean)\*\*2 / (2 \* std\*\*2))

def log\_likelihood(mean, std, data):

return sum(np.log(likelihood(mean, std, x)) for x in data)

# %%

neg\_log\_likelihood = lambda mean: -log\_likelihood(mean, std, s)

# %%

result = minimize(neg\_log\_likelihood, x0=0.0)

mean\_mle = result.x[0]

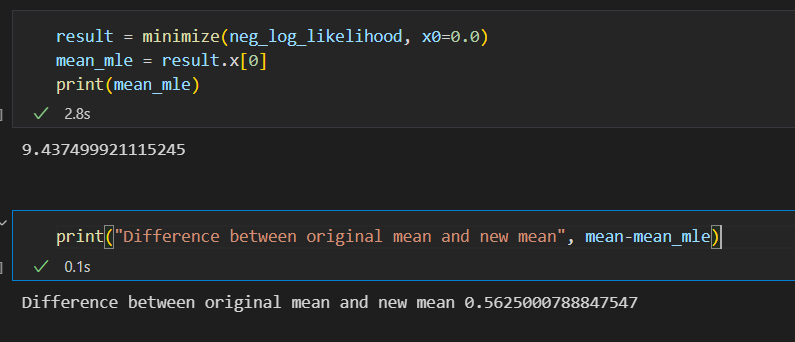
print(mean\_mle)

# %%

print("Difference between original mean and new mean", mean-mean\_mle)

# %%

Output:



Q8 Implement Principal component analysis and use it for unsupervised learning

Code:

# %%

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.cluster import AgglomerativeClustering

from sklearn.preprocessing import StandardScaler, normalize

from sklearn.metrics import silhouette\_score

import scipy.cluster.hierarchy as shc

# %%

X = pd.read\_csv('wine-clustering.csv')

X.fillna(method ='ffill', inplace = True)

# %%

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

X\_normalized = normalize(X\_scaled)

X\_normalized = pd.DataFrame(X\_normalized)

# %%

pca = PCA(n\_components = 2)

X\_principal = pca.fit\_transform(X\_normalized)

X\_principal = pd.DataFrame(X\_principal)

X\_principal.columns = ['P1', 'P2']

# %%

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

plt.title('Visualising the data')

Dendrogram = shc.dendrogram((shc.linkage(X\_principal, method ='ward')))

# %%

ac2 = AgglomerativeClustering(n\_clusters = 2)

# Visualizing the clustering

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

plt.scatter(X\_principal['P1'], X\_principal['P2'],

c = ac2.fit\_predict(X\_principal), cmap ='rainbow')

plt.show()

# %%

ac3 = AgglomerativeClustering(n\_clusters = 3)

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

plt.scatter(X\_principal['P1'], X\_principal['P2'],

c = ac3.fit\_predict(X\_principal), cmap ='rainbow')

plt.show()

# %%

ac4 = AgglomerativeClustering(n\_clusters = 4)

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

plt.scatter(X\_principal['P1'], X\_principal['P2'],

c = ac4.fit\_predict(X\_principal), cmap ='rainbow')

plt.show()

# %%

ac5 = AgglomerativeClustering(n\_clusters = 5)

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

plt.scatter(X\_principal['P1'], X\_principal['P2'],

c = ac5.fit\_predict(X\_principal), cmap ='rainbow')

plt.show()

# %%

ac6 = AgglomerativeClustering(n\_clusters = 6)

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

plt.scatter(X\_principal['P1'], X\_principal['P2'],

c = ac6.fit\_predict(X\_principal), cmap ='rainbow')

plt.show()

# %%

k = [2, 3, 4, 5, 6]

# Appending the silhouette scores of the different models to the list

silhouette\_scores = []

silhouette\_scores.append(

silhouette\_score(X\_principal, ac2.fit\_predict(X\_principal)))

silhouette\_scores.append(

silhouette\_score(X\_principal, ac3.fit\_predict(X\_principal)))

silhouette\_scores.append(

silhouette\_score(X\_principal, ac4.fit\_predict(X\_principal)))

silhouette\_scores.append(

silhouette\_score(X\_principal, ac5.fit\_predict(X\_principal)))

silhouette\_scores.append(

silhouette\_score(X\_principal, ac6.fit\_predict(X\_principal)))

# Plotting a bar graph to compare the results

plt.bar(k, silhouette\_scores)

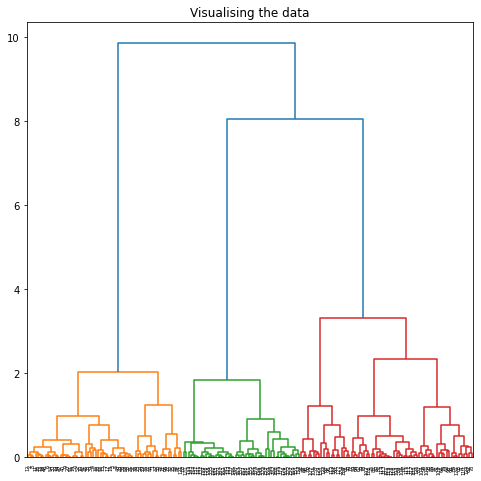
plt.xlabel('Number of clusters', fontsize = 20)

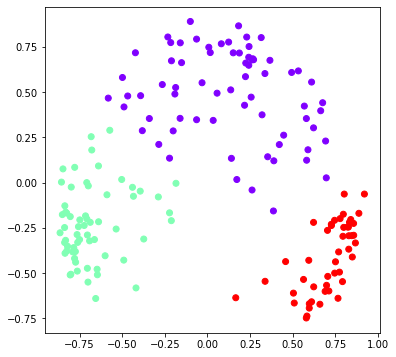
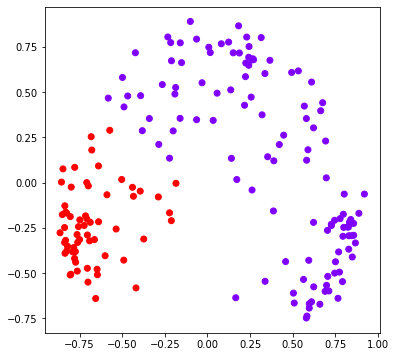
plt.ylabel('S(i)', fontsize = 20)

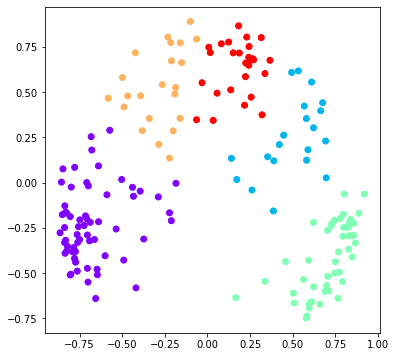
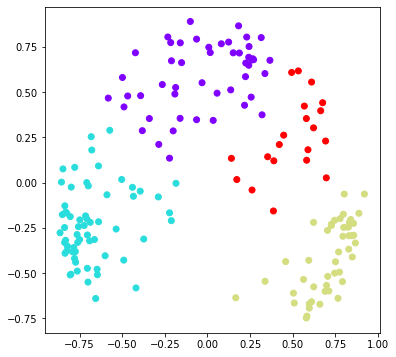
plt.show()

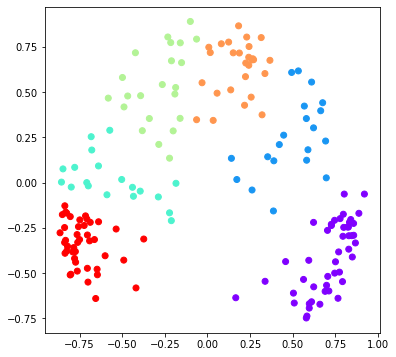
# %%

Output:



For n\_clusters = 2 For n\_clusters = 3 

For n\_clusters = 4 For n\_clusters = 5



For n\_clusters = 6

Plot of n\_clusters and score