**MACHINE LEARNING USING PYTHON LAB PRACTICAL**

# **Write a program to demonstrate Python datatypes and variables**

a=101

print(type(a))

b=0.101

print(type(b))

c=(2+3j)

print(type(c))

d="CBIT"

print(type(d))

i=True

print(type(i))

x=int(input("Enter a number: "))

y=int(input("Enter a number: "))

print("Sum of x & y is: ",(x+y))

print("Difference of x & y is: ",(x-y))

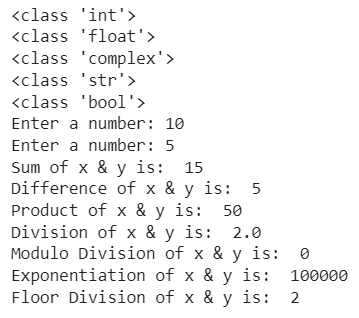
print("Product of x & y is: ",(x\*y))

print("Division of x & y is: ",(x/y))

print("Modulo Division of x & y is: ",(x%y))

print("Exponentiation of x & y is: ",(x\*\*y))

print("Floor Division of x & y is: ",(x//y))

OUTPUT  


# **Write a Python program to print the prime numbers up to ‘n’**

n=int(input("Enter n value: "))

i=2

while(i<=n):

c=0

for j in range(1,i+1):

if(i%j==0):

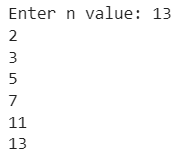
c=c+1

if(c==2):

print(i)

i=i+1

OUTPUT



# **Write a Python program to find the sum of ‘n’ natural numbers using recursion functions**

def sumuptoN(n):

if (n==0):

return 0

else:

return n+sumuptoN(n-1)

n=int(input("Enter a number: "))

if(n<0):

print("Invalid number")

else:

s=sumuptoN(n)

print("Sum of first ",n," natural numbers is : ",s)

OUTPUT



# **Write a Python program to demonstrate Strings**

a = "Welcome to the department of MCA, CBIT"

print("Length of the string: ",len(a))

print("First element of the string: ",a[0])

print("String in the given index range: ",a[15:32])

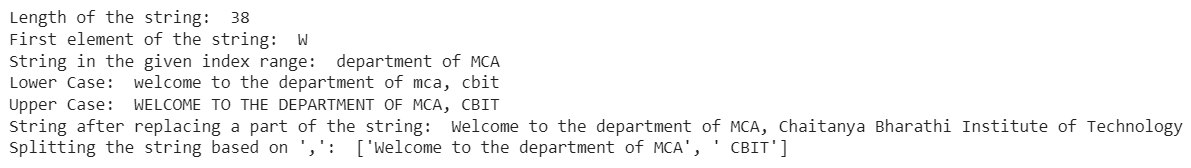
print("Lower Case: ",a.lower())

print("Upper Case: ",a.upper())

print("String after replacing a part of the string: ",a.replace("CBIT","Chaitanya Bharathi Institute of Technology"))

print("Splitting the string based on ',': ",a.split(","))

OUTPUT



# **Write a Python program to demonstrate Lists**

list=[]

print("Empty list: ",list)

list=[1,6,4,14,73,45,27,0]

print("List: ",list)

print("Length of the list: ",len(list))

list.sort()

print("List after sorting(sort operation): ",list)

print("Sum of List items is: ",sum(list))

print("Accessing each element of the list by its index: ")

for i in list:

print(i)

list.append("Python")

print("List after appending another element(append operation): ",list)

list.insert(1,"MCA")

print("List after inserting an element at a specific position(insert operation): ",list)

list.remove(4)

print("List after removing an element(remove operation): ",list)

list.pop()

print("list after poping an element(pop operation): ",list)

print("Print last element of the list: ",list[-1])

list1=list[2:5]

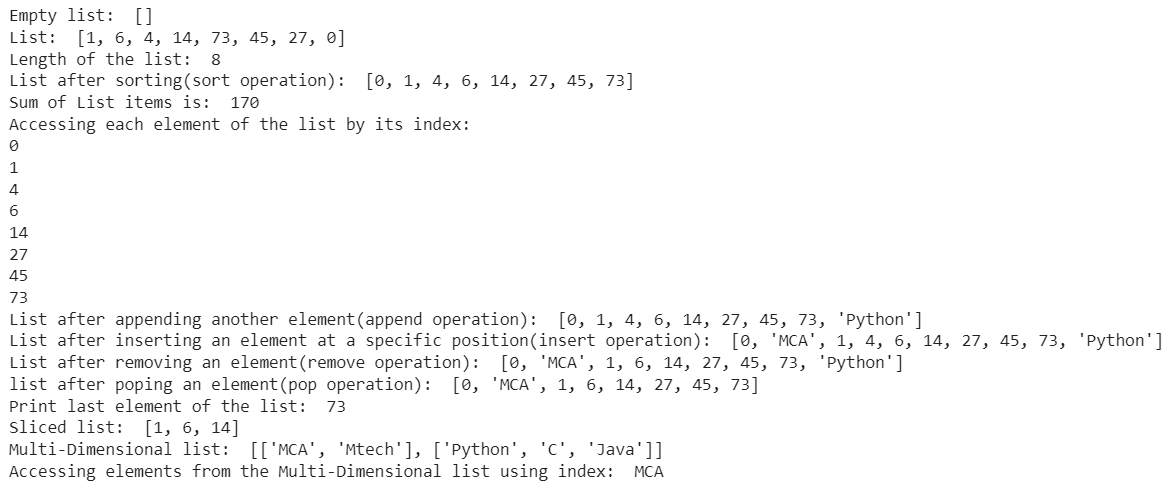
print("Sliced list: ",list1)

list2=[["MCA","Mtech"],["Python","C","Java"]]

print("Multi-Dimensional list: ",list2)

print("Accessing elements from the Multi-Dimensional list using index: ",list2[0][0])

OUTPUT



# **Write a Python program to demonstrate Tuples**

tuple=()

print("Empty tuple: ",tuple)

tuple1=(1,5,3,7)

print("tuple1: ",tuple1)

print("Length of tuple1: ",len(tuple1))

print("Maximum element in tuple1: ",max(tuple1))

print("Minimum element in tuple1: ",min(tuple1))

print("Sliced tuple: ",tuple1[2:4])

tuple2=("CBIT","MCA")

print("tuple2: ",tuple2)

print("Concatinating tuple1 and tuple2: ",tuple1+tuple2)

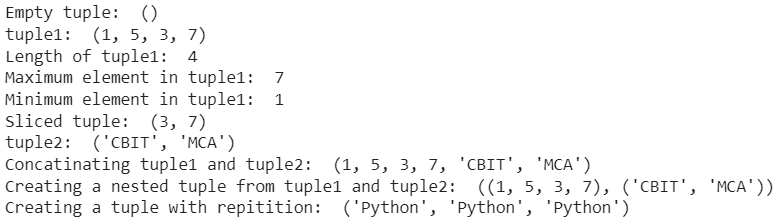
tuple3 = (tuple1, tuple2)

print("Creating a nested tuple from tuple1 and tuple2: ",tuple3)

tuple4=('Python',)\*3

print("Creating a tuple with repitition: ",tuple4)

OUTPUT



# **Write a Python program to demonstrate Dictionaries**

Dictionary={}

print("Dictionary: ",Dictionary)

Dictionary[0]='CBIT'

Dictionary[1]='MCA'

print("Dictionary after adding elements to it: ",Dictionary)

dict={1: 'Machine Learning', 2: 'Artificial Neural Network', 3: 'Cloud Computing', 4:'IOT'}

print("Dictionary dict: ",dict)

print("Acessing an element using key: ",dict[2])

print("Acessinga element using get method: ",dict.get(3))

del dict[4]

print("Dictionary after deleting a specific key(del operation): ",dict)

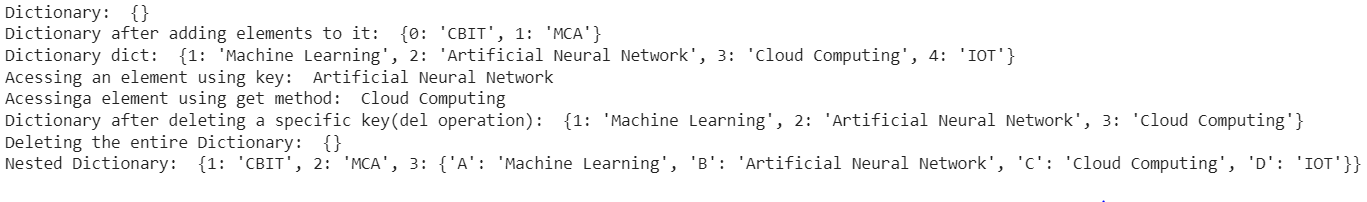
dict.clear()

print("Deleting the entire Dictionary: ",dict)

dict1={1: 'CBIT', 2: 'MCA', 3:{'A' : 'Machine Learning', 'B' : 'Artificial Neural Network', 'C' : 'Cloud Computing', 'D' : 'IOT'}}

print("Nested Dictionary: ",dict1)

OUTPUT



# **Write a Python program to demonstrate Packages and Libraries**

import statistics as st

import numpy as np

myPythonList = [1,9,8,3]

numpy\_array\_from\_list = np.array(myPythonList)

print(numpy\_array\_from\_list)

a = np.array([1,9,8,3])

print(a)

a = np.array([1,2,3])

print(a.shape)

print(a.dtype)

b = np.array([1.1,2.0,3.2])

print(b.dtype)

c = np.array([(1,2,3),

(4,5,6)])

print(c.shape)

d = np.array([

[[1, 2,3], [4, 5, 6]],

[[7, 8,9],[10, 11, 12]]

])

print(d.shape)

np.zeros((2,2), dtype=np.int16)

np.ones((1,2,3), dtype=np.int16)

e = np.array([(1,2,3), (4,5,6)])

print(e)

e.reshape(3,2)

d.flatten()

f = np.array([1,2,3])

g = np.array([4,5,6])

print('Horizontal Append:', np.hstack((f, g)))

print('Vertical Append:', np.vstack((f, g)))

print(np.arange(1, 11))

print(np.arange(1, 14, 4))

print('First row:', e[0])

print('Second row:', e[1])

print('Second column:', e[:,1])

print(e[1, 2:3])

normal\_array = np.random.normal(5, 0.5, 10)

print(normal\_array)

print(np.min(f))

print(np.max(f))

print(np.mean(f))

print(np.median(f))

print(np.std(f))

x=[0,1,2,3,4,5,6,7,8,9]

print('mean:',st.mean(x))

print('median:',st.median(x))

print('median\_low:',st.median\_low(x))

print('median\_high:',st.median\_high(x))

print('Standard Deviation:',st.stdev(x))

print('Variance',st.variance(x))

q1=np.percentile(x,np.arange(0,100,25))

q3=np.percentile(x,np.arange(0,100,75))

print(q1)

print(q3)

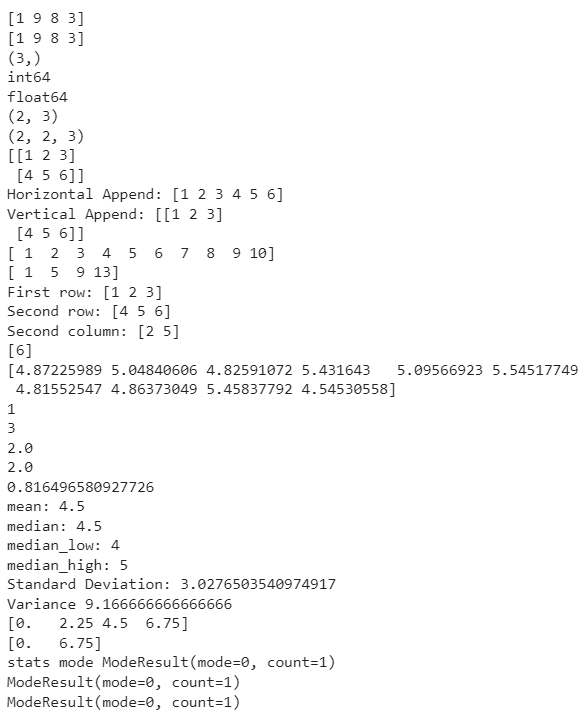
from scipy import stats

print('stats mode',stats.mode(x))

print(stats.mode(x,axis=0))

print(stats.mode(x,axis=None))

OUTPUT



# **Write a program to demonstrate Data Processing Techniques**

#Binarizer with pandas

import pandas as pd

from sklearn.preprocessing import Binarizer

dataset = pd.read\_csv('Age-salary.csv')

features = dataset.iloc[:, [2]].values # represents age column

binarizer2 = Binarizer(threshold=33)

binarizer\_scaled2 = binarizer2.fit\_transform(features)

dataset['bin\_col'] = binarizer\_scaled2

print(dataset.head())

#StandardScaler with pandas

import pandas as pd

from sklearn.preprocessing import StandardScaler

dataset = pd.read\_csv('Age-salary.csv')

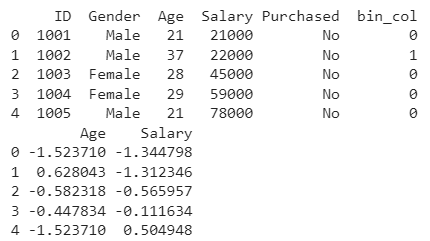
features = dataset.iloc[:, [2, 3]].values

stscaler = StandardScaler()

scaled = stscaler.fit\_transform(features)

print(pd.DataFrame(scaled, columns=['Age', 'Salary']).head())

OUTPUT



# **Write a program for simple Linear Regression**

import pandas as pd

import matplotlib.pyplot as plt

df\_train = pd.read\_csv('SalaryData\_Train.csv')

print(df\_train.head())

yoe = df\_train.iloc[:,0].values

sal = df\_train.iloc[:,1].values

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(yoe,sal,test\_size = 0.3,random\_state=0)

from sklearn.linear\_model import LinearRegression

reg = LinearRegression()

reg.fit(X\_train.reshape(-1,1),y\_train.reshape(-1,1))

plt.scatter(X\_train,y\_train,color='r')

y\_pred=reg.predict(X\_train.reshape(-1,1))

plt.plot(X\_train,reg.predict(X\_train.reshape(-1,1)),color='b')

plt.xlabel('Years of Experience')

plt.ylabel('Salary in thousands')

plt.title('Salary V/S Years of Experience')

plt.show()

print('Accuracy of Trained Data',reg.score(X\_train.reshape(-1,1),y\_train.reshape(-1,1)))

print('Accuracy of Tested Data',reg.score(X\_test.reshape(-1,1),y\_test.reshape(-1,1)))

df\_test=pd.read\_csv('SalaryData\_Test.csv')

feature\_test=df\_test.iloc[:,:].values

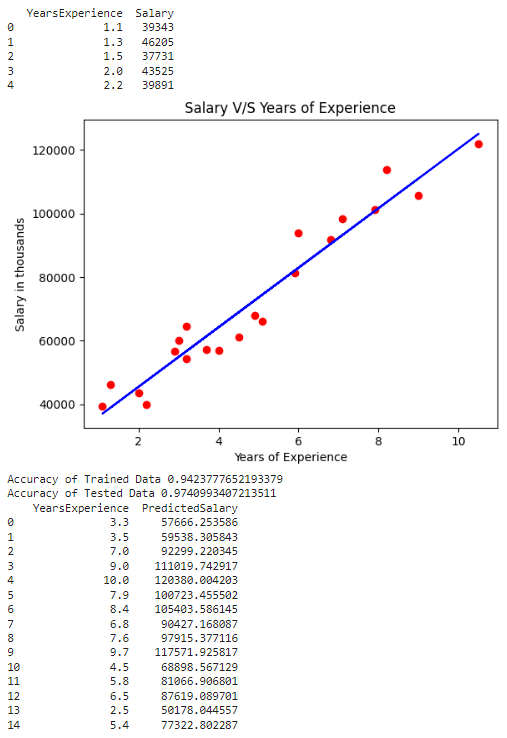
feature\_test=feature\_test.reshape(-1,1)

y\_pred\_featuretest=reg.predict(feature\_test)

df\_test['PredictedSalary']=y\_pred\_featuretest

print(df\_test)

OUTPUT



# **Write a program to demonstrate Multiple Linear Regression Backward Elimination**

import numpy as np

import pandas as pd

dataset = pd.read\_csv('50\_Startups.csv')

X = dataset.iloc[:, :-1]

y = dataset.iloc[:, 4]

states=pd.get\_dummies(X['State'],drop\_first=True)

X=X.drop('State',axis=1)

X=pd.concat([states,X],axis=1)

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

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

from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

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

y\_pred = regressor.predict(X\_test)

from sklearn.metrics import r2\_score

score=r2\_score(y\_test,y\_pred)

print('Accuracy R2 Score',score)

import statsmodels.api as sm

X = np.append(arr = np.ones((50, 1)).astype(int), values = X, axis = 1)

X\_opt = X[:, [0, 1, 2, 3, 4, 5]]

regressor\_OLS = sm.OLS( y, X\_opt).fit()

print(regressor\_OLS.summary())

X\_opt = X[:, [0, 1, 3, 4, 5]]

regressor\_OLS = sm.OLS( y, X\_opt).fit()

print(regressor\_OLS.summary())

X\_opt = X[:, [0, 3, 4, 5]]

regressor\_OLS = sm.OLS( y, X\_opt).fit()

print(regressor\_OLS.summary())

X\_opt = X[:, [0, 3, 5]]

regressor\_OLS = sm.OLS(y, X\_opt).fit()

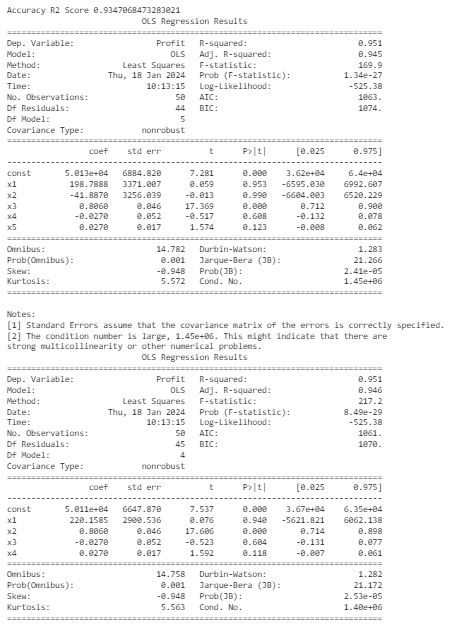
print(regressor\_OLS.summary())

X\_opt = X[:, [0, 3]]

regressor\_OLS = sm.OLS(endog = y, exog = X\_opt).fit()

print(regressor\_OLS.summary())

OUTPUT



# **Write a program to demonstrate CART (Classification and Regression Test)**

from sklearn import tree

from sklearn.datasets import load\_iris

iris = load\_iris()

clf = tree.DecisionTreeClassifier()

clf = clf.fit(iris.data, iris.target)

import graphviz

dot\_data = tree.export\_graphviz(clf, out\_file=None)

graph = graphviz.Source(dot\_data)

graph.render("iriscart")

dot\_data = tree.export\_graphviz(clf, out\_file=None,

feature\_names=iris.feature\_names,

class\_names=iris.target\_names,

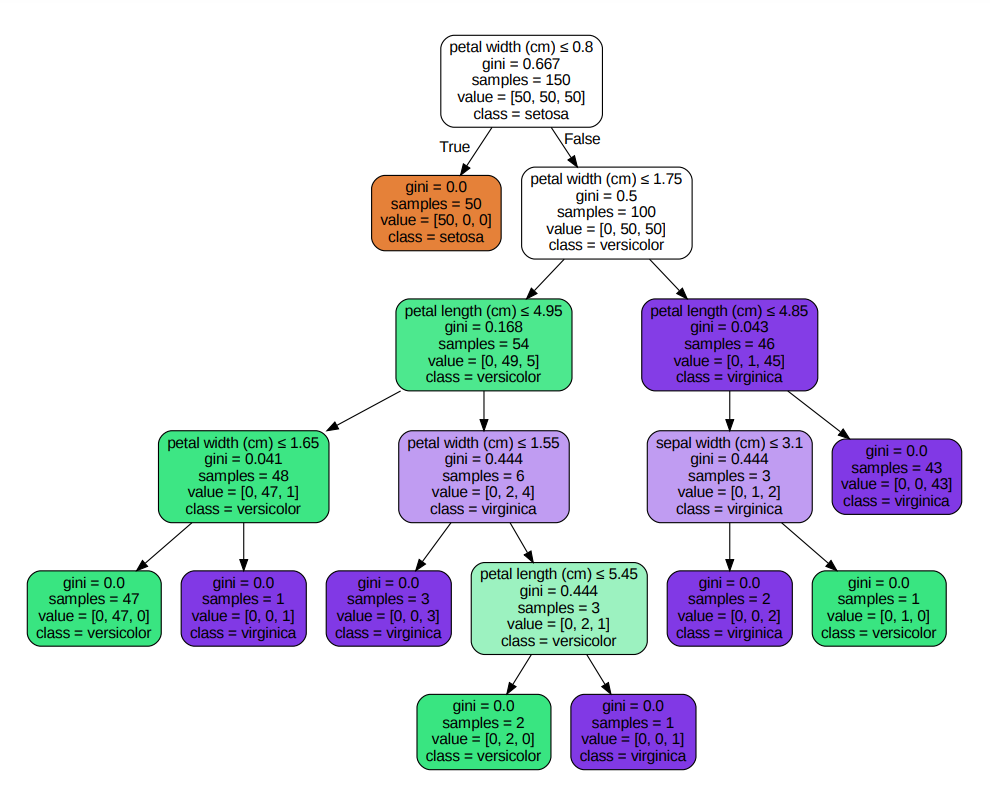
filled=True, rounded=True,

special\_characters=True)

graph = graphviz.Source(dot\_data)

graph.view()

OUTPUT



# **Write a program for Decision Tree**

from sklearn.datasets import load\_iris

from sklearn import tree

iris45 = load\_iris()

clf = tree.DecisionTreeClassifier(criterion='entropy')

clf.fit(iris45.data, iris45.target)

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris45.data, iris45.target, test\_size = 0.2, random\_state = 0)

clf.score(iris45.data, iris45.target)

predicted= clf.predict(X\_test)

import graphviz

dot\_data = tree.export\_graphviz(clf, out\_file=None, feature\_names=iris45.feature\_names, class\_names=iris45.target\_names, filled=True, rounded=True, special\_characters=True)

graph = graphviz.Source(dot\_data)

graph.view()

OUTPUT

Same as 12th Output

# **Write a program for Logistic Regression**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

X = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.20, random\_state = 0)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state = 0)

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

y\_pred = classifier.predict(X\_test)

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Logistic Regression (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Logistic Regression (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

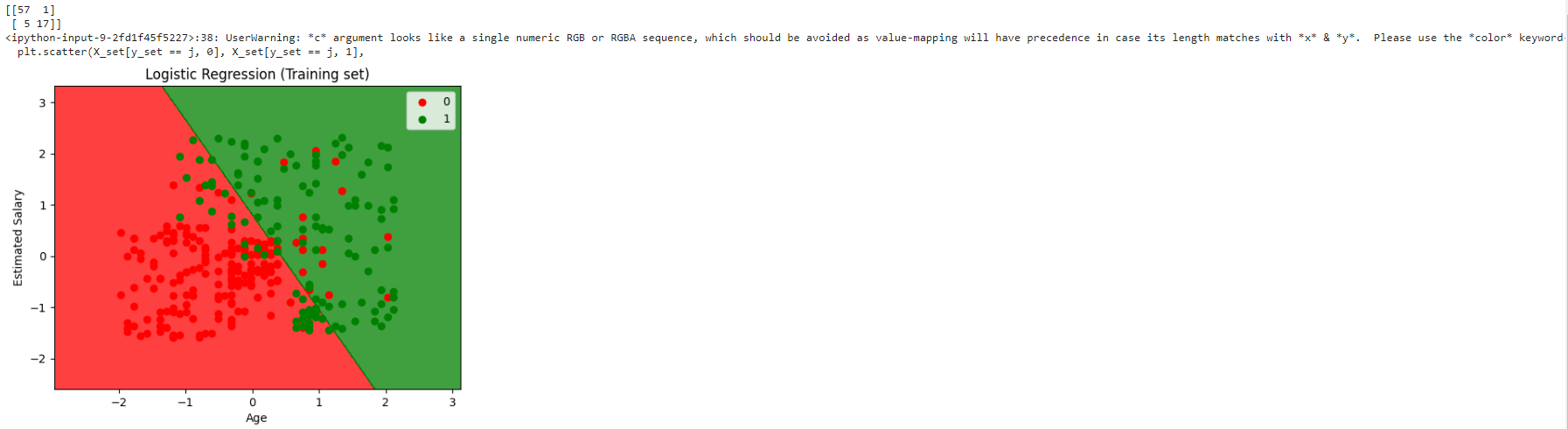
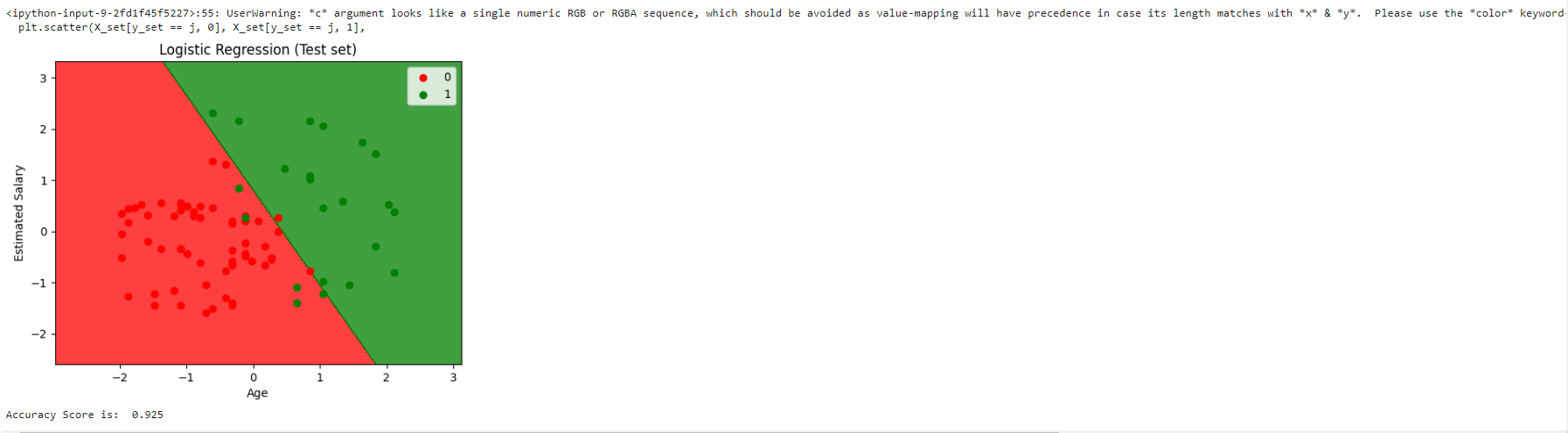
plt.legend()

plt.show()

score = classifier.score(X\_test, y\_test)

print('Accuracy Score is: ',score)

OUTPUT

# **Write a program to implement the K Nearest Neighbour Algorithm**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

X = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors = 5, metric = 'minkowski', p = 2)

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

y\_pred = classifier.predict(X\_test)

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix of KNN \n",cm)

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('K-NN (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('K-NN (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

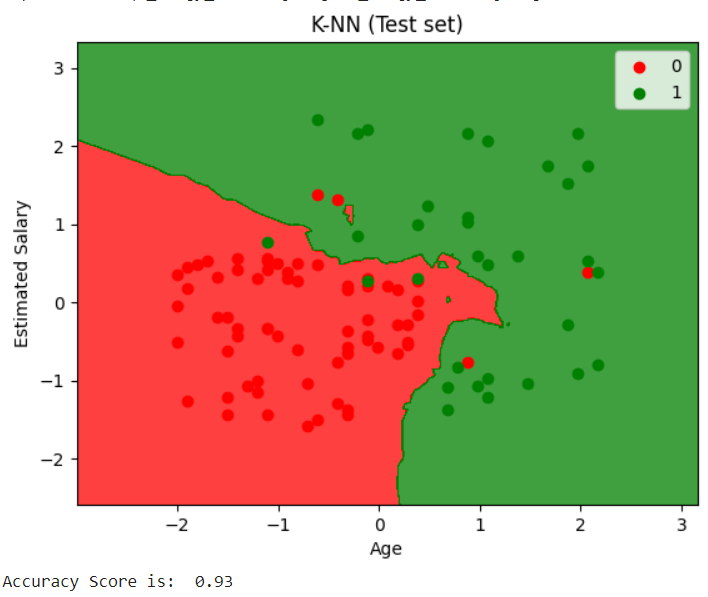
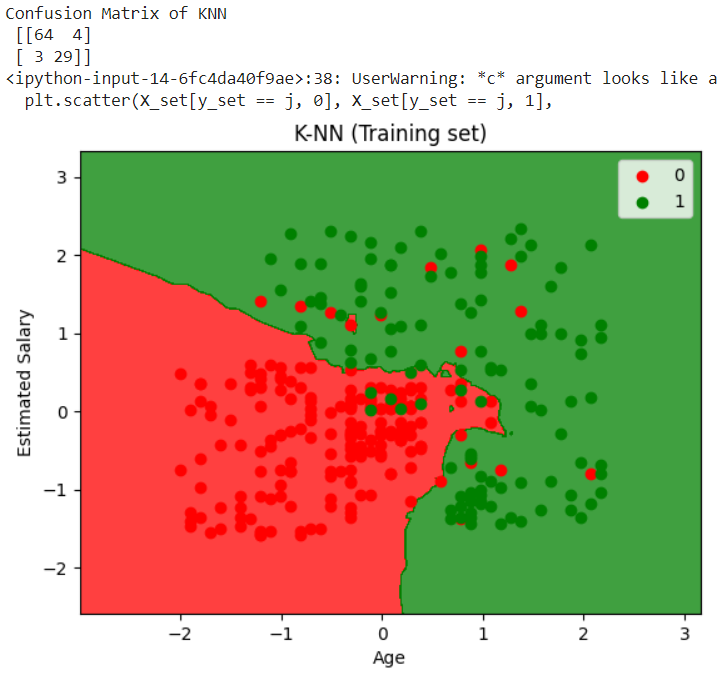
plt.legend()

plt.show()

score = classifier.score(X\_test, y\_test)

print('Accuracy Score is: ',score)

OUTPUT



# **Write a program to implement a Support Vector Machine (SVM) with different kernels**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

X = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

from sklearn.svm import SVC

classifier = SVC(kernel = 'rbf', random\_state = 0)

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

y\_pred = classifier.predict(X\_test)

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix of SVM \n",cm)

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Kernel SVM (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Kernel SVM (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

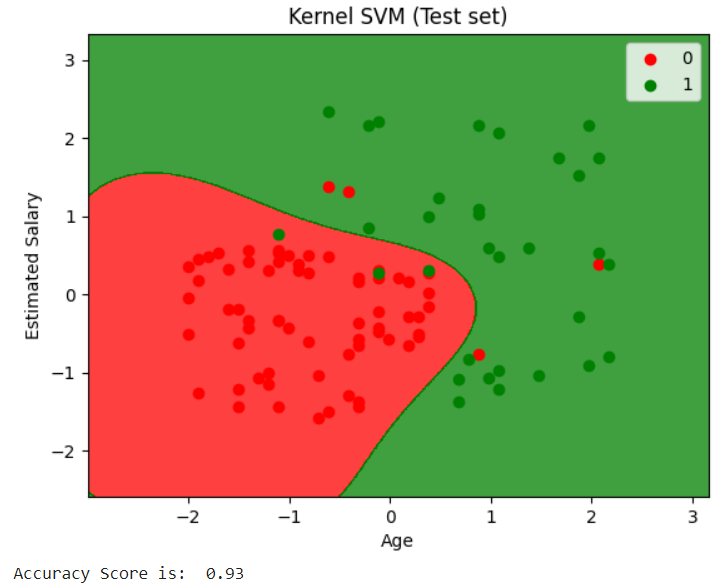
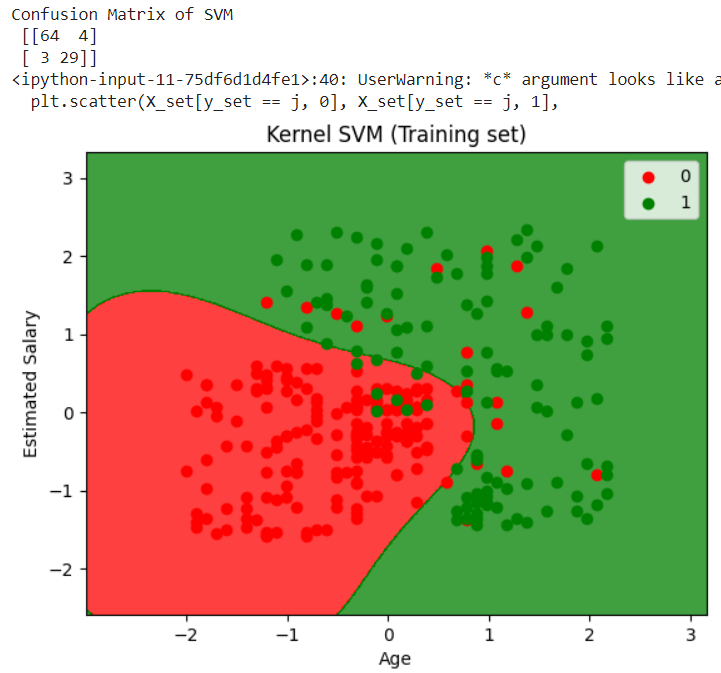
plt.legend()

plt.show()

score = classifier.score(X\_test, y\_test)

print('Accuracy Score is: ',score)

OUTPUT



# **Write a program to implement Random Forest Classification**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

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

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion\_matrix

def visualize\_results(X\_set, y\_set, title):

X1, X2 = np.meshgrid(np.arange(X\_set[:, 0].min() - 1, X\_set[:, 0].max() + 1, step=0.01),

np.arange(X\_set[:, 1].min() - 1, X\_set[:, 1].max() + 1, step=0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha=0.75, cmap=ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c=ListedColormap(('red', 'green'))(i), label=j)

plt.title(title)

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

X = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=0)

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

classifier = RandomForestClassifier(n\_estimators=10, criterion='entropy', random\_state=0)

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

y\_pred = classifier.predict(X\_test)

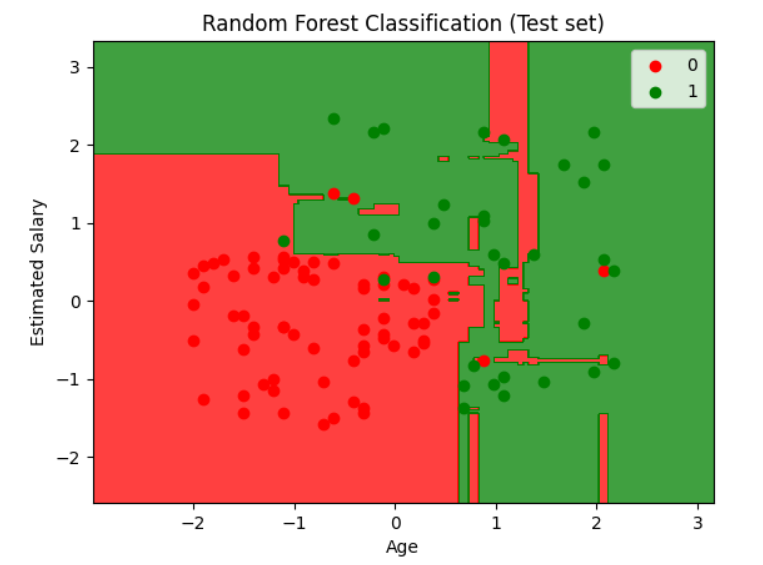
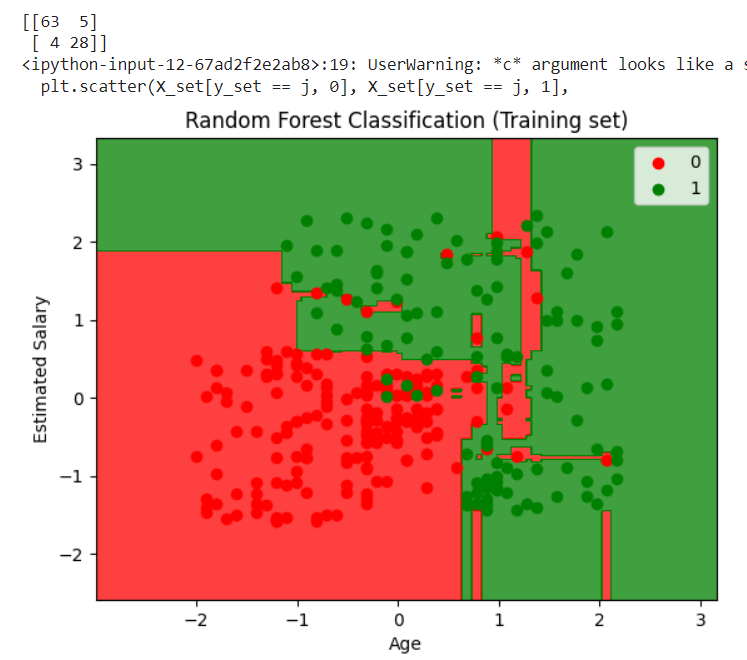
cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

visualize\_results(X\_train, y\_train, 'Random Forest Classification (Training set)')

visualize\_results(X\_test, y\_test, 'Random Forest Classification (Test set)')

OUTPUT



# **Write a program to implement the K-Means Clustering Algorithm**

from sklearn.datasets import load\_iris

from itertools import cycle

from sklearn.decomposition import PCA

from sklearn.cluster import KMeans

from numpy.random import RandomState

import pylab as pl

import matplotlib.pyplot as plt

class clustering:

def \_\_init\_\_(self):

self.plot(load\_iris().data)

def plot(self, X):

wcss=[]

for i in range(1,11):

kmeans=KMeans(n\_clusters=i,init='k-means++',max\_iter=300,n\_init=10,random\_state=0)

kmeans.fit(X)

wcss.append(kmeans.inertia\_)

plt.plot(range(1,11),wcss)

plt.title('Elbow Method')

plt.xlabel('Number of Clusters')

plt.ylabel('WCSS')

plt.show()

pca = PCA(n\_components=2, whiten=True).fit(X)

X\_pca = pca.transform(X)

kmeans = KMeans(n\_clusters=3, random\_state=RandomState(42)).fit(X\_pca)

plot\_2D(X\_pca, kmeans.labels\_, ["c0", "c1", "c2"])

def plot\_2D(data, target, target\_names):

colors = cycle('rgbcmykw')

target\_ids = range(len(target\_names))

pl.figure()

for i, c, label in zip(target\_ids, colors, target\_names):

pl.scatter(data[target == i, 0], data[target == i, 1],

c=c, label=label)

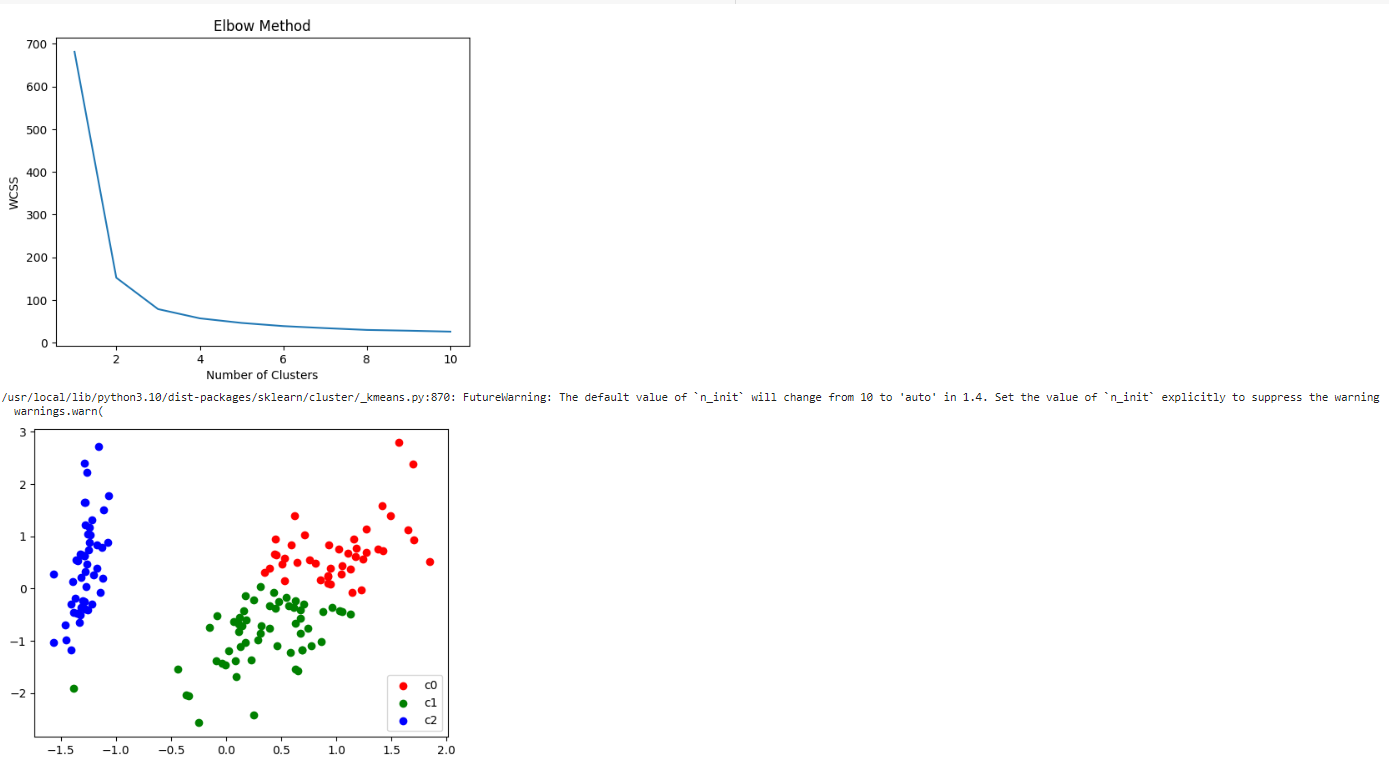
pl.legend()

pl.show()

if \_\_name\_\_ == '\_\_main\_\_':

c = clustering()

OUTPUT



# **Write a program to implement Hierarchical Clustering Algorithm**

from sklearn.datasets import load\_iris

from itertools import cycle

from sklearn.decomposition import PCA

from sklearn.cluster import KMeans

from numpy.random import RandomState

import pylab as pl

import matplotlib.pyplot as plt

class clustering:

def \_\_init\_\_(self):

self.plot(load\_iris().data)

def plot(self, X):

wcss=[]

for i in range(1,11):

kmeans=KMeans(n\_clusters=i,init='k-means++',max\_iter=300,n\_init=10,random\_state=0)

kmeans.fit(X)

wcss.append(kmeans.inertia\_)

plt.plot(range(1,11),wcss)

plt.title('Elbow Method')

plt.xlabel('Number of Clusters')

plt.ylabel('WCSS')

plt.show()

pca = PCA(n\_components=2, whiten=True).fit(X)

X\_pca = pca.transform(X)

kmeans = KMeans(n\_clusters=3, random\_state=RandomState(42)).fit(X\_pca)

plot\_2D(X\_pca, kmeans.labels\_, ["c0", "c1", "c2"])

def plot\_2D(data, target, target\_names):

colors = cycle('rgbcmykw')

target\_ids = range(len(target\_names))

pl.figure()

for i, c, label in zip(target\_ids, colors, target\_names):

pl.scatter(data[target == i, 0], data[target == i, 1],

c=c, label=label)

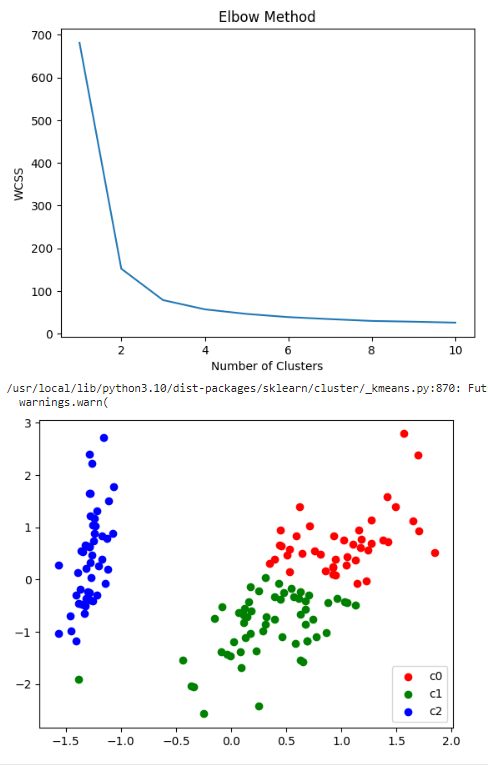
pl.legend()

pl.show()

if \_\_name\_\_ == '\_\_main\_\_':

c = clustering()

OUTPUT



# **Write a program to implement the Apriori Algorithm**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('Market\_Basket\_Optimisation.csv', low\_memory=False, header=None)

!pip install apyori

list\_of\_transactions = []

for i in range(0, 7501):

list\_of\_transactions.append([str(dataset.values[i,j]) for j in range(0, 20)])

list\_of\_transactions[0]

from apyori import apriori

rules = apriori(list\_of\_transactions, min\_support = 0.004, min\_confidence = 0.2, min\_lift = 3, min\_length = 2)

results = list(rules)

def inspect(results):

lhs = [tuple(result [2] [0] [0]) [0] for result in results]

rhs = [tuple(result [2] [0] [1]) [0] for result in results]

supports = [result [1] for result in results]

confidences = [result [2] [0] [2] for result in results]

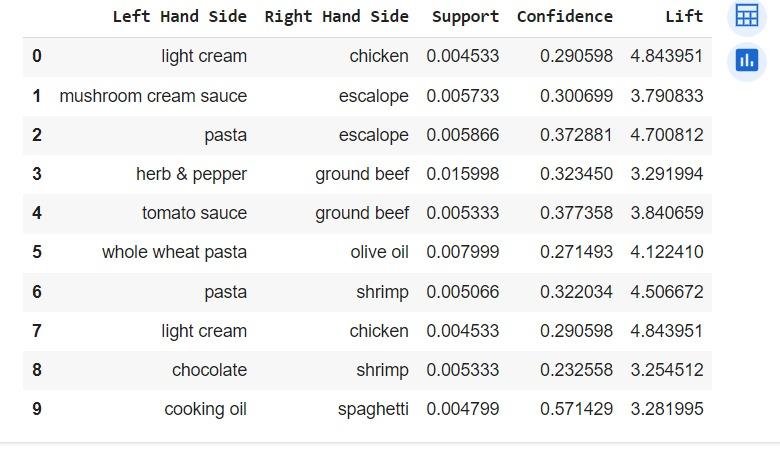
lifts = [result [2] [0] [3] for result in results]

return list(zip(lhs,rhs,supports,confidences, lifts))

resultsinDataFrame = pd.DataFrame(inspect(results),columns = ['Left Hand Side', 'Right Hand Side', 'Support', 'Confidence', 'Lift'] )

resultsinDataFrame.head(10)

OUTPUT



# **Write a program to demonstrate the logistic regression using the sigmoid function.**

import numpy as np

import pandas as pd

from sklearn import preprocessing

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

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix

import matplotlib.pyplot as plt

from matplotlib.colors import ListedColormap

try:

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

features = dataset.iloc[:, [2, 3]].values

labels = dataset.iloc[:, 4].values

x\_train, x\_test, y\_train, y\_test = train\_test\_split(features, labels, test\_size=0.25, random\_state=0)

scaler = preprocessing.StandardScaler()

x\_train = scaler.fit\_transform(x\_train)

x\_test = scaler.transform(x\_test)

classifier = LogisticRegression(random\_state=0)

classifier.fit(x\_train, y\_train)

y\_pred = classifier.predict(x\_test)

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

def visualize(x\_label, y\_label, title, features, labels):

col\_zero = features[:, 0]

col\_one = features[:, 1]

start1 = col\_zero.min() - 1

stop1 = col\_zero.max() + 1

start2 = col\_one.min() - 1

stop2 = col\_one.max() + 1

xi = np.arange(start1, stop1, 0.01)

yi = np.arange(start2, stop2, 0.01)

x, y = np.meshgrid(xi, yi)

predict\_data = np.array([x.ravel(), y.ravel()]).transpose()

y\_pred = classifier.predict(predict\_data).reshape(x.shape)

plt.contourf(x, y, y\_pred, alpha=0.75, cmap=ListedColormap(('red', 'green')))

plt.xlim(x.min(), x.max())

plt.ylim(y.min(), y.max())

unique\_labels = np.unique(labels)

for index, value in enumerate(unique\_labels):

scatter\_x = features[labels == value, 0]

scatter\_y = features[labels == value, 1]

color = ListedColormap(('orange', 'blue'))(index)

plt.scatter(scatter\_x, scatter\_y, color=color, label=value)

plt.title(title)

plt.xlabel(x\_label)

plt.ylabel(y\_label)

plt.legend()

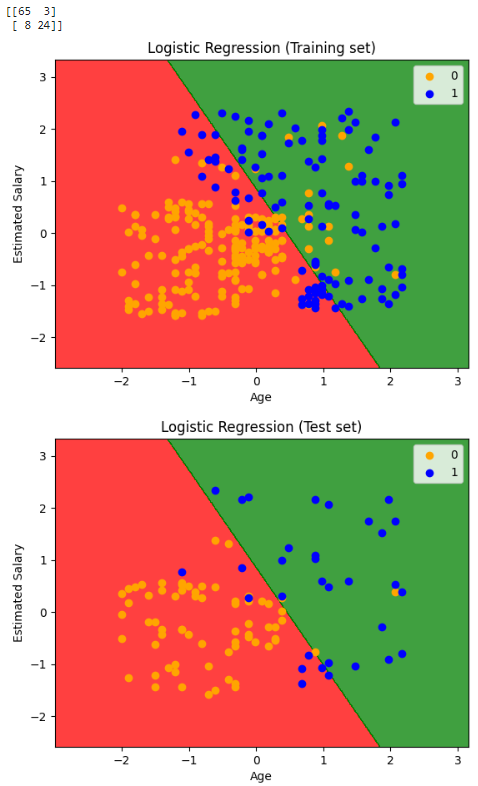
plt.show()

visualize('Age', 'Estimated Salary', 'Logistic Regression (Training set)', x\_train.copy(), y\_train.copy())

visualize('Age', 'Estimated Salary', 'Logistic Regression (Test set)', x\_test.copy(), y\_test.copy())

except Exception as e:

print(e)



# **Write a python program to find the GCD of two numbers using recursive functions.**

def gcd\_recursive(a, b):

if b == 0:

return a

else:

return gcd\_recursive(b, a % b)

num1 = int(input("Enter the first number: "))

num2 = int(input("Enter the second number: "))

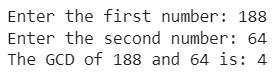
if(num1>num2):

result = gcd\_recursive(num1, num2)

else:

result = gcd\_recursive(num2, num1)

print(f"The GCD of {num1} and {num2} is: {result}")



# **Write a python program to find the gcd of a given number using functions.**

def gcd(a, b):

while b:

a, b = b, a % b

return a

num1 = int(input("Enter the first number: "))

num2 = int(input("Enter the second number: "))

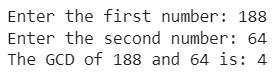
if(num1>num2):

result = gcd(num1, num2)

else:

result = gcd(num2, num1)

print(f"The GCD of {num1} and {num2} is: {result}")



# **Write a python program to find the factorial of a given number using functions**

def fact(n):

f=1

while n>0:

f=f\*n

n=n-1

return f

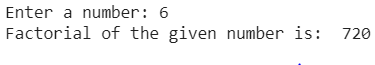
n=int(input("Enter a number: "))

if(n<0):

print("Invalid input")

else:

print("Factorial of the given number is: ",fact(n))



# **Write a python program to find the factorial of a given number using recursive Functions**

def recfact(n):

if n==0 or n==1:

return 1

else:

return n\*recfact(n-1)

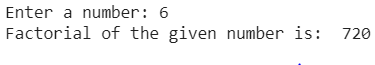
n=int(input("Enter a number: "))

if(n<0):

print("Invalid input")

else:

print("Factorial of the given number is: ",recfact(n))



# **A recursive program to find the Permutations and combinations of the given numbers.**

def factorial(n):

if n == 0 or n == 1:

return 1

else:

return n \* factorial(n - 1)

def combinations(n, r):

return factorial(n) // (factorial(r) \* factorial(n - r))

def permutations(n, r):

return factorial(n) // factorial(n - r)

def generate\_permutations\_combinations(nums, selected\_nums, index, r):

if r == 0:

print("Permutation:", selected\_nums)

return

if index == len(nums):

return

selected\_nums.append(nums[index])

generate\_permutations\_combinations(nums, selected\_nums, index + 1, r - 1)

selected\_nums.pop() # Backtrack

generate\_permutations\_combinations(nums, selected\_nums, index + 1, r)

if \_\_name\_\_ == "\_\_main\_\_":

numbers = list(map(int, input("Enter space-separated numbers: ").split()))

r = int(input("Enter the value of r: "))

print(f"Combinations of {numbers} taken {r} at a time:")

for i in range(len(numbers) + 1):

print(f" {i} elements: {combinations(len(numbers), i)} combinations")

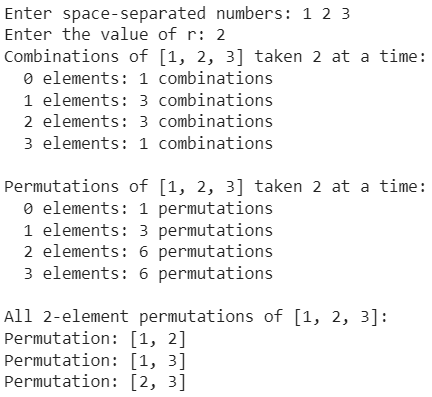
print(f"\nPermutations of {numbers} taken {r} at a time:")

for i in range(len(numbers) + 1):

print(f" {i} elements: {permutations(len(numbers), i)} permutations")

print(f"\nAll {r}-element permutations of {numbers}:")

generate\_permutations\_combinations(numbers, [], 0, r)



import pandas as pd

import numpy as np

from sklearn import datasets

import seaborn as sns

# **A Program for loading different datasets in Python**

# 1. Load CSV file using pandas

df\_csv = pd.read\_csv("data/my\_data.csv")

print(df\_csv.head()) # Preview the first few rows

# 2. Load text file using NumPy

data\_txt = np.loadtxt("data/my\_data.txt")

print(data\_txt)

# 3. Load dataset from scikit-learn

iris = datasets.load\_iris()

print(iris.data) # Features

print(iris.target) # Labels

# 4. Load dataset from Seaborn

penguins = sns.load\_dataset("penguins")

print(penguins.head())

# 5. Load Excel file using pandas

df\_excel = pd.read\_excel("data/my\_data.xlsx")

print(df\_excel.head())

# 6. Load JSON file using pandas

df\_json = pd.read\_json("data/my\_data.json")

print(df\_json.head())

# **A program on tuples and dictionaries using user defined functions**

# User-defined function to create a tuple

def create\_tuple():

elements = input("Enter elements for the tuple (comma-separated): ")

user\_tuple = tuple(map(int, elements.split(',')))

return user\_tuple

# User-defined function to manipulate the tuple

def manipulate\_tuple(user\_tuple):

print(f"Original Tuple: {user\_tuple}")

print(f"Length of the Tuple: {len(user\_tuple)}")

print(f"Sum of Tuple Elements: {sum(user\_tuple)}")

print(f"Maximum Element: {max(user\_tuple)}")

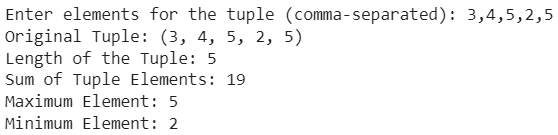
print(f"Minimum Element: {min(user\_tuple)}")

if \_\_name\_\_ == "\_\_main\_\_":

# Create a tuple and manipulate it

user\_tuple = create\_tuple()

manipulate\_tuple(user\_tuple)



# User-defined function to create a dictionary

def create\_dictionary():

key\_value\_pairs = input("Enter key-value pairs for the dictionary (key1:value1, key2:value2): ")

key\_value\_list = key\_value\_pairs.split(',')

user\_dict = dict(item.split(':') for item in key\_value\_list)

return user\_dict

# User-defined function to manipulate the dictionary

def manipulate\_dictionary(user\_dict):

print(f"Original Dictionary: {user\_dict}")

# Add a new key-value pair to the dictionary

new\_key = input("Enter a new key: ")

new\_value = input("Enter the value for the new key: ")

user\_dict[new\_key] = new\_value

print(f"Updated Dictionary: {user\_dict}")

# Remove a key-value pair from the dictionary

key\_to\_remove = input("Enter the key to remove: ")

if key\_to\_remove in user\_dict:

del user\_dict[key\_to\_remove]

print(f"Dictionary after removing key '{key\_to\_remove}': {user\_dict}")

else:

print(f"Key '{key\_to\_remove}' not found in the dictionary.")

if \_\_name\_\_ == "\_\_main\_\_":

# Create a dictionary and manipulate it

user\_dict = create\_dictionary()

manipulate\_dictionary(user\_dict)

