Q1: KMeans:

from sklearn.cluster import KMeans

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

from matplotlib import pyplot as plt

%matplotlib inline

df = pd.read\_csv("AI.csv")

print(df.head())

plt.scatter(df.Age,df['AnnualIncome'])

plt.xlabel('Age')

plt.ylabel('AnnualIncome')

plt.show()

km = KMeans(n\_clusters = 1)

y\_predicted = km.fit\_predict(df[['Age','AnnualIncome']])

print(y\_predicted)

df['cluster'] = y\_predicted

print(df.head())

km = KMeans(n\_clusters=3)

y\_predicted = km.fit\_predict(df[['Age','AnnualIncome']])

print(y\_predicted)

df['cluster'] = y\_predicted

print(df.head())

print(km.cluster\_centers\_)

df1 = df[df.cluster==0]

df2 = df[df.cluster==1]

df3 = df[df.cluster==2]

plt.scatter(df1.Age,df1['AnnualIncome'],color='green')

plt.scatter(df2.Age,df2['AnnualIncome'],color='red')

plt.scatter(df3.Age,df3['AnnualIncome'],color='black')

plt.scatter(km.cluster\_centers\_[:,0],km.cluster\_centers\_[:,1],color='purple',marker='\*',label='centroid')

plt.xlabel('Age')

plt.ylabel('AnnualIncome')

plt.legend()

plt.show()

scaler = MinMaxScaler()

df['AnnualIncome']=scaler.fit\_transform(df[['AnnualIncome']])

df['Age']=scaler.fit\_transform(df[['Age']])

print(df.head())

plt.scatter(df.Age,df['AnnualIncome'])

plt.xlabel('Age')

plt.ylabel('AnnualIncome')

plt.show()

km = KMeans(n\_clusters=1)

y\_predicted = km.fit\_predict(df[['Age','AnnualIncome']])

print(y\_predicted)

df['cluster'] = y\_predicted

print(df.head())

print(km.cluster\_centers\_)

df1 = df[df.cluster == 0]

df2 = df[df.cluster == 1]

df3 = df[df.cluster == 2]

plt.scatter(df1.Age,df1['AnnualIncome'],color='green')

plt.scatter(df2.Age,df2['AnnualIncome'],color='red')

plt.scatter(df3.Age,df3['AnnualIncome'],color='black')

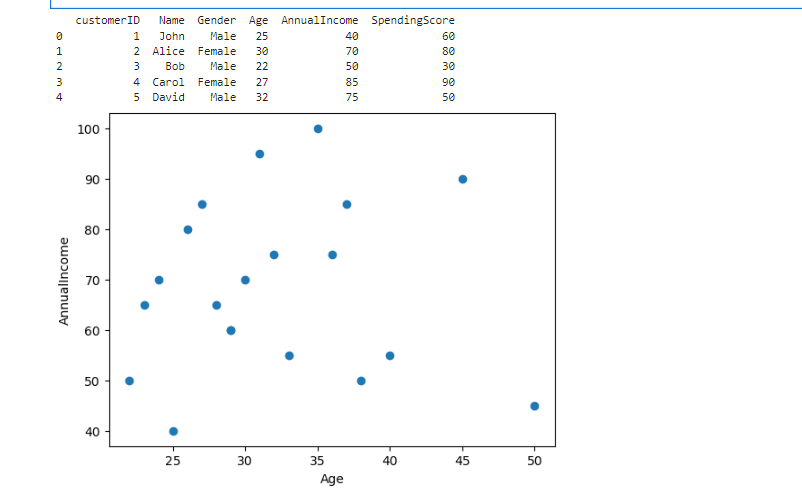
plt.scatter(km.cluster\_centers\_[:,0],km.cluster\_centers\_[:,1],color='purple',marker='\*',label='centroid')

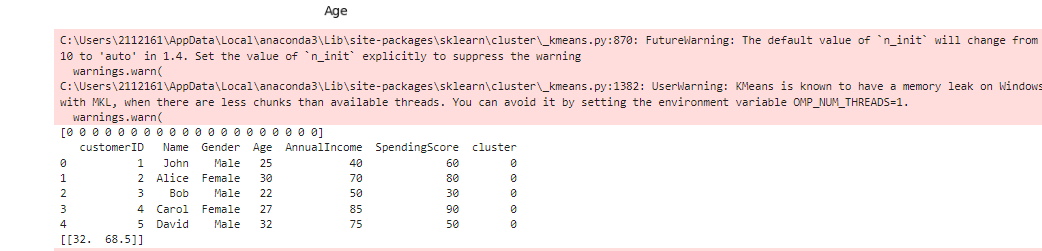
plt.xlabel('Age')

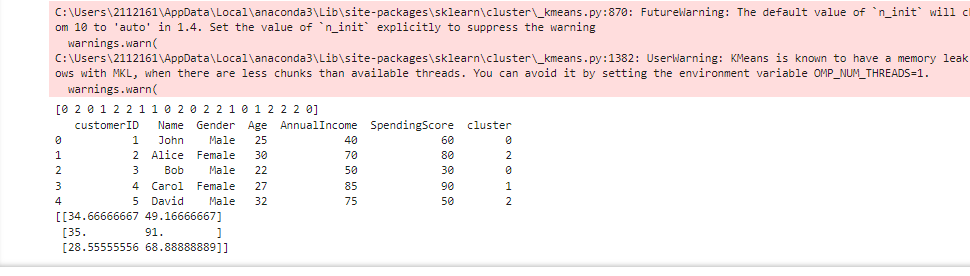
plt.ylabel('AnnualIncome')

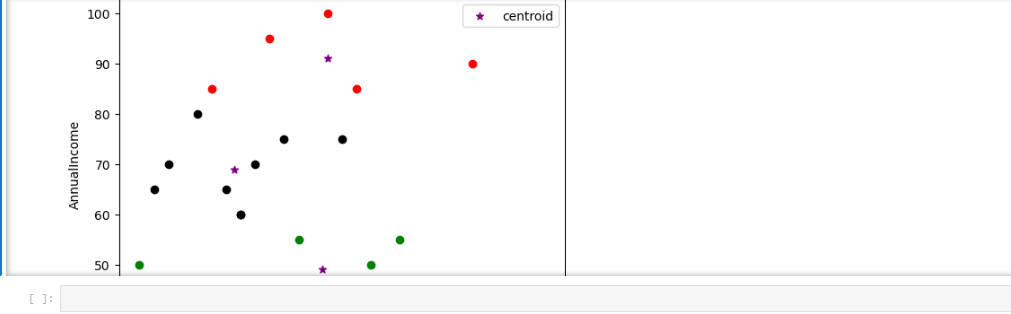
plt.legend()

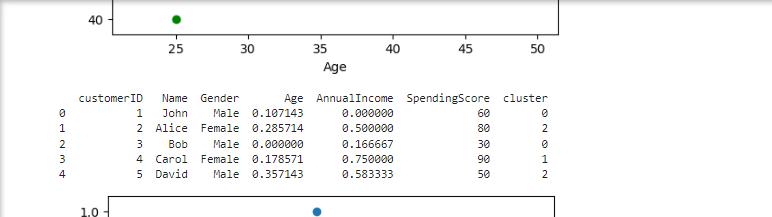
plt.show()

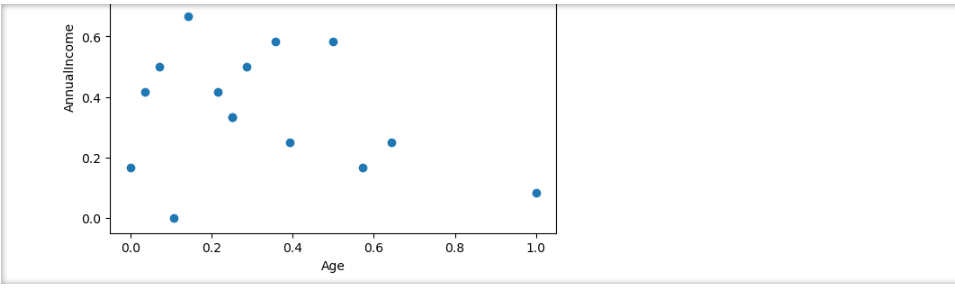


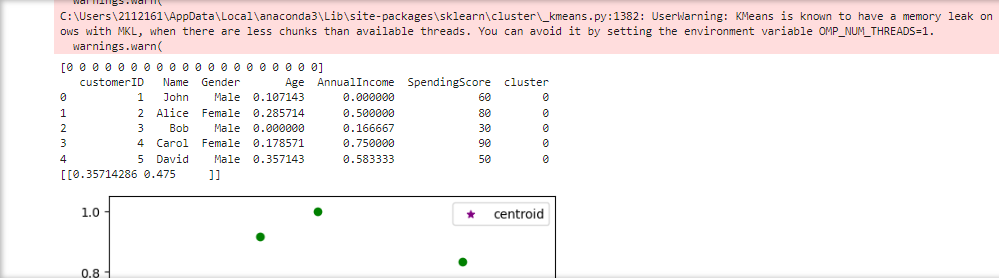


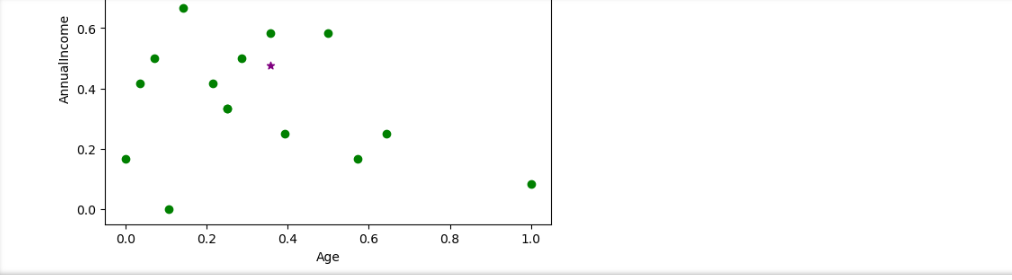












Q2: SLP

import numpy as np

class SLP(object):

def \_init\_(self, input\_size, learning\_rate = 0.1, epochs=161):

self.weights = np.zeros(input\_size+1)

print("Creating weight vector, one dimensional array with zeros ", self.weights)

self.epochs = epochs

self.learning\_rate = learning\_rate

def activation\_function(self , input\_value):

return 1 if iput\_value >= 0 else 0

def predict (self , input\_value):

z= self.weights.T.dot(input\_value)

a = self.activation\_function(z)

return a

def perceptronLearning(self, given\_input, desired\_output):

for j in range(self.epochs):

for i in range(desired\_output.shape[0]):

x=np.insert(given\_input[i], 0,1)

y=self.predict(x)

e= desired\_output[i]-y

print("Error: ", e)

print("Predicted output: ", y)

self.weights = self.weights + self.learning\_rate \* e\* x

#And Gate

given\_input = np.array([

[0,0],

[0,1],

[1,0],

[1,1]

])

desired\_output = np.array([0,0,0,1])

print("Given Input: \n", given\_input)

print("Desired output: \n", desired\_output)

slp= SLP(input\_size=2)

slp.perceptronLearning(given\_input, desired\_output)

print("Input weights with bias: \n", slp.weights)

print("Learning rate: \n", slp.learning\_rate)

print("Total epochs: \n", slp.epochs)

#OR Gate

given\_input = np.array([

[0,0],

[0,1],

[1,0],

[1,1]

])

desired\_output = np.array([0,1,1,1])

print("Given Input: \n", given\_input)

print("Desired output: \n", desired\_output)

slp= SLP(input\_size=2)

slp.perceptronLearning(given\_input, desired\_output)

print("Input weights with bias: \n", slp.weights)

print("Learning rate: \n", slp.learning\_rate)

print("Total epochs: \n", slp.epochs)

#NAND Gate

given\_input = np.array([

[0,0],

[0,1],

[1,0],

[1,1]

])

desired\_output = np.array([1,1,1,0])

print("Given Input: \n", given\_input)

print("Desired output: \n", desired\_output)

slp= SLP(input\_size=2)

slp.perceptronLearning(given\_input, desired\_output)

print("Input weights with bias: \n", slp.weights)

print("Learning rate: \n", slp.learning\_rate)

print("Total epochs: \n", slp.epochs)

#NOR

given\_input = np.array([

[0,0],

[0,1],

[1,0],

[1,1]

])

desired\_output = np.array([1,0,0,0])

print("Given Input: \n", given\_input)

print("Desired output: \n", desired\_output)

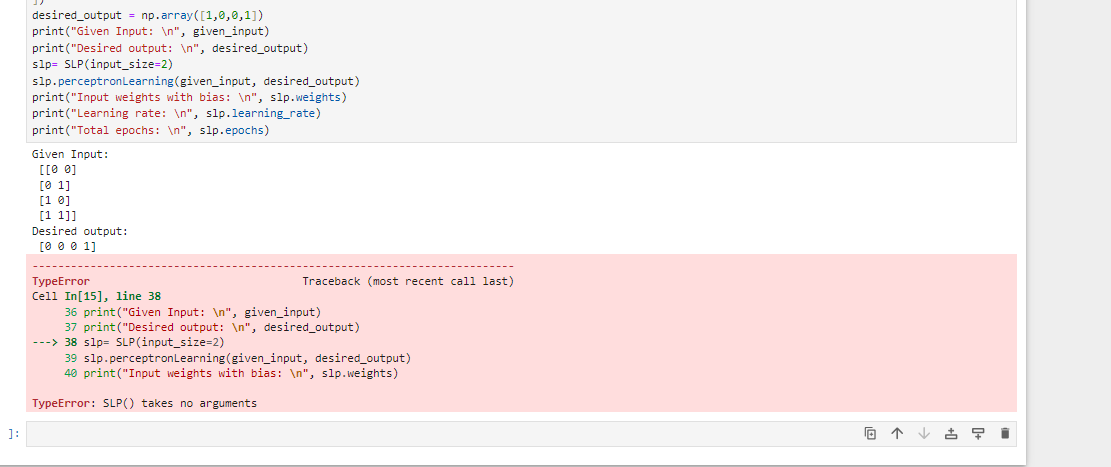
slp= SLP(input\_size=2)

slp.perceptronLearning(given\_input, desired\_output)

print("Input weights with bias: \n", slp.weights)

print("Learning rate: \n", slp.learning\_rate)

print("Total epochs: \n", slp.epochs)



Q3: Decision Tree

import pandas as pd

from sklearn.preprocessing import LabelEncoder

from sklearn import tree

df = pd.read\_csv("AI2.csv")

df.head()

inputs=df.drop(['MPG','Displacement','CylinderClass','Horsepower', 'Weight','Origin'], axis = 'columns')

target = df['MPG']

le\_acceleration = LabelEncoder()

le\_ModelYear = LabelEncoder()

inputs\_n = df.drop(['ModelYear','Acceleration'], axis = 'columns')

inputs\_n

inputs['acceleartion\_n'] = le\_acceleration.fit\_transform(inputs['Acceleartion'])

inputs['modelyear\_n'] = le\_ModelYear.fit\_transform(inputs['ModelYear'])

inputs

inputs\_n = inputs.drop(['Acceleartion', 'ModelYear'], axis = 'columns')

inputs\_n

target

model = tree.DecisionTreeClassifier()

model.fit(inputs\_n, target)

model.score(inputs\_n, target)

model.predict([[18, 307, 130, 3504, 12, 70]])

model.predict([[15, 350, 165, 3693, 11.5, 70]])

