**Week-1**

**Source code:**

from collections import defaultdict

class Graph:

    def \_\_init\_\_(self):

        self.value = defaultdict(list)

    def drawGraph(self, parent, child):

        self.value[parent].append(child)

    def DFS(self, start):

        visited = []

        stack = [start]

        print("DFS Traversal:", end=" ")

        while stack:

            s = stack.pop()

            if s not in visited:

                print(s, end=" ")

                visited.append(s)

                for neighbor in reversed(self.value[s]):

                    if neighbor not in visited:

                        stack.append(neighbor)

    def BFS(self, start):

        visited = []

        queue = [start]

        print("\nBFS Traversal:", end=" ")

        while queue:

            x = queue.pop(0)

            if x not in visited:

                print(x, end=" ")

                visited.append(x)

                for neighbor in self.value[x]:

                    if neighbor not in visited:

                        queue.append(neighbor)

# Create a graph and add edges.

g = Graph()

g.drawGraph(1, 4)

g.drawGraph(1, 2)

g.drawGraph(2, 3)

g.drawGraph(2, 6)

g.drawGraph(4, 5)

g.drawGraph(4, 7)

g.drawGraph(7, 96)

# Perform DFS and BFS traversals.

g.DFS(1)

g.BFS(1)

**output:**

DFS Traversal: 1 4 5 7 96 2 3 6

BFS Traversal: 1 4 2 5 7 3 6 96

**Week-3a:**

**Source code:**

from sys import maxsize

from itertools import permutations

V=4

def travellingSalespersonProblem(graph,s):

    vertex=[]

    for i in range(V):

        if i!=s:

            vertex.append(i)

    min\_path = maxsize

    next\_permutation=permutations(vertex)

    for i in next\_permutation:

        print(i)

        current\_pathweight=0

        k=s

        for j in i:

            current\_pathweight += graph[k][j]

            k=j

        current\_pathweight += graph[k][s]

        min\_path = min(min\_path, current\_pathweight)

    return min\_path

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

    graph = [[0,10,15,20],[10,0,35,25],[15,35,0,30],[20,25,30,0]]

    s=0

    print(travellingSalespersonProblem(graph,s))

**output:**

(1, 2, 3)

(1, 3, 2)

(2, 1, 3)

(2, 3, 1)

(3, 1, 2)

(3, 2, 1)

80

**Week-3b:**

**Source code:**

colors=['red','blue','green','yellow','black']

states=['andhra','karnataka','tamilnadu','kerela']

neighbors={}

neighbors['andhra']=['karnataka','tamilnadu']

neighbors['karnataka']=['andhra','tamilnadu','kerela']

neighbors['tamilnadu']=['andhra','kerela','karnataka']

neighbors['kerela']=['karnataka','tamilnadu']

colors\_of\_states={}

def promising(state,color):

    for neighbor in neighbors.get(state):

        colors\_of\_neighbor=colors\_of\_states.get(neighbor)

        if colors\_of\_neighbor==color:

            return False

    return True

def get\_color\_for\_state(state):

    for color in colors:

        if promising(state,color):

            return color

def main():

    for state in states:

        colors\_of\_states[state]=get\_color\_for\_state(state)

    print(colors\_of\_states)

main()

**output:**

{'andhra': 'red', 'karnataka': 'blue', 'tamilnadu': 'green', 'kerela': 'red'}

**Week-4:**

**Source code:**

from sympy import symbols, Or, Not , Implies, satisfiable

Rain = symbols('Rain')

Harry\_Visited\_Hagrid = symbols('Harry\_Visited\_Hagrid')

Harry\_Visited\_Dumbledore = symbols('Harry\_Visited\_Dumbledore')

sentence\_1 = Implies(Not(Rain), Harry\_Visited\_Hagrid)

sentence\_2 = Or (Harry\_Visited\_Hagrid,Harry\_Visited\_Dumbledore) & Not(Harry\_Visited\_Hagrid & Harry\_Visited\_Dumbledore)

sentence\_3 = Harry\_Visited\_Dumbledore

knowledge\_base = sentence\_1 & sentence\_2 & sentence\_3

solution = satisfiable(knowledge\_base, all\_models=True)

for model in solution:

   if model[Rain]:

       print("It rained today.")

   else:

         print("It did not rain today.")

**output:**

it rained today

**week-5:**

**source code:**

import numpy

from pomegranate import \*

guest = DiscreteDistribution({'A':1./3,'B':1./3,'C':1./3})

prize = DiscreteDistribution({'A':1./3,'B':1./3,'C':1./3})

monty = ConditionalProbabilityTable(

    [['A','A','A',0.0],

     ['A','A','B',0.5],

     ['A','A','C',0.5],

     ['A','B','A',0.0],

     ['A','B','B',0.0],

     ['A','B','C',1.0],

     ['A','C','A',0.0],

     ['A','C','B',1.0],

     ['A','C','C',0.0],

     ['B','A','A',0.0],

     ['B','A','B',0.0],

     ['B','A','C',1.0],

     ['B','B','A',0.5],

     ['B','B','B',0.0],

   ['B','B','C',0.5],

     ['B','C','A',1.0],

     ['B','C','B',0.0],

     ['B','C','C',0.0],

  ['C','A','A',0.0],

     ['C','A','B',1.0],

    ['C','A','C',0.0],

     ['C','B','A',1.0],

   ['C','B','B',0.0],

     ['C','B','C',0.0],

  ['C','C','A',0.5],

    ['C','C','B',0.5],

    ['C','C','C',0.0]], [guest,prize])

s1 = State(guest, name="guest")

s2 = State(prize, name="prize")

s3 = State(monty, name="monty")

model = BayesianNetwork("monty Hall Problem")

model.add\_states(s1,s2,s3)

model.add\_edge(s1,s3)

model.add\_edge(s2,s3)

model.bake()

print(model.probability([['A','B','C'],['A','A','C'],['A','C','C']]))

print(model.predict([['A',None,'C'],['A','A',None],[None,'B','A']]))

**week-6:**

**source code:**

import numpy as np

import itertools

import pandas as pd

#create state space and initial state probabilities

states=['sleeping','eating','pooping']

hidden\_states=['healthy','sick']

pi=[0.5,0.5]

state\_space=pd.Series(pi, index=hidden\_states,name='states')

print(state\_space)

a\_df=pd.DataFrame(columns=hidden\_states,index=hidden\_states)

a\_df.loc[hidden\_states[0]]=[0.7,0.3]

a\_df.loc[hidden\_states[1]]=[0.4,0.6]

print(a\_df)

observable\_states=states

b\_df=pd.DataFrame(columns=observable\_states,index=hidden\_states)

b\_df.loc[hidden\_states[0]]=[0.2,0.6,0.2]

b\_df.loc[hidden\_states[1]]=[0.4,0.1,0.5]

print(b\_df)

def HMM(obsq,a\_df,b\_df,pi,states,hidden\_states):

    hidst=list(itertools.product(hidden\_states,repeat=len(obsq)))

    print(hidst)

    sum=0

    for k in hidst:

        prod=1

        for j in range(len(k)):

            c=0

            for i in obsq:

                if c==0:

                    prod\*=a\_df[i][k[j]]\*pi[hidden\_states.index(k[j])]

                    c=1

                else:

                    prod\*=b\_df[k[j]][k[j-1]]\*a\_df[i][k[j]]

        sum+=prod

        c=0

    return sum

def vertibi(obsq,a\_df,b\_df,pi,states,hidden\_states):

    sum=0

    hidst=list(itertools.product(hidden\_states,repeat=len(obsq)))

    for k in hidst:

        sum1=0

        prod=1

        for j in range(len(k)):

            c=0

            for i in obsq:

                if c==0:

                    prod\*=a\_df[i][k[j]]\*pi[hidden\_states.index(k[j])]

                    c=1

                else:

                    prod\*=b\_df[k[j]][k[j-1]]\*a\_df[i][k[j]]

        c=0

        sum1+=prod

        if(sum1>sum):

            sum=sum1

            hs=k

    return sum,hs

obsq=['eating','sleeping','sleeping']

print(HMM(obsq,b\_df,a\_df,pi,states,hidden\_states))

print(vertibi(obsq,b\_df,a\_df,pi,states,hidden\_states))

**output:**

healthy 0.5

sick 0.5

Name: states, dtype: float64

healthy sick

healthy 0.7 0.3

sick 0.4 0.6

sleeping eating pooping

healthy 0.2 0.6 0.2

sick 0.4 0.1 0.5

[('healthy', 'healthy', 'healthy'), ('healthy', 'healthy', 'sick'), ('healthy', 'sick', 'healthy'), ('healthy', 'sick', 'sick'), ('sick', 'healthy', 'healthy'), ('sick', 'healthy', 'sick'), ('sick', 'sick', 'healthy'), ('sick', 'sick', 'sick')]

2.635148159999999e-07

(2.0329747199999986e-07, ('healthy', 'healthy', 'healthy'))

**week-7:**

**source code:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

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

from sklearn.linear\_model import LinearRegression

dataset = pd.read\_csv('Salary\_Data.csv')

dataset.head()

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

y = dataset.iloc[:, -1].values

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

regressor = LinearRegression()

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

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

y\_pred

y\_test

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

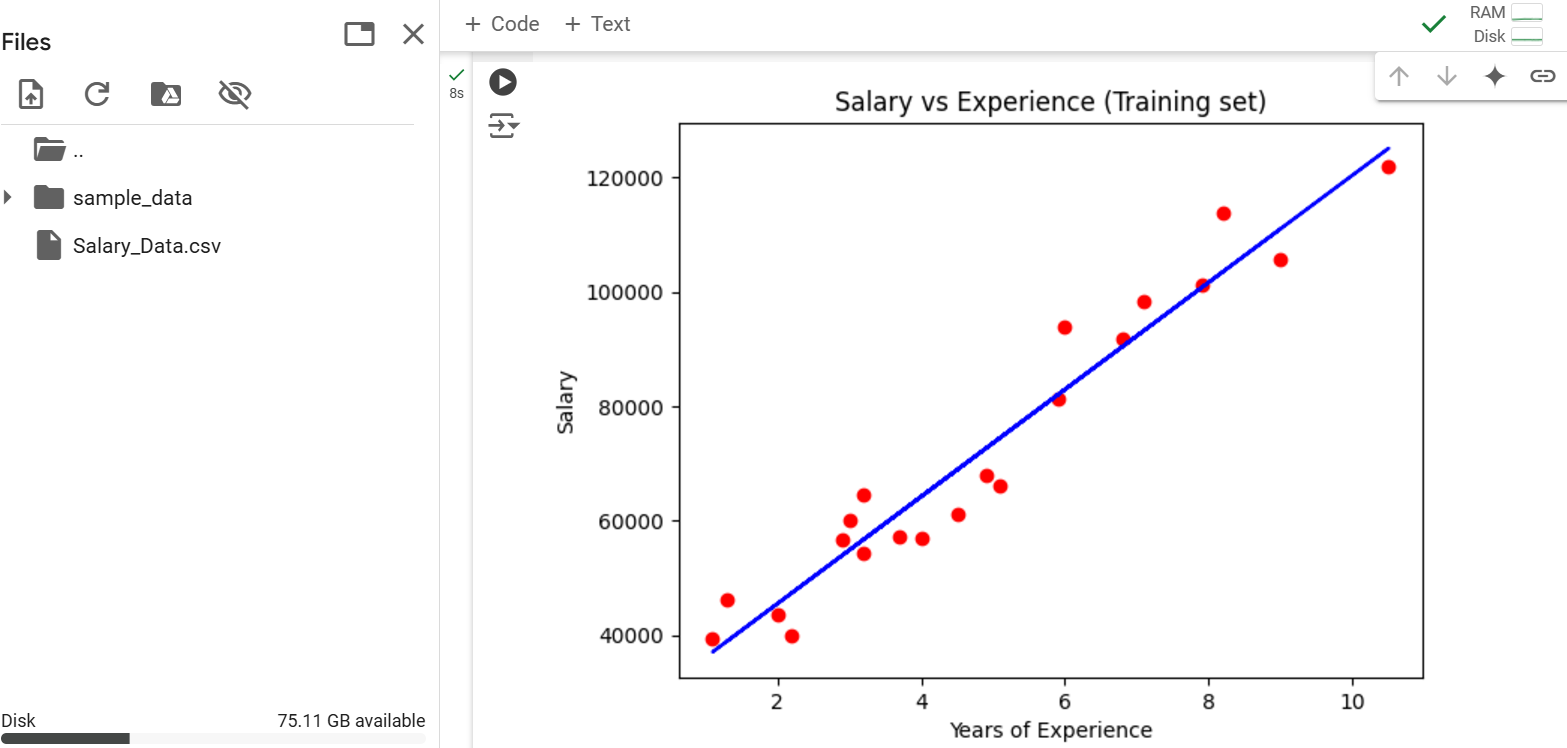
plt.plot(X\_train, regressor.predict(X\_train), color='blue')

plt.title("Salary vs Experience (Training set)")

plt.xlabel("Years of Experience")

plt.ylabel("Salary")

plt.show()



**Week-9:**

**Source code:**

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.cluster import KMeans

import pandas as pd

import numpy as np

iris = datasets.load\_iris()

x = pd.DataFrame(iris.data)

x.columns = ['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width']

y = pd.DataFrame(iris.target, columns=['target'])

plt.figure(figsize=(14, 7))

colormap = np.array(['red', 'lime', 'black'])

plt.subplot(1, 2, 1)

plt.scatter(x.sepal\_length, x.sepal\_width, c=colormap[y.target], s=40)

plt.title('Sepal')

plt.subplot(1, 2, 2)

plt.scatter(x.petal\_length, x.petal\_width, c=colormap[y.target], s=40)

plt.title('Petal')

model = KMeans(n\_clusters=3)

model.fit(x)

print(model.labels\_)

plt.subplot(1, 2, 1)

plt.scatter(x.petal\_length, x.petal\_width, c=colormap[y.target], s=40)

plt.title('Real Classification')

plt.subplot(1, 2, 2)

plt.scatter(x.petal\_length, x.petal\_width, c=model.labels\_, s=40)

plt.title('KMeans Classification')

plt.show()

**output:**

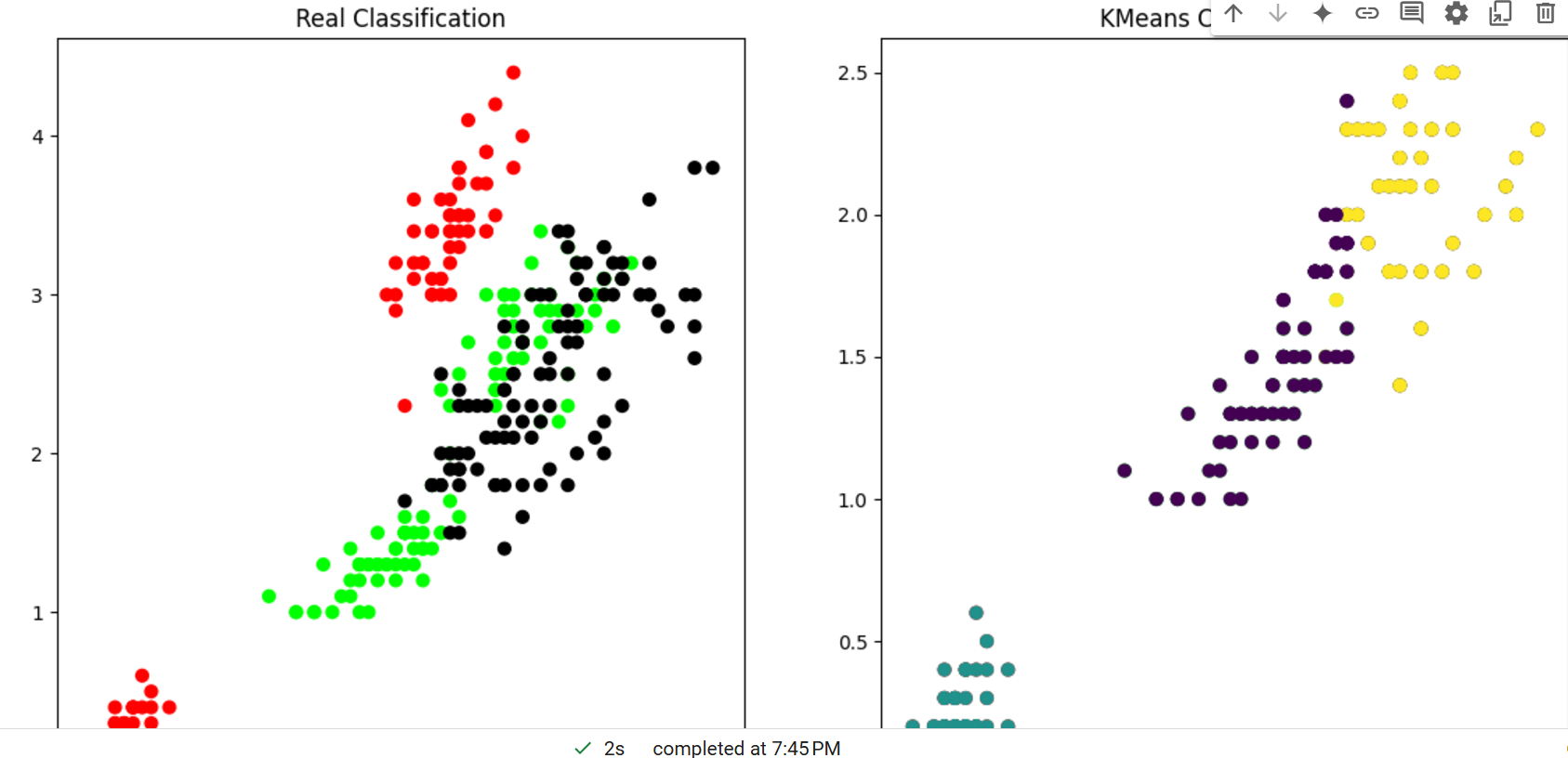
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2

2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0 2

2 0]

****

**Week-10:**

**Source code:**

import pandas as pd

from sklearn.datasets import load\_iris

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

iris = load\_iris()

print(iris.keys())

df = pd.DataFrame(iris['data'])

print(df)

print(iris['target\_names'])

print(iris['feature\_names'])

print(iris['target'])

X = df

y = iris['target']

from sklearn.neighbors import KNeighborsClassifier

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=42)

knn = KNeighborsClassifier(n\_neighbors=3)

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

from sklearn.metrics import confusion\_matrix, accuracy\_score, classification\_report

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

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

print("Correct prediction", accuracy\_score(y\_test, y\_pred))

print("Wrong prediction", (1 - accuracy\_score(y\_test, y\_pred)))

y\_test\_train = knn.predict(X\_train)

cm1 = confusion\_matrix(y\_train, y\_test\_train)

print(cm1)

**output:**

dict\_keys(['data', 'target', 'frame', 'target\_names', 'DESCR', 'feature\_names', 'filename', 'data\_module'])

0 1 2 3

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

.. ... ... ... ...

145 6.7 3.0 5.2 2.3

146 6.3 2.5 5.0 1.9

147 6.5 3.0 5.2 2.0

148 6.2 3.4 5.4 2.3

149 5.9 3.0 5.1 1.8

[150 rows x 4 columns]

['setosa' 'versicolor' 'virginica']

['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2

2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

2 2]

Correct prediction 0.98

Wrong prediction 0.020000000000000018

[[31 0 0]

[ 0 33 2]

[ 0 2 32]]

**Week-11:**

**Source code:**

import numpy as np

x = np.array([[2, 9], [1, 5], [3, 6]], dtype=float)

print(x)

y = np.array([[92], [86], [89]], dtype=float)

y = y / 100

print(y)

def sigmoid(x):

    return 1 / (1 + np.exp(-x))

def derivatives\_sigmoid(x):

    return x \* (1 - x)

epoch = 1000

lr = 0.01

input\_layer\_neurons = 2

hidden\_layer\_neurons = 2

output\_neurons = 1

wh = np.random.uniform(size=(input\_layer\_neurons, hidden\_layer\_neurons))

bh = np.random.uniform(size=(1, hidden\_layer\_neurons))

wout = np.random.uniform(size=(hidden\_layer\_neurons, output\_neurons))

bout = np.random.uniform(size=(1, output\_neurons))

for i in range(epoch):

    hinp1 = np.dot(x, wh)

    hinp = hinp1 + bh

    hlayer\_act = sigmoid(hinp)

    outinp1 = np.dot(hlayer\_act, wout)

    outinp = outinp1 + bout

    output = sigmoid(outinp)

    EO = (y - output)

    outgrad = derivatives\_sigmoid(output)

    d\_output = EO \* outgrad

    EH = d\_output.dot(wout.T)

    hiddengrad = derivatives\_sigmoid(hlayer\_act)

    d\_hiddenlayer = EH \* hiddengrad

    wout += hlayer\_act.T.dot(d\_output) \* lr

    bout += np.sum(d\_output, axis=0, keepdims=True) \* lr

    wh += x.T.dot(d\_hiddenlayer) \* lr

    bh += np.sum(d\_hiddenlayer, axis=0, keepdims=True) \* lr

    print("Actual output:" + str(y))

    print("Predicted output:" + str(output))

    print("Error:"+str(EO))

**output:**

[0.87169281]

[0.87292951]]

Error:[[ 0.04699477]

[-0.01169281]

[ 0.01707049]]

Actual output:[[0.92]

[0.86]

[0.89]]

Predicted output:[[0.87302452]

[0.87171213]

[0.87294881]]

Error:[[ 0.04697548]

[-0.01171213]

[ 0.01705119]]

Actual output:[[0.92]

[0.86]

[0.89]]

Predicted output:[[0.87304378]

[0.87173142]

[0.87296808]]

Error:[[ 0.04695622]

[-0.01173142]

[ 0.01703192]]

Actual output:[[0.92]

[0.86]

[0.89]]

Predicted output:[[0.87306302]

[0.87175068]

[0.87298732]]

Error:[[ 0.04693698]

[-0.01175068]

[ 0.01701268]]

.

.

.

.

**Week-12:**

**Source code:**

from sklearn.datasets import load\_breast\_cancer

import matplotlib.pyplot as plt

from sklearn.inspection import DecisionBoundaryDisplay

from sklearn.svm import SVC

cancer = load\_breast\_cancer()

x = cancer.data[:, :2]

y = cancer.target

svm = SVC(kernel="rbf", gamma=0.5, C=1.0)

svm.fit(x, y)

DecisionBoundaryDisplay.from\_estimator(

    svm,

    x,

    response\_method="predict",

    cmap=plt.cm.Spectral,

    alpha=0.8,

    xlabel=cancer.feature\_names[0],

    ylabel=cancer.feature\_names[1]

)

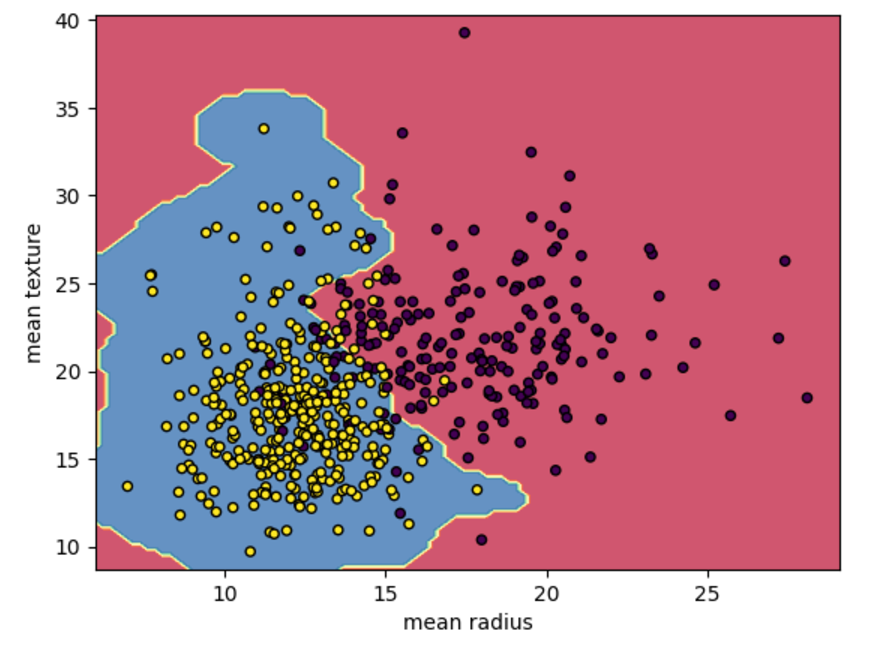
plt.scatter(x[:, 0], x[:, 1],

            c=y,

            s=20, edgecolors="k")

plt.show()

**output:**

****