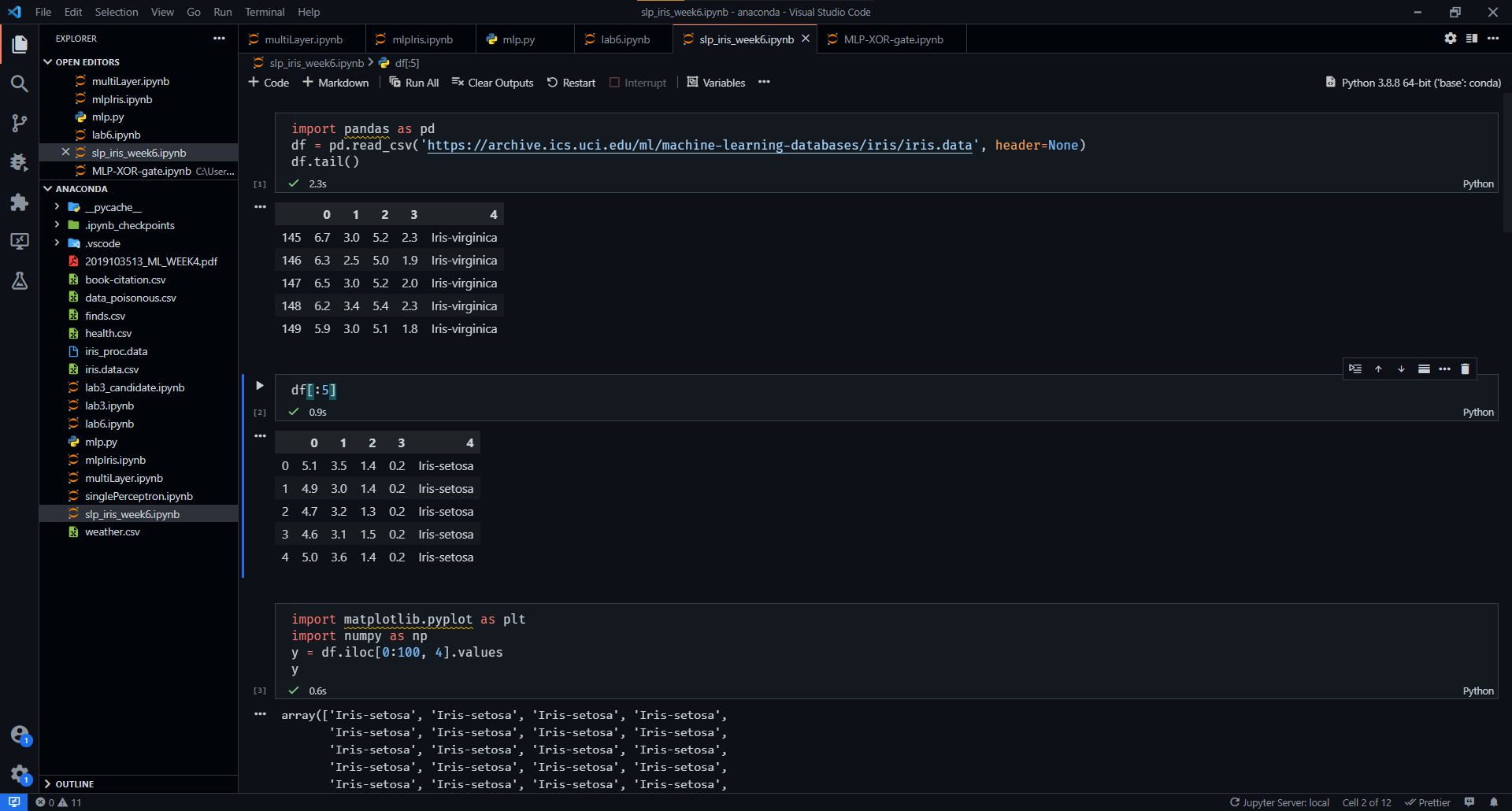
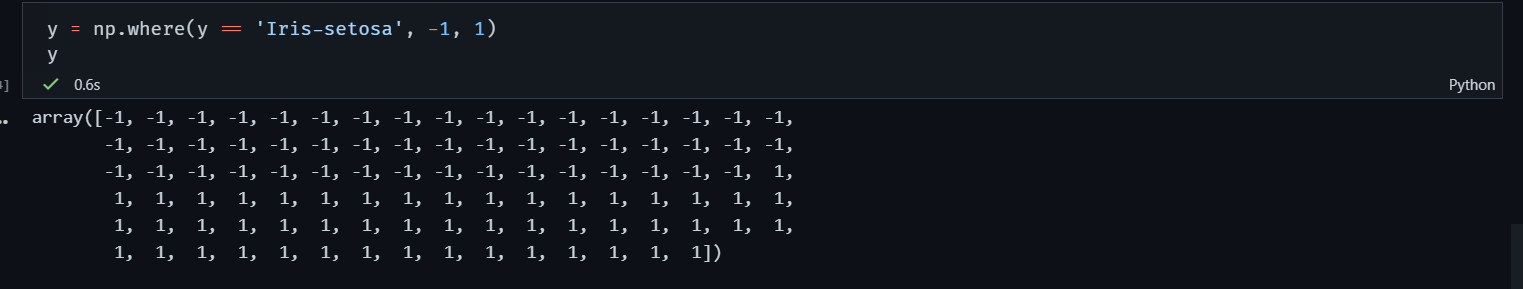
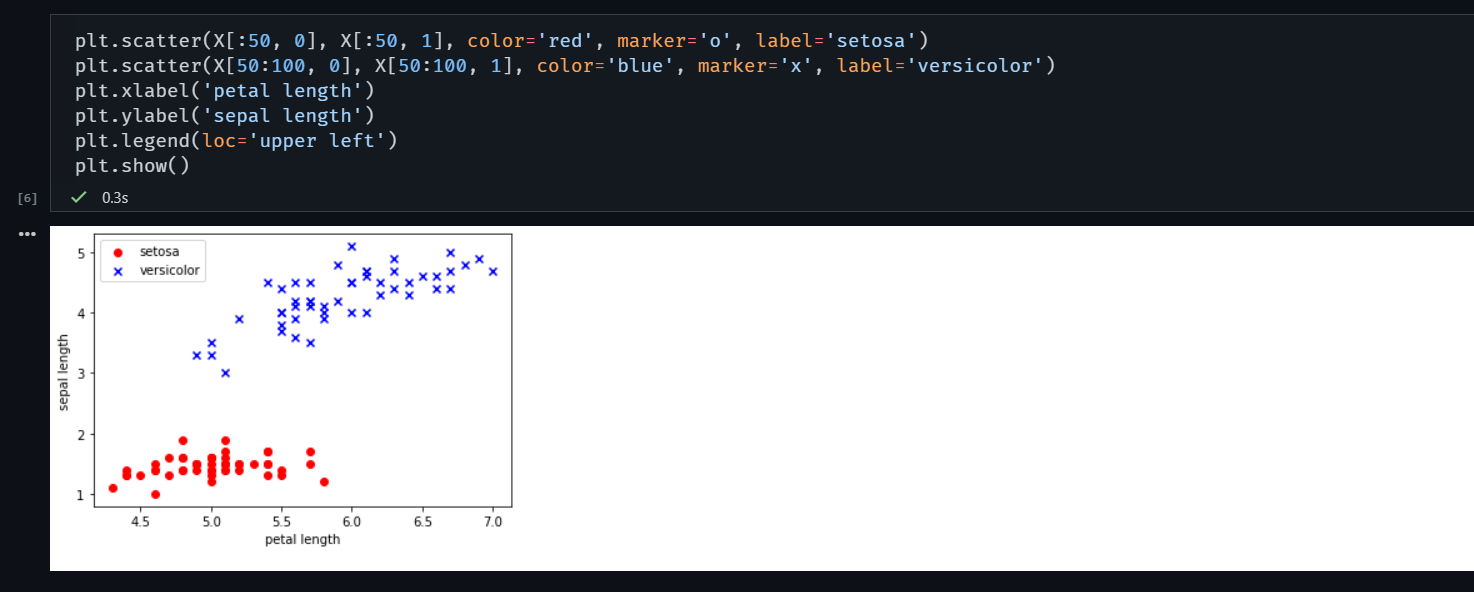
# 2019103555 PRANAVA RAMAN B M S 11/10/2021

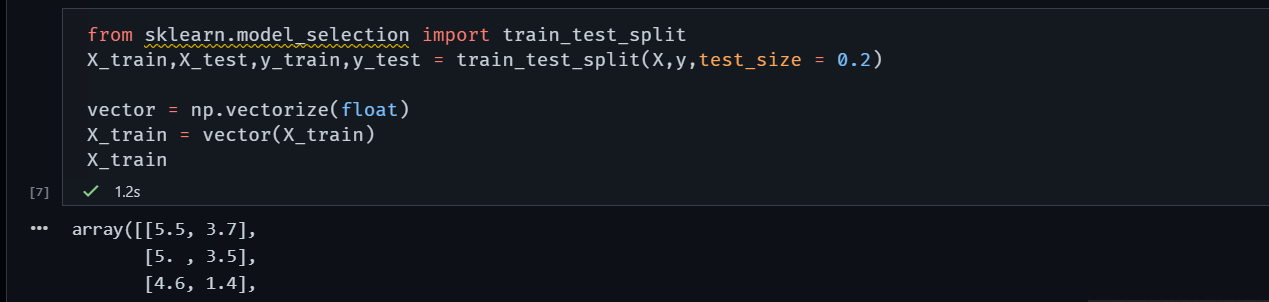
Execution:-

1. SLP for IRIS



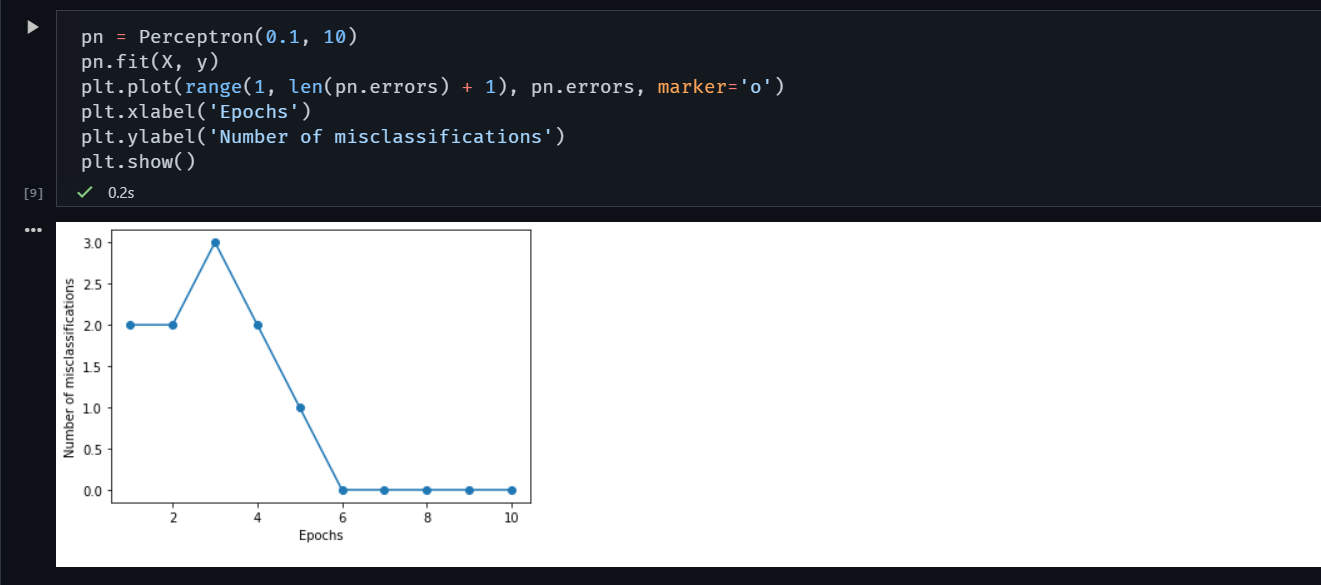






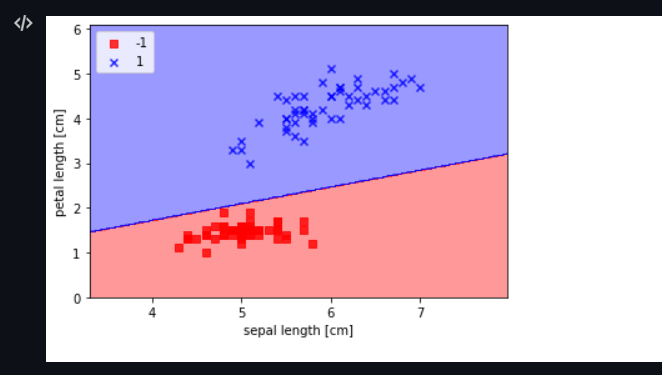
Perceptron:-







Separation of each class:-



**The core of the MLP network:-**

import numpy as np

class mlp:

    """A Multi-Layer Perceptron"""

    def \_\_init\_\_(

        self, inputs, targets, nhidden, beta=1, momentum=0.9, outtype="logistic"

    ):

        """Constructor"""

        self.nin = np.shape(inputs)[1]

        self.nout = np.shape(targets)[1]

        self.ndata = np.shape(inputs)[0]

        self.nhidden = nhidden

        self.beta = beta

        self.momentum = momentum

        self.outtype = outtype

        # Initialise network

        self.weights1 = (

            (np.random.rand(self.nin + 1, self.nhidden) - 0.5) \* 2 / np.sqrt(self.nin)

        )

        self.weights2 = (

            (np.random.rand(self.nhidden + 1, self.nout) - 0.5)

            \* 2

            / np.sqrt(self.nhidden)

        )

    def earlystopping(self, inputs, targets, valid, validtargets, eta, niterations=100):

        valid = np.concatenate((valid, -np.ones((np.shape(valid)[0], 1))), axis=1)

        old\_val\_error1 = 100002

        old\_val\_error2 = 100001

        new\_val\_error = 100000

        count = 0

        while ((old\_val\_error1 - new\_val\_error) > 0.001) or (

            (old\_val\_error2 - old\_val\_error1) > 0.001

        ):

            count += 1

            print(count)

            self.mlptrain(inputs, targets, eta, niterations)

            old\_val\_error2 = old\_val\_error1

            old\_val\_error1 = new\_val\_error

            validout = self.mlpfwd(valid)

            new\_val\_error = 0.5 \* np.sum((validtargets - validout) \*\* 2)

        print("Stopped", new\_val\_error, old\_val\_error1, old\_val\_error2)

        return new\_val\_error

    def mlptrain(self, inputs, targets, eta, niterations):

        """Train the neural network"""

        # Add the inputs that match the bias node

        inputs = np.concatenate((inputs, -np.ones((self.ndata, 1))), axis=1)

        change = range(self.ndata)

        updatew1 = np.zeros((np.shape(self.weights1)))

        updatew2 = np.zeros((np.shape(self.weights2)))

        for n in range(niterations):

            self.outputs = self.mlpfwd(inputs)

            error = 0.5 \* np.sum((self.outputs - targets) \*\* 2)

            if np.mod(n, 100) == 0:

                print("Iteration: ", n, " Error: ", error)

            # Different types of output neurons and their activation functions

            if self.outtype == "linear":

                deltao = (self.outputs - targets) / self.ndata

            elif self.outtype == "logistic":

                deltao = (

                    self.beta

                    \* (self.outputs - targets)

                    \* self.outputs

                    \* (1.0 - self.outputs)

                )

            elif self.outtype == "softmax":

                deltao = (

                    (self.outputs - targets)

                    \* (self.outputs \* (-self.outputs) + self.outputs)

                    / self.ndata

                )

            else:

                print("error")

            # hidden network delta

            deltah = (

                self.hidden

                \* self.beta

                \* (1.0 - self.hidden)

                \* (np.dot(deltao, np.transpose(self.weights2)))

            )

            updatew1 = (

                eta \* (np.dot(np.transpose(inputs), deltah[:, :-1]))

                + self.momentum \* updatew1

            )

            updatew2 = (

                eta \* (np.dot(np.transpose(self.hidden), deltao))

                + self.momentum \* updatew2

            )

            self.weights1 -= updatew1

            self.weights2 -= updatew2

    def mlpfwd(self, inputs):

        """Run the network forward"""

        self.hidden = np.dot(inputs, self.weights1)

        self.hidden = 1.0 / (1.0 + np.exp(-self.beta \* self.hidden))

        self.hidden = np.concatenate(

            (self.hidden, -np.ones((np.shape(inputs)[0], 1))), axis=1

        )

        outputs = np.dot(self.hidden, self.weights2)

        # Different types of output neurons

        if self.outtype == "linear":

            return outputs

        elif self.outtype == "logistic":

            return 1.0 / (1.0 + np.exp(-self.beta \* outputs))

        elif self.outtype == "softmax":

            normalisers = np.sum(np.exp(outputs), axis=1) \* np.ones(

                (1, np.shape(outputs)[0])

            )

            return np.transpose(np.transpose(np.exp(outputs)) / normalisers)

        else:

            print("error")

    def confmat(self, inputs, targets):

        """Confusion matrix"""

        # Add the inputs that match the bias node

        inputs = np.concatenate((inputs, -np.ones((np.shape(inputs)[0], 1))), axis=1)

        outputs = self.mlpfwd(inputs)

        nclasses = np.shape(targets)[1]

        if nclasses == 1:

            nclasses = 2

            outputs = np.where(outputs > 0.5, 1, 0)

        else:

            # 1-of-N encoding

            outputs = np.argmax(outputs, 1)

            targets = np.argmax(targets, 1)

        cm = np.zeros((nclasses, nclasses))

        for i in range(nclasses):

            for j in range(nclasses):

                cm[i, j] = np.sum(

                    np.where(outputs == i, 1, 0) \* np.where(targets == j, 1, 0)

                )

        output = cm

        print("Confusion matrix is:")

        print(cm)

        print("Percentage Correct: ", np.trace(cm) / np.sum(cm) \* 100)

        return output

**2) XOR MLP IMPLEMENTATION**

import numpy as np

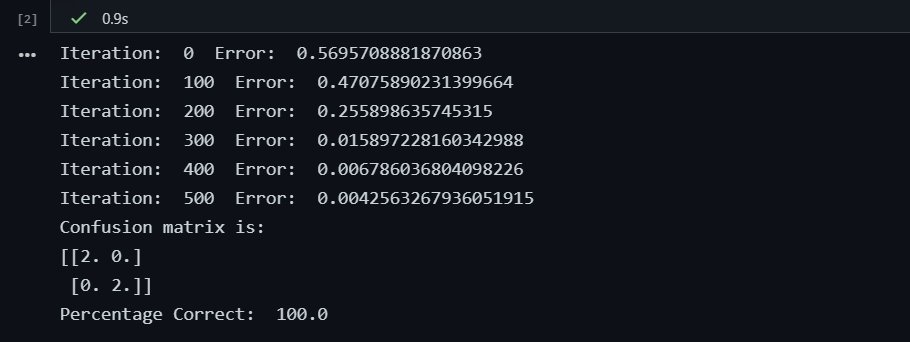
import mlp

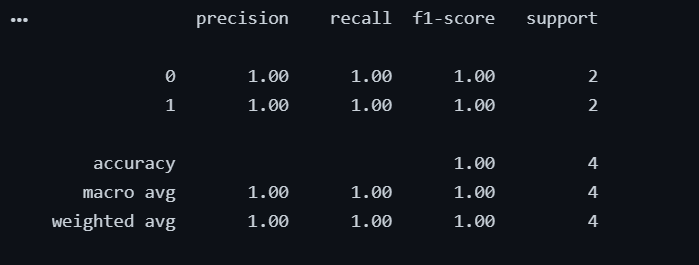
xordata = np.array([[0,0,0],[0,1,1],[1,0,1],[1,1,0]])

q = mlp.mlp(xordata[:,0:2],xordata[:,2:3],2,outtype='logistic')

q.mlptrain(xordata[:,0:2],xordata[:,2:3],0.25,501)

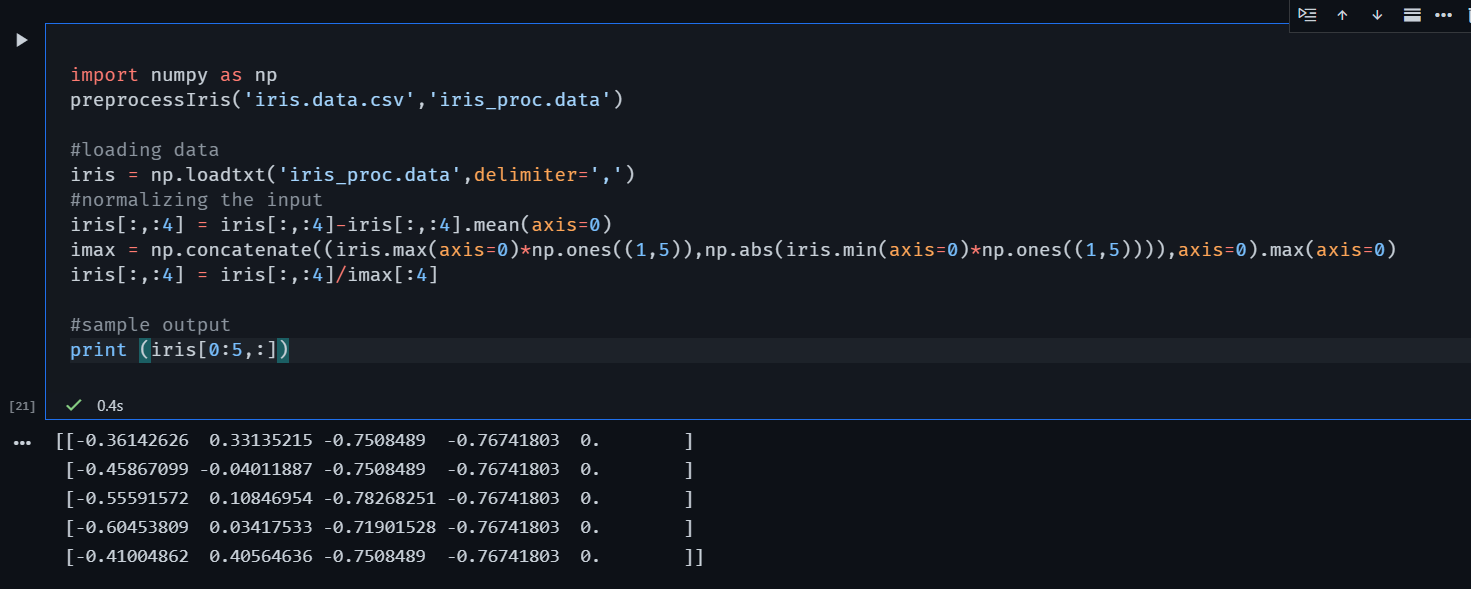
q.confmat(xordata[:,0:2],xordata[:,2:3])



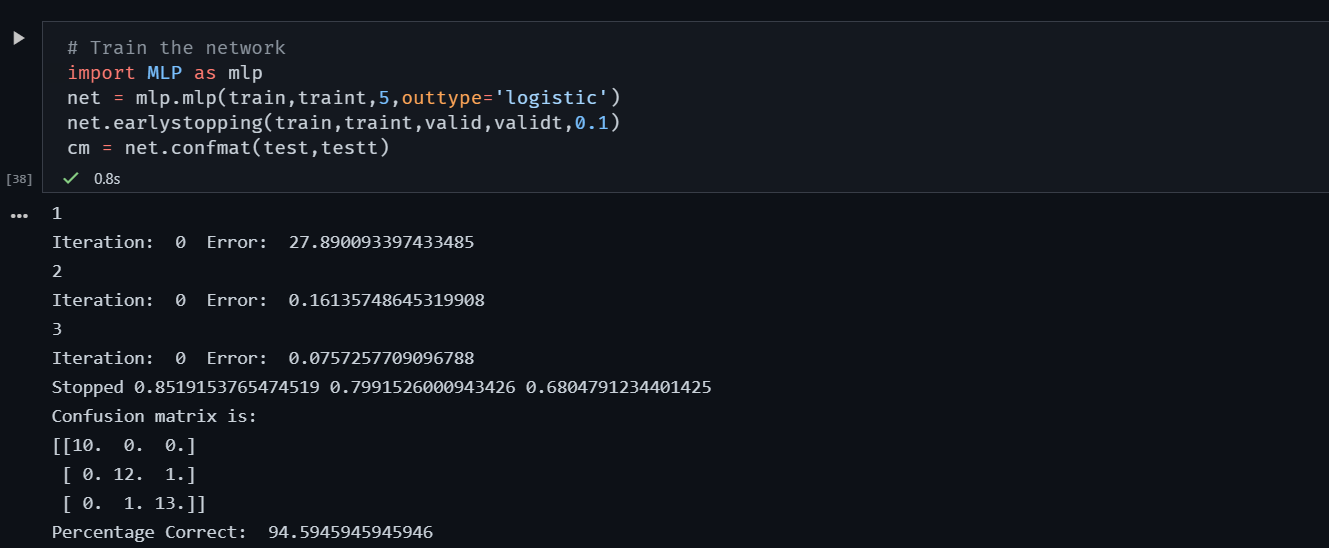


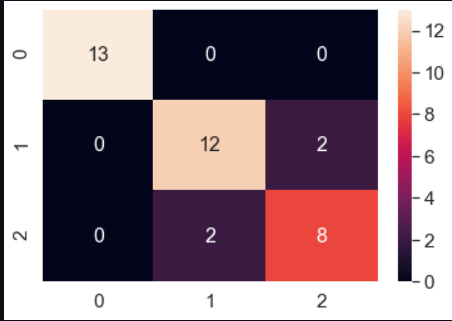
**3. IRIS DATASET:-**











**Performance metrics:-**

