**MLA0313-DEEP LEARNING FOR COMPLEX DATA HANDLING**

**NAME: KURUVA YERRISWAMY**

**REG NO: 192225106**

**EXPERIMENT:1(A)**

**AIM:** To demonstrate confusion matrix using python

**PROGRAM:**

**import numpy as np**

**from sklearn.metrics import confusion\_matrix**

**import seaborn as sns**

**import matplotlib.pyplot as plt**

**actual = np.array(**

**['Dog','Dog','Dog','Not Dog','Dog','Not Dog','Dog','Dog','Not Dog','Not Dog'])**

**predicted = np.array(**

**['Dog','Not Dog','Dog','Not Dog','Dog','Dog','Dog','Dog','Not Dog','Not Dog'])**

**cm = confusion\_matrix(actual, predicted)**

**cmap = sns.color\_palette("coolwarm", as\_cmap=True)**

**sns.heatmap(cm,**

**annot=True,**

**fmt='g',**

**cmap=cmap,**

**xticklabels=['Dog','Not Dog'],**

**yticklabels=['Dog','Not Dog'])**

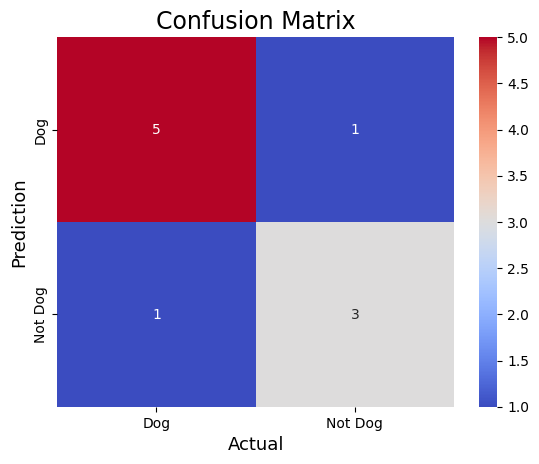
**plt.ylabel('Prediction', fontsize=13)**

**plt.xlabel('Actual', fontsize=13)**

**plt.title('Confusion Matrix', fontsize=17)**

**plt.show()**

**OUTPUT:**



**EXPERIMENT:1(B)**

**AIM:** To demonstrate 2 class confusion matrix using python

**PROGRAM:**

**from sklearn.datasets import load\_breast\_cancer**

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

**from sklearn.tree import DecisionTreeClassifier**

**from sklearn.metrics import confusion\_matrix**

**import seaborn as sns**

**import matplotlib.pyplot as plt**

**from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score**

**X, y = load\_breast\_cancer(return\_X\_y=True)**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25)**

**tree = DecisionTreeClassifier(random\_state=23)**

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

**y\_pred = tree.predict(X\_test)**

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

**sns.heatmap(cm,**

**annot=True,**

**fmt='g',**

**cmap='Blues',**

**xticklabels=['malignant', 'benign'],**

**yticklabels=['malignant', 'benign'])**

**plt.ylabel('Prediction', fontsize=13)**

**plt.xlabel('Actual', fontsize=13)**

**plt.title('Confusion Matrix', fontsize=17)**

**plt.show()**

**accuracy = accuracy\_score(y\_test, y\_pred)**

**print("Accuracy :", accuracy)**

**precision = precision\_score(y\_test, y\_pred)**

**print("Precision :", precision)**

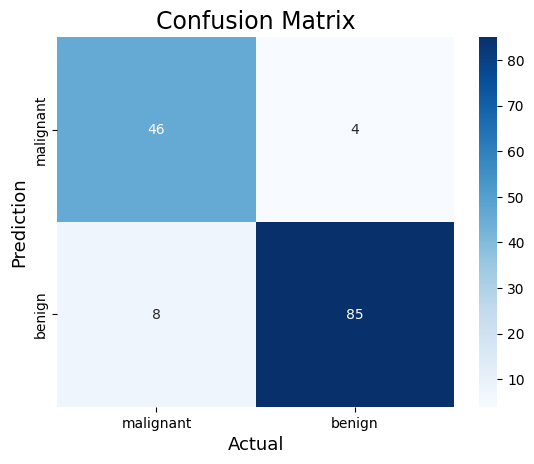
**recall = recall\_score(y\_test, y\_pred)**

**print("Recall :", recall)**

**F1\_score = f1\_score(y\_test, y\_pred)**

**print("F1-score :", F1\_score)**

**OUTPUT:**



**EXPERIMENT:2**

**AIM:** Verifying the performance of a multi class confusion matrix by using chosen database with python code

**PROGRAM:**

**from sklearn.datasets import load\_digits**

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

**from sklearn.ensemble import RandomForestClassifier**

**from sklearn.metrics import confusion\_matrix**

**import seaborn as sns**

**import matplotlib.pyplot as plt**

**from sklearn.metrics import accuracy\_score**

**X, y = load\_digits(return\_X\_y=True)**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25)**

**clf = RandomForestClassifier(random\_state=23)**

**clf.fit(X\_train, y\_train)**

**y\_pred = clf.predict(X\_test)**

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

**# Plot the confusion matrix with a custom color map and output color**

**sns.heatmap(cm,**

**annot=True,**

**fmt='g',**

**cmap='Purples', # Change color map here (e.g., 'Purples', 'Blues', 'Reds', etc.)**

**annot\_kws={"color": "red"}) # Set the output text color to red (or any color you prefer)**

**plt.ylabel('Prediction', fontsize=13)**

**plt.xlabel('Actual', fontsize=13)**

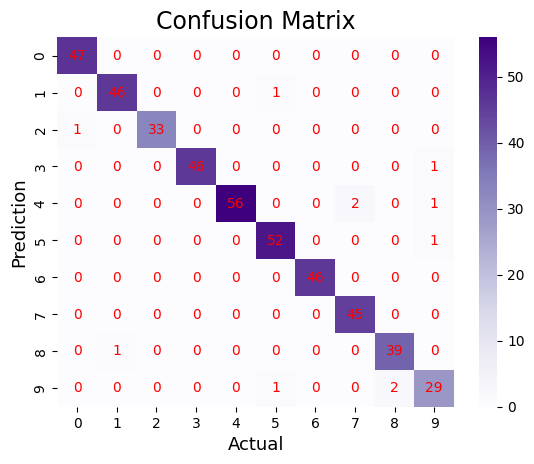
**plt.title('Confusion Matrix', fontsize=17)**

**plt.show()**

**accuracy = accuracy\_score(y\_test, y\_pred)**

**print("Accuracy :", accuracy)**

**OUTPUT:**

****

**Accuracy : 0.9755555555555555**

**EXPERIMENT:3**

**AIM:** : Verifying the performance of a overfitting by using chosen database with python code

**PROGRAM:**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import cross\_val\_score

def true\_fun(X):

return np.cos(1.5 \* np.pi \* X)

np.random.seed(0)

n\_samples = 30

degrees = [1, 4, 15]

X = np.sort(np.random.rand(n\_samples))

y = true\_fun(X) + np.random.randn(n\_samples) \* 0.1

plt.figure(figsize=(14, 5))

for i in range(len(degrees)):

ax = plt.subplot(1, len(degrees), i + 1)

plt.setp(ax, xticks=(), yticks=())

polynomial\_features = PolynomialFeatures(degree=degrees[i], include\_bias=False)

linear\_regression = LinearRegression()

pipeline = Pipeline(

[

("polynomial\_features", polynomial\_features),

("linear\_regression", linear\_regression),

]

)

pipeline.fit(X[:, np.newaxis], y)

scores = cross\_val\_score(

pipeline, X[:, np.newaxis], y, scoring="neg\_mean\_squared\_error", cv=10

)

X\_test = np.linspace(0, 1, 100)

plt.plot(X\_test, pipeline.predict(X\_test[:, np.newaxis]), label="Model", color='red') # Red for the model

plt.plot(X\_test, true\_fun(X\_test), label="True function", color='green') # Green for true function

plt.scatter(X, y, edgecolor="blue", facecolor="yellow", s=40, label="Samples") # Blue edge, yellow face for the samples

plt.xlabel("x")

plt.ylabel("y")

plt.xlim((0, 1))

plt.ylim((-2, 2))

plt.legend(loc="best")

plt.title(

"Degree {}\nMSE = {:.2e}(+/- {:.2e})".format(

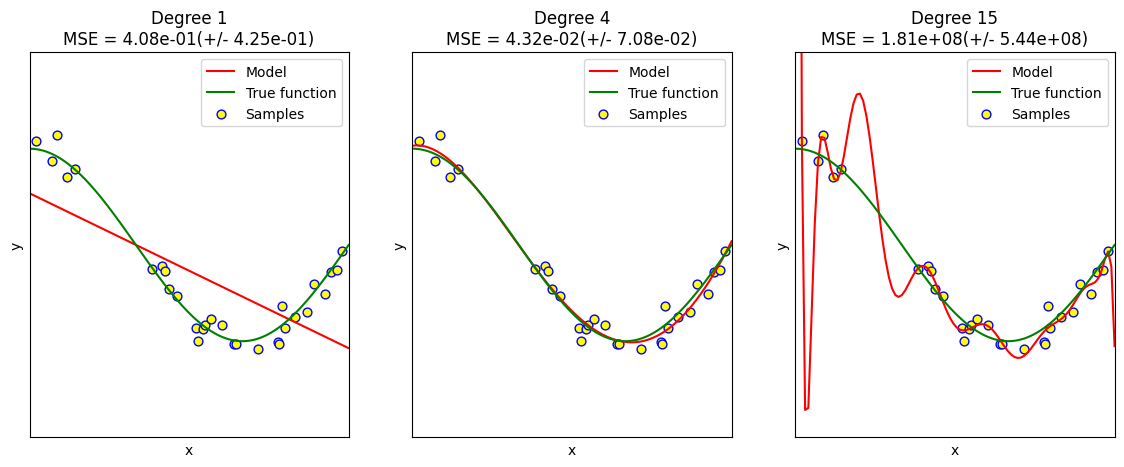
degrees[i], -scores.mean(), scores.std()

)

)

plt.show()

**OUTPUT:**

****

**EXPERIMENT:4**

**AIM:** To demonstrate the performance of a linear regression by using chosen database with python code

**PROGRAM: LINEAR REGRESSION**

import numpy as np

import matplotlib.pyplot as plt

def estimate\_coef(x, y):

n = np.size(x)

m\_x = np.mean(x)

m\_y = np.mean(y)

SS\_xy = np.sum(y\*x) - n\*m\_y\*m\_x

SS\_xx = np.sum(x\*x) - n\*m\_x\*m\_x

b\_1 = SS\_xy / SS\_xx

b\_0 = m\_y - b\_1\*m\_x

return (b\_0, b\_1)

def plot\_regression\_line(x, y, b):

plt.scatter(x, y, color="green", marker="o", s=50, label="Data points")

y\_pred = b[0] + b[1] \* x

plt.plot(x, y\_pred, color="purple", label="Regression line", linewidth=2)

plt.xlabel('x')

plt.ylabel('y')

plt.legend()

plt.show()

def main():

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

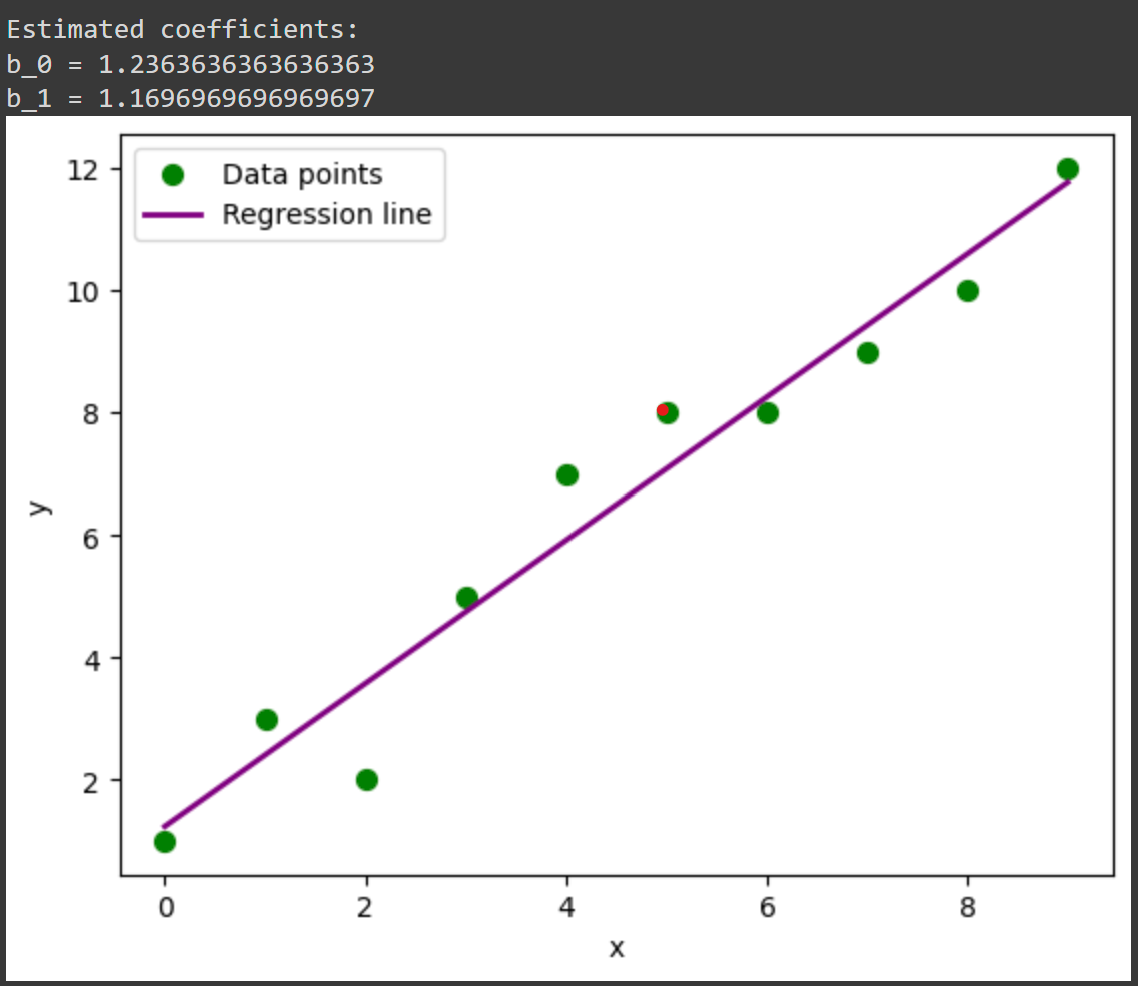
b = estimate\_coef(x, y)

print("Estimated coefficients:\nb\_0 = {} \nb\_1 = {}".format(b[0], b[1]))

plot\_regression\_line(x, y, b)

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

main()

**OUTPUT: **

**EXPERIMENT:5**

**AIM :** To demonstrate the performance of a logistic regression by using a chosen database with python code.

**PROGRAM:**

import numpy as np

import matplotlib.pyplot as plt

def sigmoid(z):

return 1 / (1 + np.exp(-z))

x = np.arange(-5, 5, 0.1)

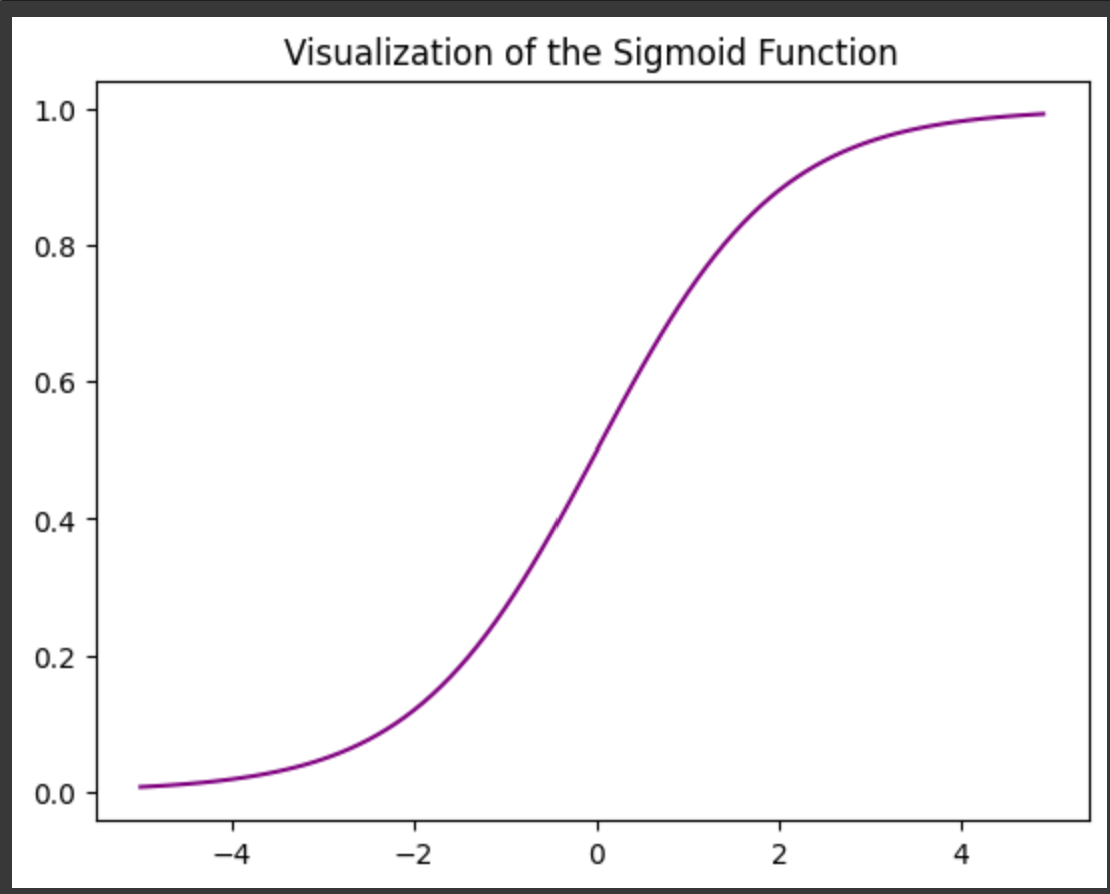
y = sigmoid(x)

plt.plot(x, y, color="purple")

plt.title('Visualization of the Sigmoid Function')

plt.show()

**OUTPUT:**



**EXPERIMENT:6(a)KNN**

**AIM:** Finding accuracy value of iris data set using KNN algorithm

**PROGRAM:**

import numpy as np

import pandas as pd

dataset = pd.read\_csv("/content/IRIS.csv")

"""

The breast cancer dataset has the following features: Sample code number, Clump Thickness, Uniformity of Cell Size,

Uniformity of Cell Shape, Marginal Adhesion, Single Epithelial Cell Size, Bare Nuclei, Bland Chromatin,

Normal Nucleoli, Mitosis, Class.

"""

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

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

dataset.shape

#splitting the dataset into the Training set and Test set

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 = 42)

#Feature Scaling

"""

Feature scaling is the process of converting the data into a given range.

In this case, the standard scalar technique is used.

"""

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

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

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

#Training the K-Nearest Neighbors (K-NN) Classification model on the Training set

"""

Once the dataset is scaled, next, the K-Nearest Neighbors (K-NN) classifier algorithm is used to create a model.

The hyperparameters such as n\_neighbors, metric, and p are set to 5, Minkowski, and 2 respectively.

The remaining hyperparameters are set to default values.

"""

from sklearn.neighbors import KNeighborsClassifier

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

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

"""

Display the results (confusion matrix and accuracy)

Here evaluation metrics such as confusion matrix and accuracy are used to evaluate the performance of the model built using a decision tree classifier.

"""

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

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

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

print(cm)

accuracy\_score(y\_test, y\_pred)

**OUTPUT:**

****

**EXPERIMENT:6(B)NAVIE**

**AIM: :** finding accuracy value of iris data set using NAIVE BAYES algorithm

**PROGRAM:**

import numpy as np

import pandas as pd

#Importing the dataset

"""

Next, we import or read the dataset. Click here to download the breast cancer dataset used in this implementation.

After reading the dataset, divide the dataset into concepts and targets. Store the concepts into X and

targets into y.

"""

dataset = pd.read\_csv("/content/IRIS.csv ")

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

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

"""

Splitting the dataset into the Training set and Test set

Once the dataset is read into the memory, next, divide the dataset into two parts, training and

testing using the train\_test\_split function from sklearn.

The test\_size and random\_state attributes are set to 0.25 and 0 respectively.

You can change these attributes as per your requirements.

"""

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)

"""

Feature scaling is the process of converting the data into a min-max range. In this case,

The standard scalar method is used.

"""

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

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

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

"""

Training the Naive Bayes Classification model on the Training set

Once the dataset is scaled, next, the Naive Bayes classifier algorithm is used to create a model.

The GaussianNB function is imported from sklearn.naive\_bayes library. The hyperparameters such as kernel,

and random\_state to linear, and 0 respectively. The remaining hyperparameters of the support vector machine

algorithms are set to default values.

"""

from sklearn.naive\_bayes import GaussianNB

classifier = GaussianNB()

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

GaussianNB(priors=None, var\_smoothing=1e-09)

"""

Here evaluation metrics such as confusion matrix and accuracy are used to evaluate the performance of

the model built using a decision tree classifier.

"""

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

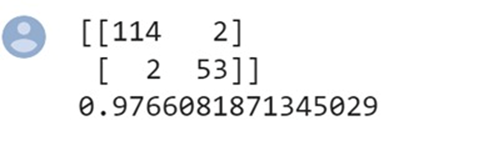
y\_pred = classifier.predict(X\_test)

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

print(cm)

accuracy\_score(y\_test, y\_pred)

**OUTPUT:**

****

**EXPERIMENT:6(C)LOGISTIC**

**AIM: :** finding accuracy value of iris data set using LOGISTIC REGRESSION algorithm

**PROGRAM:**

import numpy as np

import pandas as pd

"""

After importing the necessary libraries, next, we import or read the dataset.

Click here to download the breast cancer dataset used in this implementation.

The breast cancer dataset has the following features:

Sample code number, Clump Thickness, Uniformity of Cell Size, Uniformity of Cell Shape, Marginal Adhesion,

Single Epithelial Cell Size, Bare Nuclei, Bland Chromatin, Normal Nucleoli, Mitosis, Class.

"""

dataset = pd.read\_csv("/content/IRIS.csv ")

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

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

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.30, random\_state = 2)

"""

Feature scaling is the process of converting the data into a given range. In this case, the standard scalar technique is used.

from sklearn.preprocessing import StandardScaler

"""

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

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

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

"""

Training the Logistic Regression (LR) Classification model on the Training set

Once the dataset is scaled, next, the Logistic Regression (LR) classifier algorithm is used to create a model.

The hyperparameters such as random\_state to 0 respectively.

The remaining hyperparameters Logistic Regression (LR) are set to default values.

"""

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state = 0)

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

LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,

intercept\_scaling=1, l1\_ratio=None, max\_iter=100,

multi\_class='warn', n\_jobs=None, penalty='l2',

random\_state=0, solver='warn', tol=0.0001, verbose=0,

warm\_start=False)

"""

Here evaluation metrics such as confusion matrix and accuracy are used to evaluate the performance of the model

built using a decision tree classifier.

"""

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

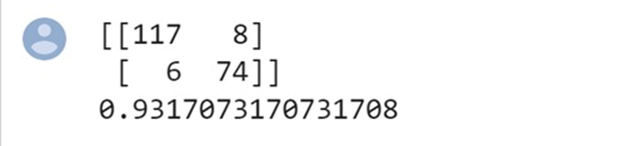
y\_pred = classifier.predict(X\_test)

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

print(cm)

accuracy\_score(y\_test, y\_pred)

**OUTPUT:**



**EXPERIMENT:6(D)DECISION**

**AIM: :** Finding accuracy value of iris data set using DECISION TREE algorithm

**PROGRAM:**

import numpy as np

import pandas as pd

dataset = pd.read\_csv("/content/IRIS.csv ")

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

y = dataset.iloc[:, -1].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 = 8)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

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

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

from sklearn.tree import DecisionTreeClassifier

classifier = DecisionTreeClassifier(criterion = 'entropy', random\_state = 5)

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

from sklearn.tree import plot\_tree

import matplotlib.pyplot as plt

plt.figure(figsize=(20,10))

plot\_tree(classifier, filled=True, rounded=True, feature\_names=dataset.columns[:-1])

plt.show()

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

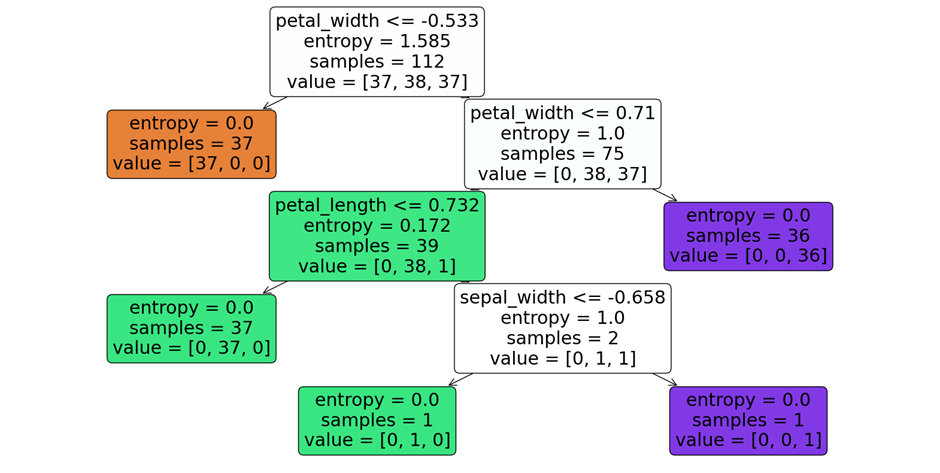
from sklearn.metrics import confusion\_matrix, accuracy\_score

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

print(cm)

accuracy\_score(y\_test, y\_pred)

**output:**



**0.8947368421052632**

**[[13 0 0]**

**[ 0 11 1]**

**[ 0 3 10]]**

**EXPERIMENT:6(E)SVM**

**AIM: :** finding accuracy value of iris data set using SVM algorithm

**PROGRAM:**

import numpy as np

import pandas as pd

dataset = pd.read\_csv("/content/IRIS.csv ")

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

y = dataset.iloc[:, -1].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=32)

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='linear', random\_state=0)

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

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

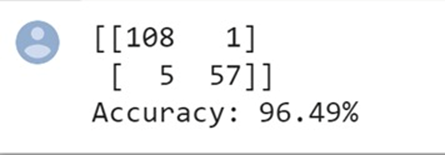
from sklearn.metrics import confusion\_matrix, accuracy\_score

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

print(cm)

print('Accuracy: {:.2f}%'.format(accuracy\_score(y\_test, y\_pred) \* 100))

**OUTPUT:**



**EXPERIMENT:6(F)RANDOM**

**AIM: :** finding accuracy value of iris data set using RANDOM FOREST algorithm

**PROGRAM:**

import numpy as np

import pandas as pd

dataset = pd.read\_csv("/content/IRIS.csv ")

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

y = dataset.iloc[:, -1].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=39)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

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

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

from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)

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

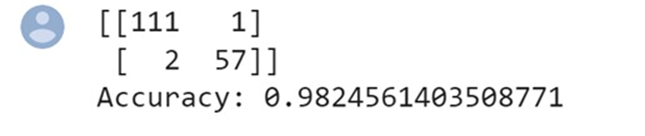
y\_pred = classifier.predict(X\_test)from sklearn.metrics import confusion\_matrix, accuracy\_score

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

print(cm)

print('Accuracy:', accuracy\_score(y\_test, y\_pred))

**OUTPUT:**

****

**EXPERIMENT:7(A)**

**AIM:** To demonstrate gradient descent using python(actual data)

**PROGRAM:**

import numpy as np

import matplotlib.pyplot as plt

def mean\_squared\_error(y\_true, y\_predicted):

cost = np.sum((y\_true - y\_predicted) \*\* 2) / len(y\_true)

return cost

def gradient\_descent(x, y, iterations=1000, learning\_rate=0.0001,

stopping\_threshold=1e-6):

current\_weight = 0.1

current\_bias = 0.01

n = float(len(x))

costs = []

weights = []

previous\_cost = None

for i in range(iterations):

y\_predicted = (current\_weight \* x) + current\_bias

current\_cost = mean\_squared\_error(y, y\_predicted)

if previous\_cost and abs(previous\_cost - current\_cost) <= stopping\_threshold:

break

previous\_cost = current\_cost

costs.append(current\_cost)

weights.append(current\_weight)

weight\_derivative = -(2/n) \* sum(x \* (y - y\_predicted))

bias\_derivative = -(2/n) \* sum(y - y\_predicted)

current\_weight -= (learning\_rate \* weight\_derivative)

current\_bias -= (learning\_rate \* bias\_derivative)

print(f"Iteration {i + 1}: Cost {current\_cost}, Weight {current\_weight}, Bias {current\_bias}")

plt.figure(figsize=(8, 6))

plt.plot(weights, costs, color='green')

plt.scatter(weights, costs, marker='o', color='orange')

plt.title("Cost vs Weights")

plt.ylabel("Cost")

plt.xlabel("Weight")

plt.show()

return current\_weight, current\_bias

def main():

X = np.array([32.50234527, 53.42680403, 61.53035803, 47.47563963, 59.81320787,

55.14218841, 52.21179669, 39.29956669, 48.10504169, 52.55001444,

45.41973014, 54.35163488, 44.1640495 , 58.16847072, 56.72720806,

48.95588857, 44.68719623, 60.29732685, 45.61864377, 38.81681754])

Y = np.array([31.70700585, 68.77759598, 62.5623823 , 71.54663223, 87.23092513,

78.21151827, 79.64197305, 59.17148932, 75.3312423 , 71.30087989,

55.16567715, 82.47884676, 62.00892325, 75.39287043, 81.43619216,

60.72360244, 82.89250373, 97.37989686, 48.84715332, 56.87721319])

estimated\_weight, estimated\_bias = gradient\_descent(X, Y, iterations=2000)

print(f"Estimated Weight: {estimated\_weight}\nEstimated Bias: {estimated\_bias}")

Y\_pred = estimated\_weight \* X + estimated\_bias

plt.figure(figsize=(8, 6))

plt.scatter(X, Y, marker='o', color='purple') # Changed scatter color to purple

plt.plot([min(X), max(X)], [min(Y\_pred), max(Y\_pred)], color='cyan', markerfacecolor='red',

markersize=10, linestyle='dashed') # Changed line color to cyan

plt.xlabel("X")

plt.ylabel("Y")

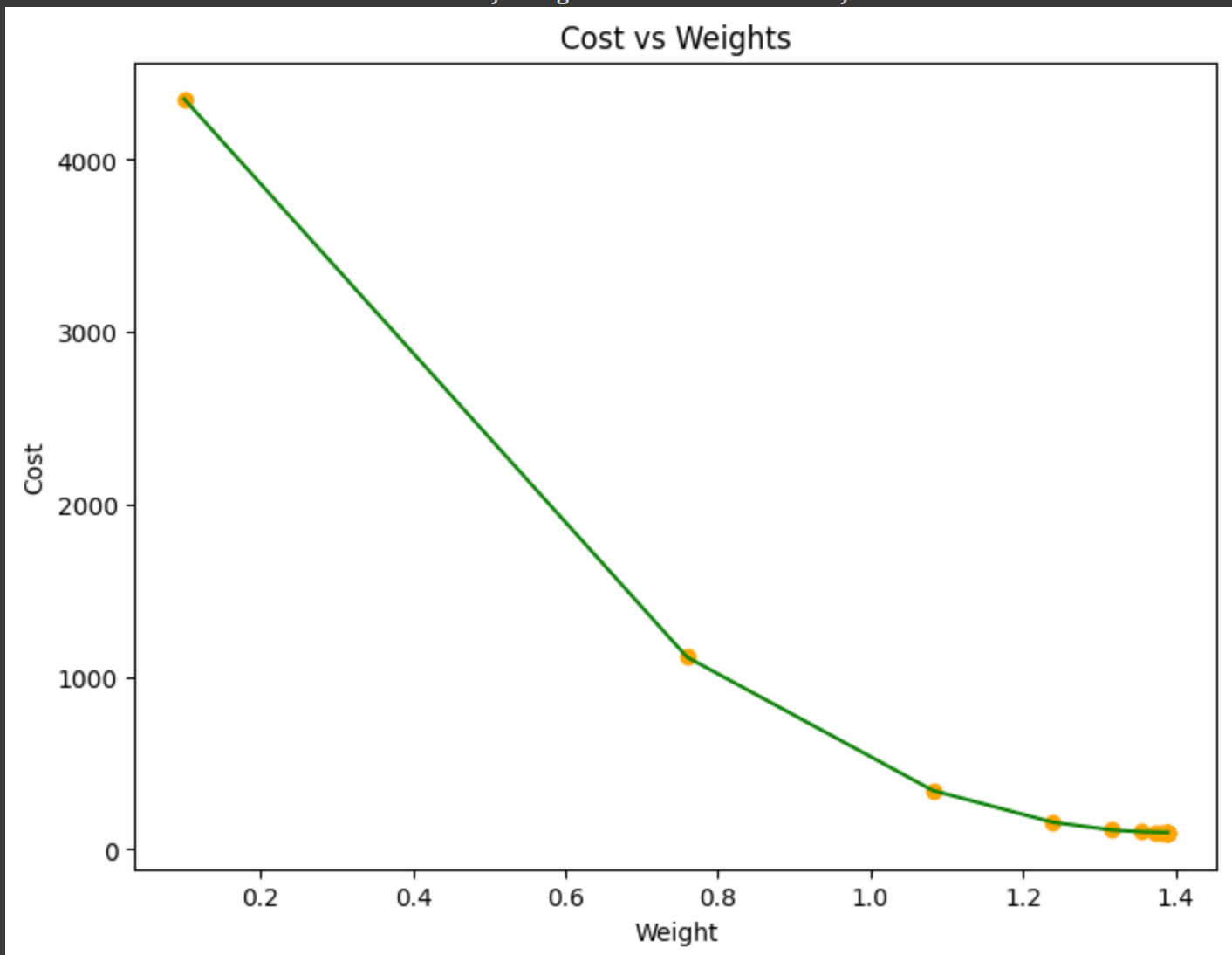
plt.title("Regression Line and Data Points")

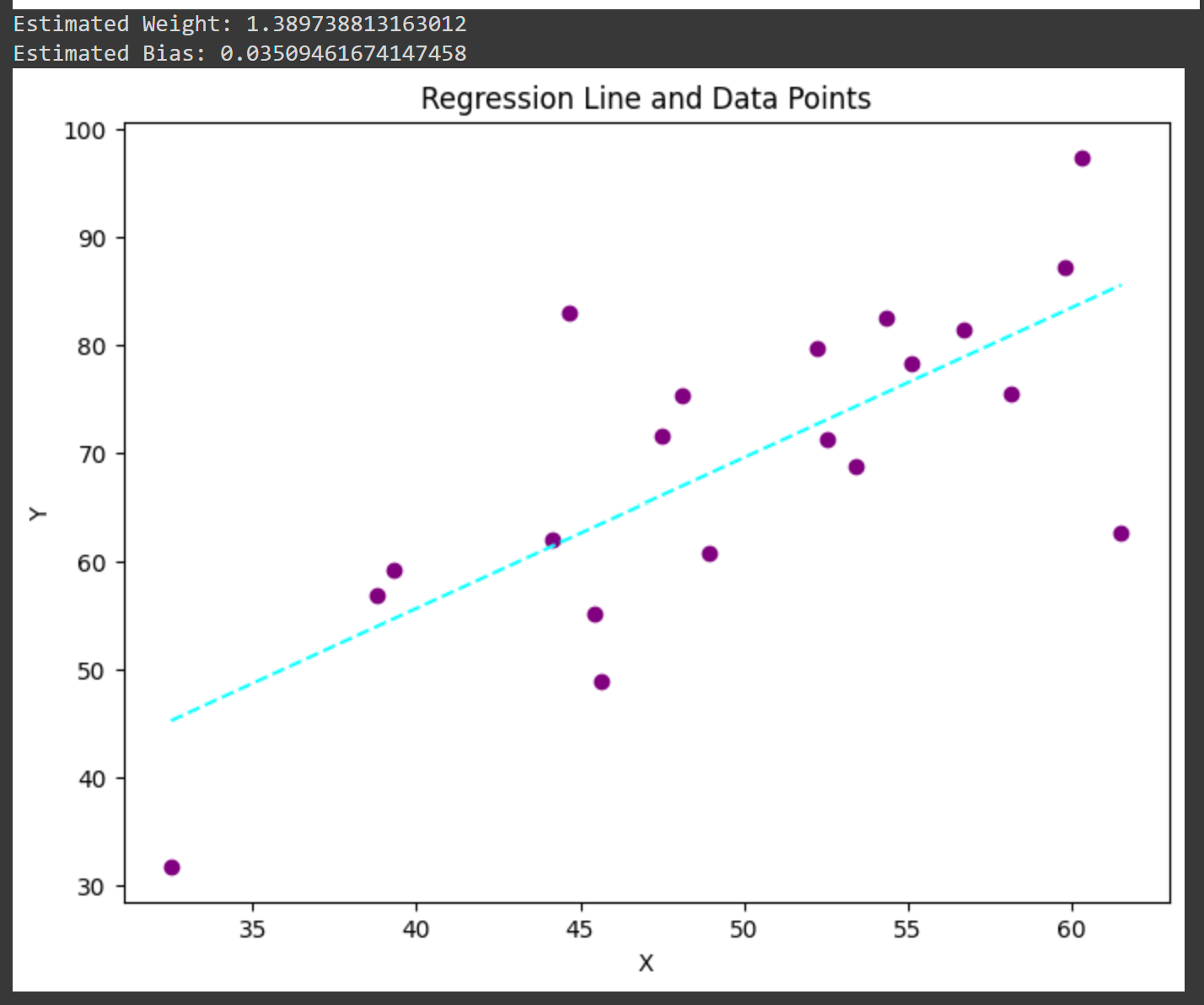
plt.show()

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

main()

**output:**

****

****

**Experiment:7(b)**

**AIM:** To demonstrate gradient descent using python( modified data)

**PROGRAM:**

import numpy as np

import matplotlib.pyplot as plt

def mean\_squared\_error(y\_true, y\_predicted):

cost = np.sum((y\_true - y\_predicted) \*\* 2) / len(y\_true)

return cost

def gradient\_descent(x, y, iterations=1000, learning\_rate=0.0001,

stopping\_threshold=1e-6):

current\_weight = 0.1

current\_bias = 0.01

n = float(len(x))

costs = []

weights = []

previous\_cost = None

for i in range(iterations):

y\_predicted = (current\_weight \* x) + current\_bias

current\_cost = mean\_squared\_error(y, y\_predicted)

if previous\_cost and abs(previous\_cost - current\_cost) <= stopping\_threshold:

break

previous\_cost = current\_cost

costs.append(current\_cost)

weights.append(current\_weight)

weight\_derivative = -(2/n) \* sum(x \* (y - y\_predicted))

bias\_derivative = -(2/n) \* sum(y - y\_predicted)

current\_weight -= (learning\_rate \* weight\_derivative)

current\_bias -= (learning\_rate \* bias\_derivative)

print(f"Iteration {i + 1}: Cost {current\_cost}, Weight {current\_weight}, Bias {current\_bias}")

plt.figure(figsize=(8, 6))

plt.plot(weights, costs, color='purple') # Changed line color to purple

plt.scatter(weights, costs, marker='o', color='yellow') # Changed scatter color to yellow

plt.title("Cost vs Weights")

plt.ylabel("Cost")

plt.xlabel("Weight")

plt.show()

return current\_weight, current\_bias

def main():

X = np.array([52.50234527, 63.42680403, 81.53035803, 47.47563963, 89.81320787,

55.14218841, 52.21179669, 39.29956669, 48.10504169, 52.55001444,

45.41973014, 54.35163488, 44.1640495 , 58.16847072, 56.72720806,

48.95588857, 44.68719623, 60.29732685, 45.61864377, 38.81681754])

Y = np.array([41.70700585, 78.77759598, 82.5623823 , 91.54663223, 77.23092513,

78.21151827, 79.64197305, 59.17148932, 75.3312423 , 71.30087989,

55.16567715, 82.47884676, 62.00892325, 75.39287043, 81.43619216,

60.72360244, 82.89250373, 97.37989686, 48.84715332, 56.87721319)

estimated\_weight, estimated\_bias = gradient\_descent(X, Y, iterations=2000)

print(f"Estimated Weight: {estimated\_weight}\nEstimated Bias: {estimated\_bias}")

Y\_pred = estimated\_weight \* X + estimated\_bias

plt.figure(figsize=(8, 6))

plt.scatter(X, Y, marker='o', color='magenta')

plt.plot([min(X), max(X)], [min(Y\_pred), max(Y\_pred)], color='green', linestyle='dashed')

plt.xlabel("X")

plt.ylabel("Y")

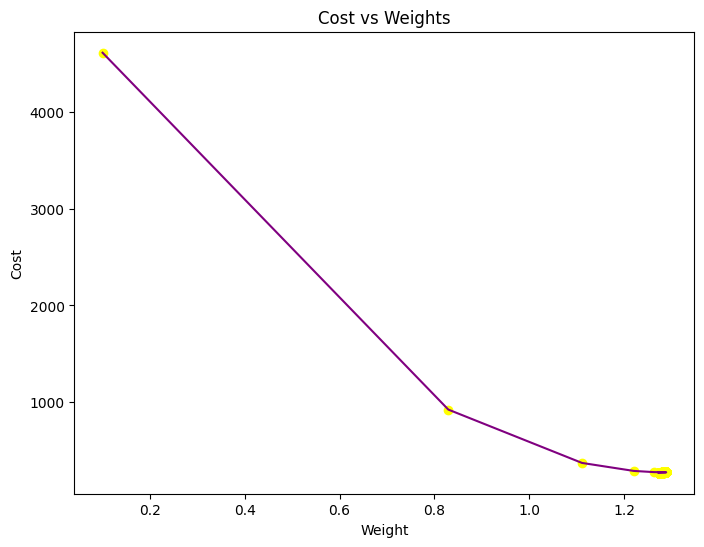
plt.title("Regression Line and Data Points")

plt.show()

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

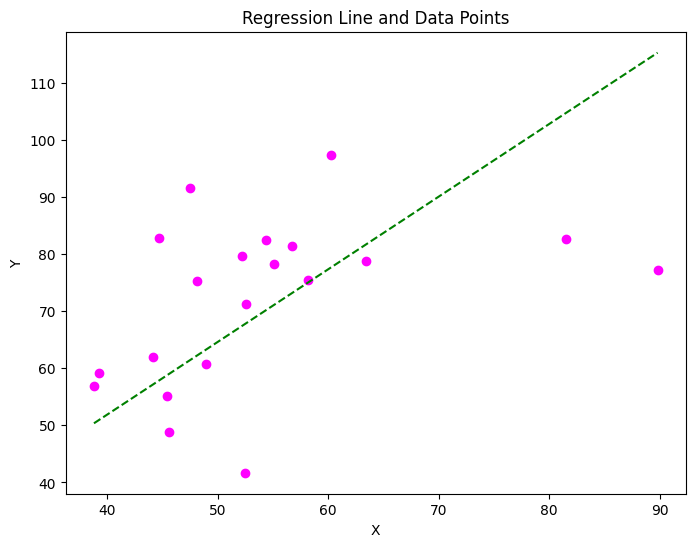
main()

**OUTPUT:**

****

**Estimated Weight: 1.272888252198252**

**Estimated Bias: 0.9558815898740776**

****

**EXPERIMENT:8(A)SEGMENTATION**

**AIM:** : Verifying the performance of a image processing by using chosen database with python code

**PROGRAM:**

**import numpy as np**

**import cv2**

**from matplotlib import pyplot as plt**

**img = cv2.imread(r'C33P1thinF\_IMG\_20150619\_114756a\_cell\_181.png')**

**b,g,r = cv2.split(img)**

**rgb\_img = cv2.merge([r,g,b])**

**gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)**

**ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH\_BINARY\_INV+cv2.THRESH\_OTSU)**

**kernel = np.ones((2,2),np.uint8)**

**closing = cv2.morphologyEx(thresh,cv2.MORPH\_CLOSE,kernel, iterations = 2)**

**sure\_bg = cv2.dilate(closing,kernel,iterations=3)**

**dist\_transform = cv2.distanceTransform(sure\_bg,cv2.DIST\_L2,3)**

**ret, sure\_fg = cv2.threshold(dist\_transform,0.1\*dist\_transform.max(),255,0)**

**sure\_fg = np.uint8(sure\_fg)**

**unknown = cv2.subtract(sure\_bg,sure\_fg)**

**ret, markers = cv2.connectedComponents(sure\_fg)**

**markers = markers+1**

**markers[unknown==255] = 0**

**markers = cv2.watershed(img,markers)**

**img[markers == -1] = [255,0,0]**

**plt.subplot(211),plt.imshow(rgb\_img)**

**plt.title('Input Image'), plt.xticks([]), plt.yticks([])**

**plt.subplot(212),plt.imshow(thresh, 'gray')**

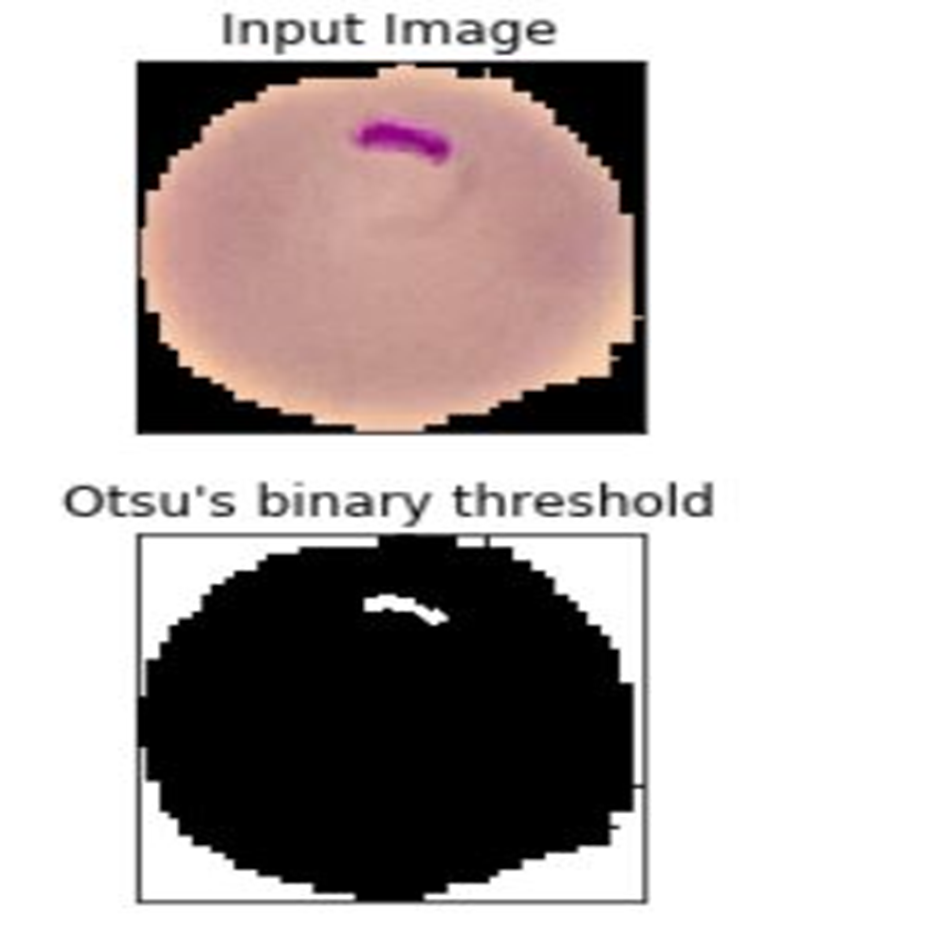
**plt.imsave(r'thresh.png',thresh)**

**plt.title("Otsu's binary threshold"), plt.xticks([]), plt.yticks([])**

**plt.tight\_layout()**

**plt.show()**

**OUTPUT:**



**EXPERIMENT:8(B)**

**AIM:** : Verifying the performance of a image processing by using water shed database with python code

**PROGRAM:**

import numpy as np

import cv2

from matplotlib import pyplot as plt

img = cv2.imread(r'C33P1thinF\_IMG\_20150619\_114756a\_cell\_181.png')

b,g,r = cv2.split(img)

rgb\_img = cv2.merge([r,g,b])

gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH\_BINARY\_INV+cv2.THRESH\_OTSU)

plt.subplot(211),plt.imshow(closing, 'gray')

plt.title("morphologyEx:Closing:2x2"), plt.xticks([]), plt.yticks([])

plt.subplot(212),plt.imshow(sure\_bg, 'gray')

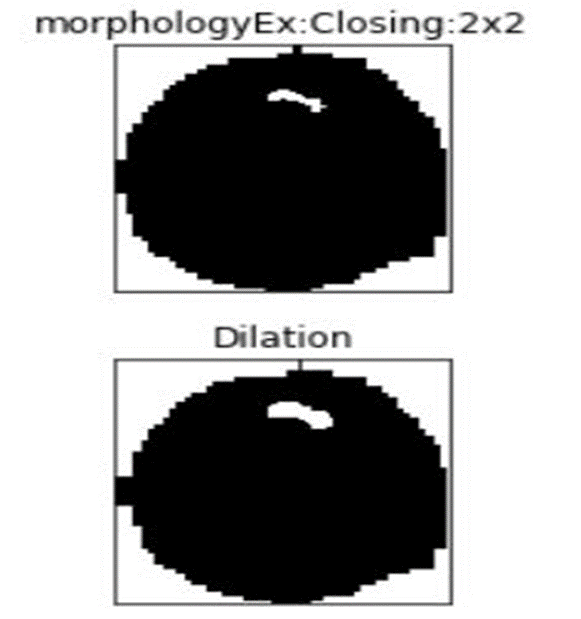
plt.imsave(r'dilation.png',sure\_bg)

plt.title("Dilation"), plt.xticks([]), plt.yticks([])

plt.tight\_layout()

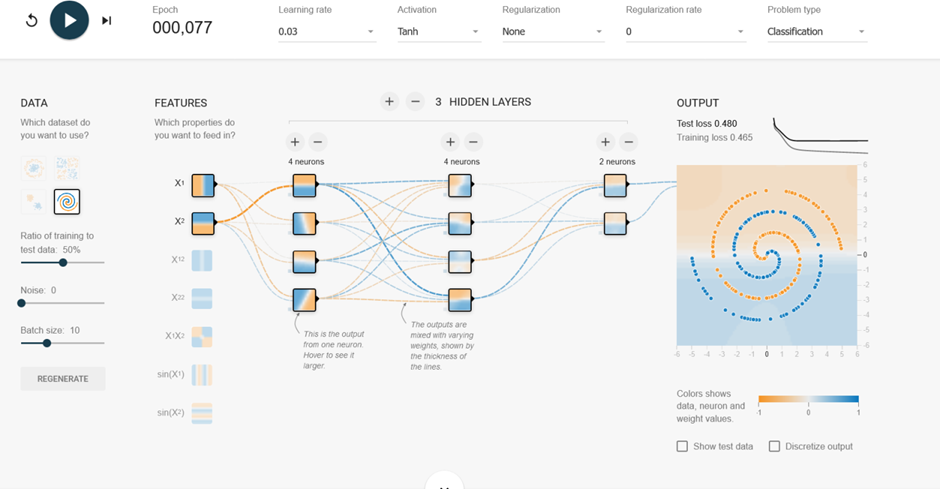
plt.show()

**OUTPUT:**

**:**

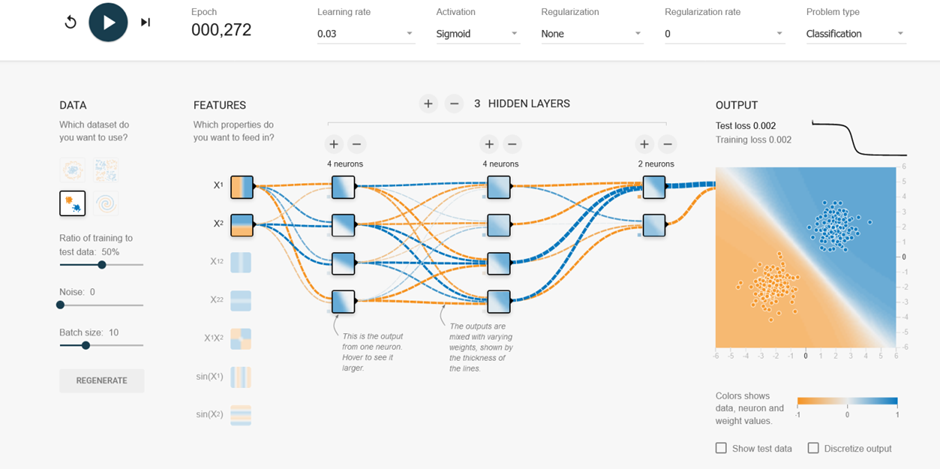
**EXPERIMENT:9 (a) TANH**

**AIM:** Neural network analysis using TANH activation

**OUTPUT:**

**EXPERIMENT:9(B) SIGMIOD**

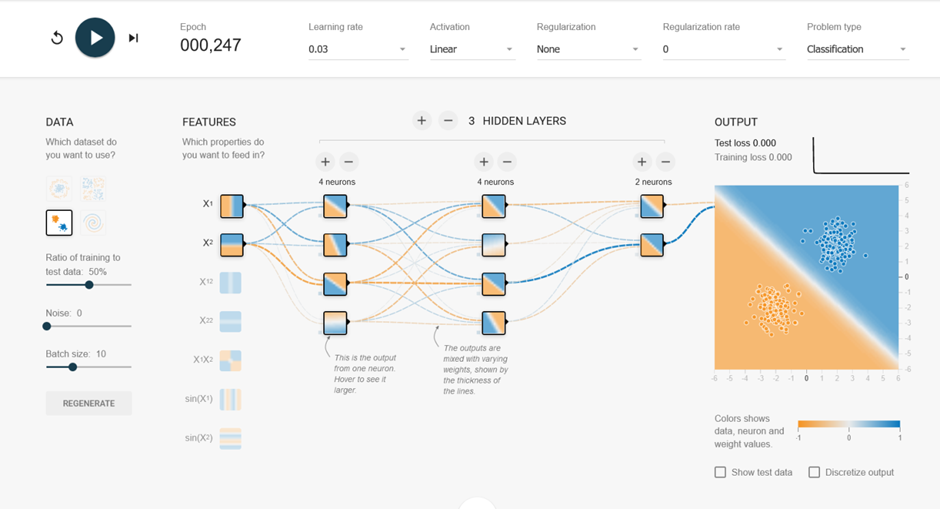
**AIM:** Neural network analysis using SIGMOID activation

****

**EXPERIMENT:9(C) LINEAR**

**AIM:** Neural network analysis using LINEAR activation

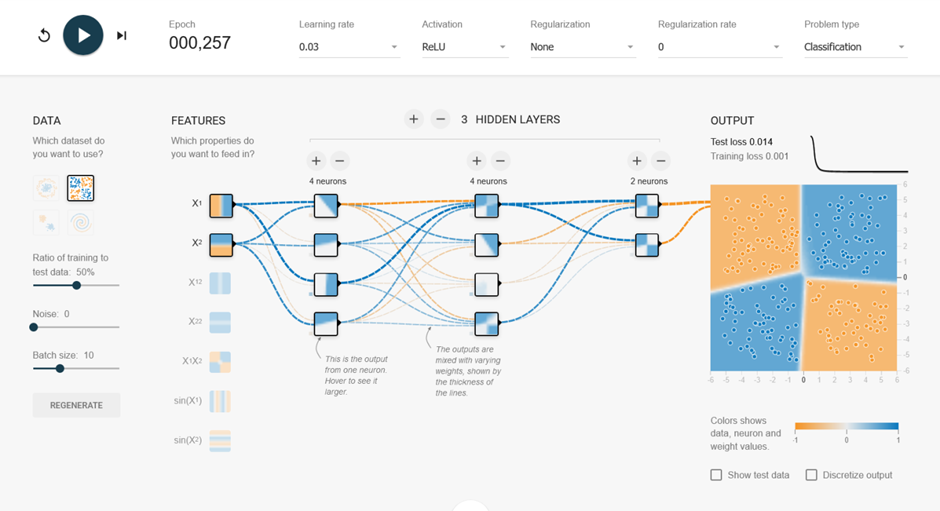
**OUTPUT:**



**EXPERIMENT:9(D)RELU**

**AIM:** Neural network analysis using ReLU activation

**OUTPUT:**

****

**EXPERIMENT:10**

**AIM:** To demonstrate linear separability using python code

**PROGRAM:**

import numpy as np

import matplotlib.pyplot as plt

def estimate\_coef(x, y):

n = np.size(x)

m\_x = np.mean(x)

m\_y = np.mean(y)

SS\_xy = np.sum(y \* x) - n \* m\_y \* m\_x

SS\_xx = np.sum(x \* x) - n \* m\_x \* m\_x

b\_1 = SS\_xy / SS\_xx

b\_0 = m\_y - b\_1 \* m\_x

return (b\_0, b\_1)

def plot\_regression\_line(x, y, b):

plt.scatter(x, y, color="purple", marker="o", s=50)

y\_pred = b[0] + b[1] \* x

plt.plot(x, y\_pred, color="orange")

plt.xlabel('x')

plt.ylabel('y')

plt.show()

def main():

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

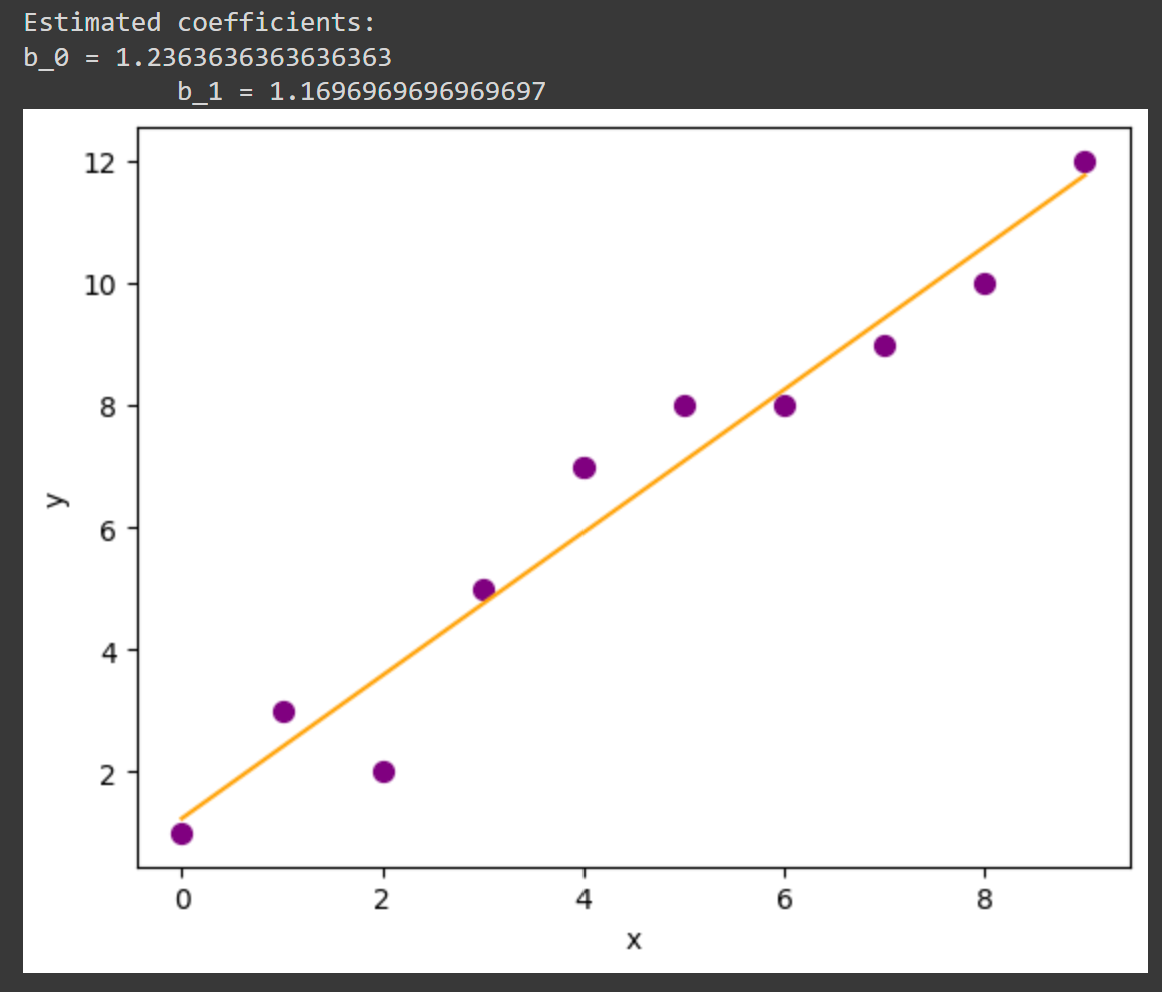
b = estimate\_coef(x, y)

print("Estimated coefficients:\nb\_0 = {}\n b\_1 = {}".format(b[0], b[1]))

plot\_regression\_line(x, y, b)

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

main()

**OUTPUT:**