**Q.1) Write a program to Plot the correlation plot on dataset and visualize giving an overview of relationships among data on iris data.**

import seaborn as sns

import matplotlib.pyplot as plt

# Load the iris dataset

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

# Calculate the correlation matrix

corr = iris.corr()

# Plot the heatmap

plt.figure(figsize=(10, 8)) # Set the size of the figure

sns.heatmap(corr, annot=True, cmap='coolwarm', vmin=-1, vmax=1)

# Display the plot

plt.title("Correlation Plot of Iris Dataset")

plt.show()

Output:



**Q2) Write a program to implement linear regression algorithm to create and evaluate a model on a given dataset**

import numpy as np

import matplotlib.pyplot as plt

def estimate\_coef(x, y):

# number of observations/points

n = np.size(x)

# mean of x and y vector

m\_x = np.mean(x)

m\_y = np.mean(y)

# calculating cross-deviation and deviation about x

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

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

# calculating regression coefficients

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):

# plotting the actual points as scatter plot

plt.scatter(x, y, color = "m",

marker = "o", s = 30)

# predicted response vector

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

# plotting the regression line

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

# putting labels

plt.xlabel('x')

plt.ylabel('y')

# function to show plot

plt.show()

def main():

# observations / data

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])

# estimating coefficients

b = estimate\_coef(x, y)

print("Estimated coefficients:\nb\_0 = {} \

\nb\_1 = {}".format(b[0], b[1]))

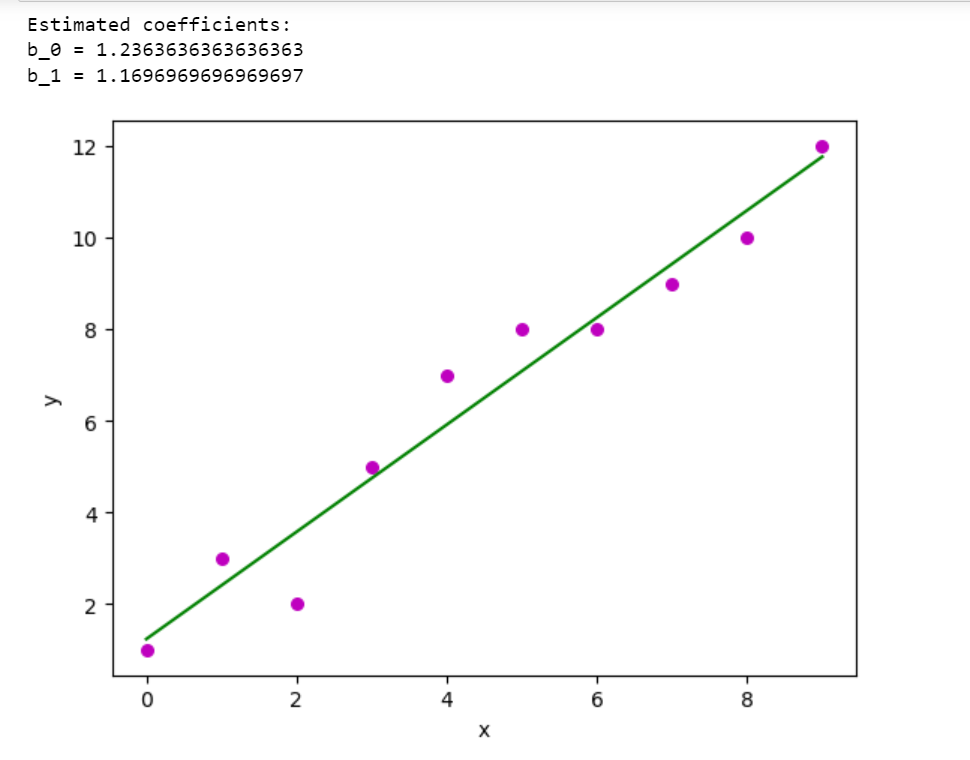
# plotting regression line

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

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

main()

Output:



**Q3) Write a program to classify the given dataset using logistic regression and evaluate the model**

import numpy as np

import pandas as pd

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

from sklearn.linear\_model import LogisticRegression

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

# Load iris dataset

from sklearn.datasets import load\_iris

data = load\_iris()

df = pd.DataFrame(data.data, columns=data.feature\_names)

df['species'] = data.target

# Set target variable to 1 if species is 'setosa', and 0 otherwise

y = (df['species'] == 0).astype(int)

X = df.drop('species', axis=1)

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create a logistic regression model

model = LogisticRegression(max\_iter=1000)

# Train the model

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

# Predict values for the test set

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

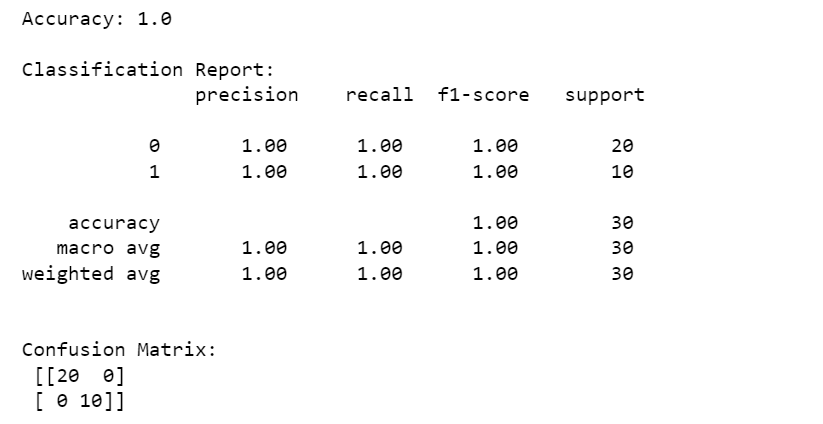
# Evaluate the model

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

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

print("\nConfusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

Output:



**Q4) Write a program to implement support vector machine algorithm**

# Import necessary libraries

from sklearn import datasets

from sklearn import svm

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

from sklearn.metrics import accuracy\_score

# Load a sample dataset (e.g., the Iris dataset)

iris = datasets.load\_iris()

X = iris.data

y = iris.target

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Create an SVM classifier (you can choose different kernel types, such as 'linear', 'rbf', etc.)

clf = svm.SVC(kernel='linear')

# Fit the classifier on the training data

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

# Make predictions on the test data

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

# Calculate the accuracy of the classifier

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

print(f"Accuracy: {accuracy}")

# You can also get the support vectors and other parameters

print("Support Vectors:")

print(clf.support\_vectors\_)

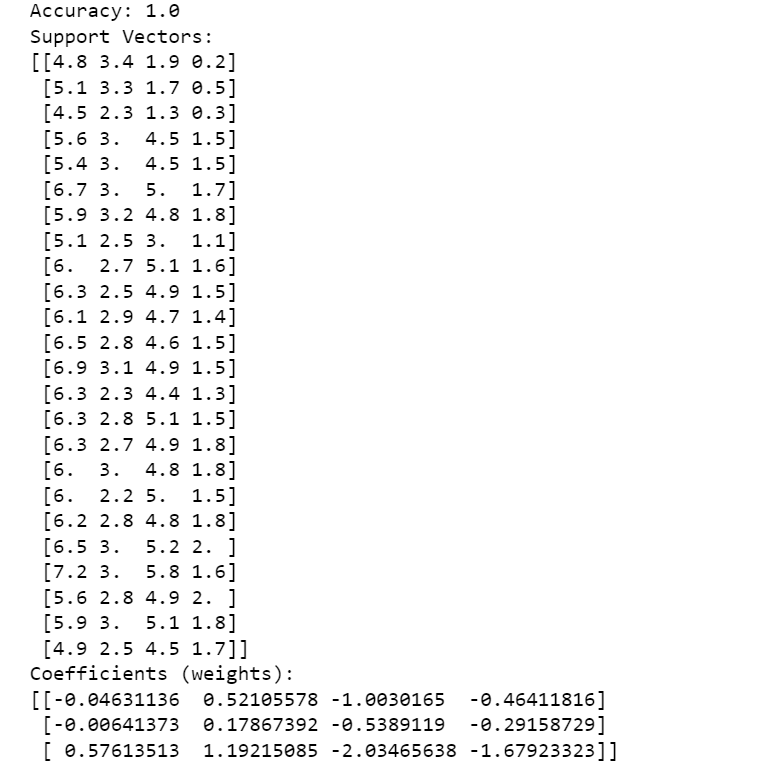
print("Coefficients (weights):")

print(clf.coef\_)

print("Intercepts:")

print(clf.intercept\_)

output:



**Q5) Write a program to implement Decision Tree model on the given dataset**

import pandas

from sklearn import tree

from sklearn.tree import DecisionTreeClassifier

import matplotlib.pyplot as plt

df = pandas.read\_csv(r"C:\Users\hp\Downloads\data.csv")

d = {'UK': 0, 'USA': 1, 'N': 2}

df['Nationality'] = df['Nationality'].map(d)

d = {'YES': 1, 'NO': 0}

df['Go'] = df['Go'].map(d)

features = ['Age', 'Experience', 'Rank', 'Nationality']

X = df[features]

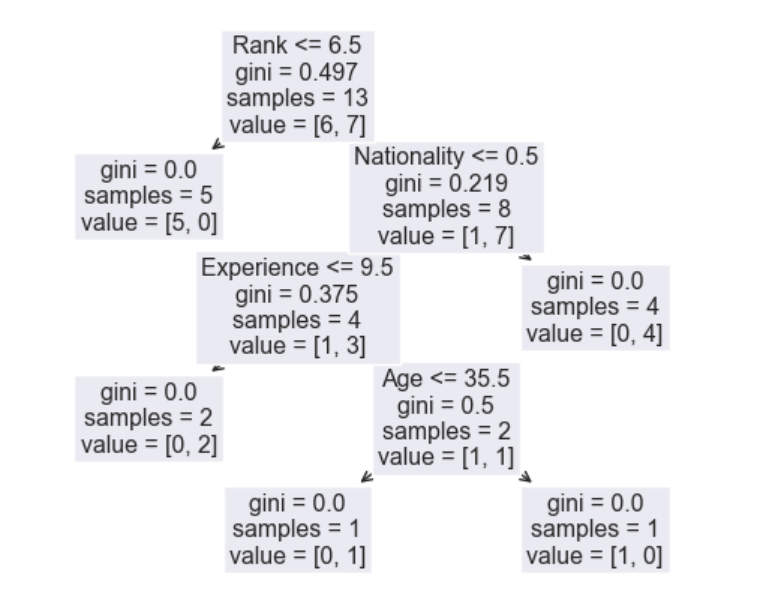
y = df['Go']

dtree = DecisionTreeClassifier()

dtree = dtree.fit(X, y)

tree.plot\_tree(dtree, feature\_names=features)

output:



**Q6) Write a program to implement Bayesian classification on given dataset.**

import numpy as np

import pandas as pd

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

from sklearn.naive\_bayes import GaussianNB

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

# Load iris dataset

from sklearn.datasets import load\_iris

data = load\_iris()

df = pd.DataFrame(data.data, columns=data.feature\_names)

df['species'] = data.target

X = df.drop('species', axis=1)

y = df['species']

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create a Gaussian Naive Bayes classifier

model = GaussianNB()

# Train the model

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

# Predict values for the test set

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

# Evaluate the model

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

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

print("\nConfusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

Output:

