**CO1**

**Experiment 1**

1. **Create a numpy array and perform the following operations**
2. **Append values to the end of an array.**
3. **Insert values into an array at a specified position.**
4. **Delete elements from an array.**
5. **Find unique elements in an array.**
6. **Sort an array.**
7. **Save an array to a text file.**
8. **Load data from a text file into an array.**

import numpy as np

arr = np.array([3, 2, 1, 2])

print("Original array: ", arr)

print("Append (6,7,8): ",np.append(arr, [6, 7, 8]))

print("Insert Specific (10,11) at third second position: ", np.insert(arr, 2, [10, 11]))

print("Delete values (1,3): ", np.delete(arr, [0, 2]))

print("Unique element: ", np.unique(arr))

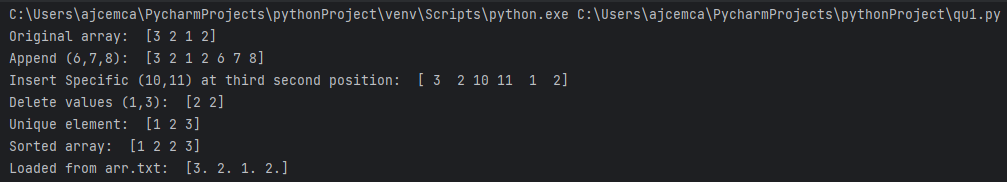
print("Sorted array: ", np.sort(arr))

np.savetxt('arr.txt', arr)

ld = np.loadtxt('arr.txt')

print("Loaded from arr.txt: ", ld)

**Output:**



1. **Task: You have two NumPy arrays, arr1 and arr2, containing the following data:**

**arr1 = np.array([1, 2, 3, 4, 5])**

**arr2 = np.array([6, 7, 8, 9, 10])**

**Write NumPy code to perform the following operations:**

1. **Add arr1 and arr2 to create a new array called result\_add.**
2. **Multiply arr1 and arr2 to create a new array called result\_multiply.**
3. **Calculate the mean of result\_add.**
4. **Find the maximum value in result\_multiply.**

import numpy as np

arr1 = np.array([1, 2, 3, 4, 5])

arr2 = np.array([6, 7, 8, 9, 10])

result\_add = arr1 + arr2

print("Sum of array: ", result\_add)

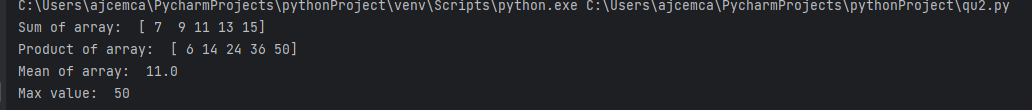
result\_multiply = arr1 \* arr2

print("Product of array: ", result\_multiply)

print("Mean of array: ", np.mean(result\_add))

print("Max value: ", np.max(result\_multiply))

**Output:**



1. **Task: You have a NumPy array called grades that represents the scores of students in a class:**

**grades = np.array([85, 90, 78, 92, 88, 76, 95, 89, 84, 91])**

**Write NumPy code to answer the following questions:**

* 1. **What is the average (mean) grade in the class?**
  2. **How many students scored above 90?**
  3. **Calculate the standard deviation of the grades.**

import numpy as np

grades = np.array([85, 90, 78, 92, 88, 76, 95, 89, 84, 91])

print("Average grade: ", np.mean(grades))

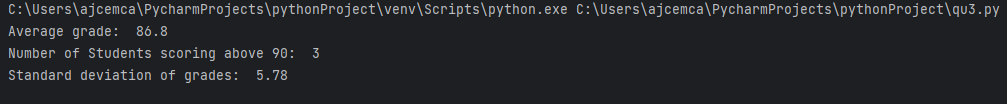
filter\_grade = grades[grades > 90]

print("Number of Students scoring above 90: ", len(filter\_grade))

std\_grade = np.std(grades)

print("Standard deviation of grades: ", np.round(std\_grade, decimals=2))

**Output**

:

1. **Matrix Operations with NumPy**

**Define the matrices**

**matrix\_A = np.array([[1, 2, 3],**

**[4, 5, 6],**

**[7, 8, 9]])**

**matrix\_B = np.array([[9, 8, 7],**

**[6, 5, 4],**

**[3, 2, 1]])**

**Perform the following matrix operations:**

1. **Add matrix\_A and matrix\_B element-wise to create a new matrix, matrix\_sum.**
2. **Multiply matrix\_A and matrix\_B element-wise to create a new matrix, matrix\_product.**
3. **Calculate the matrix product of matrix\_A and matrix\_B (dot product) and store it in matrix\_dot.**
4. **Transpose matrix\_A and store it in matrix\_A\_transpose.**
5. **Calculate the determinant of matrix\_B and store it in determinant\_B.**
6. **Find the eigenvalues and eigenvectors of matrix\_A and store them in eigenvalues\_A and eigenvectors\_A.**
7. **Find SVD of a matrix**

**Print the results of each operation.**

import numpy as np

from scipy.linalg import svd

matrix\_A = np.array([[1, 2, 3],[4, 5, 6],[7, 8, 9]])

matrix\_B = np.array([[9, 8, 7],[6, 5, 4],[3, 2, 1]])

matrix\_sum = matrix\_A + matrix\_B

print("Sum of matrices: \n", matrix\_sum)

matrix\_product = matrix\_A \* matrix\_B

print("Product of matrices: \n", matrix\_product)

matrix\_dot = np.dot(matrix\_A, matrix\_B)

print("Dot Product of matrices: \n", matrix\_dot)

matrix\_A\_transpose = np.transpose(matrix\_A)

print("Transpose of matrix A: \n", matrix\_A\_transpose)

determinant\_B = np.linalg.det(matrix\_B)

print("Determinant of matrix B: ", determinant\_B)

eigenvalues\_A, eigenvectors\_A = np.linalg.eig(matrix\_A)

print("Eigenvalues of matrix A: \n", eigenvalues\_A)

print("Eigenvectors of matrix A: \n", eigenvectors\_A)

U, s, VT = svd(matrix\_A)

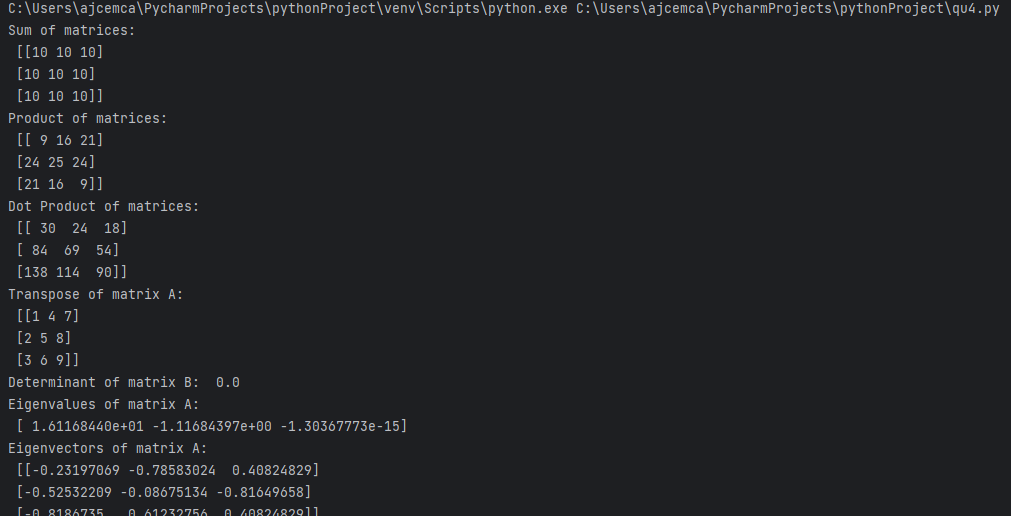
print("SVD of Martrix A: ")

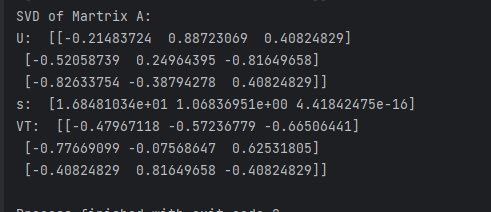
print("U: ",U)

print("s: ",s)

print("VT: ",VT)

**Output:**





1. **Task: You have a CSV file named "sales\_data.csv" containing sales data with**

**columns for "Date," "Product," "Quantity," and "Revenue." Load this data using Pandas and answer the following questions:**

1. **How many rows and columns are there in the dataset?**
2. **What is the total revenue for all the sales?**

import pandas as pd

df = pd.read\_csv("sales\_data.csv")

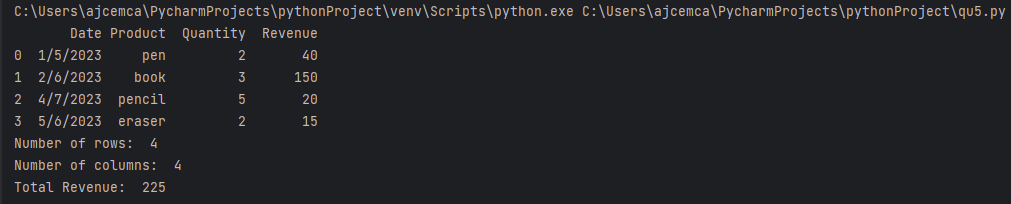
print(df.head())

print("Number of rows: ", len(df))

print("Number of columns: ", len(df.columns))

print("Total Revenue: ", sum(df['Revenue']))

**Output:**



1. **Task: You have a DataFrame called "student\_data" with columns "Student\_ID," "Name," "Age," and "GPA." Perform the following operations using Pandas:**
2. **Filter and display the rows of students who are 20 years old or older.**
3. **Calculate the average GPA of the students in the DataFrame.**
4. **Sort the DataFrame in descending order of GPA and display the top 5 students with the highest GPAs.**
5. **Group the students by their ages and calculate the average GPA for each age group.**

import pandas as pd

df = pd.read\_csv("student\_data.csv")

print(df.head(), "\n")

age = df[df['Age'] > 19]

print("Students who are 20 years old or older: \n", age)

print("\nAverage GPA of all students:",df['GPA'].mean().round(2))

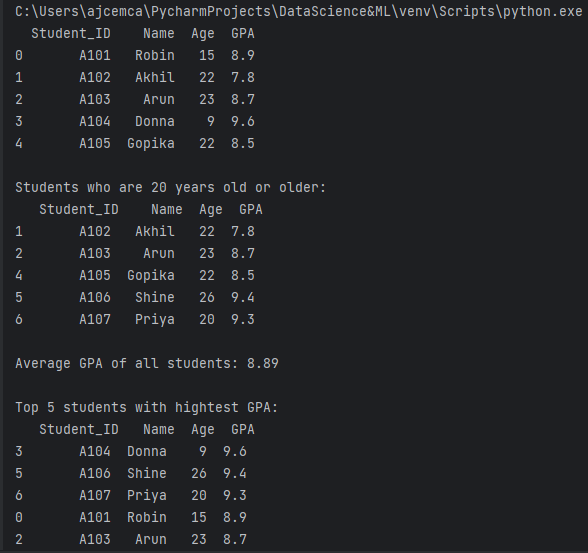
data1 = df.sort\_values(by='GPA', ascending= False)

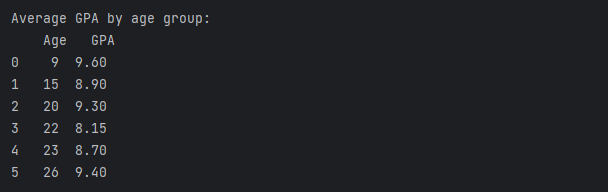
print("\nTop 5 students with hightest GPA: \n", data1.head(5))

data2 = df.groupby('Age')['GPA'].mean().reset\_index()

print("\nAverage GPA by age group: \n", data2)

**Output:**

****

****

**7. Histogram and Quartile Plot:**

**Objective: Visualize the distribution of a univariate dataset and analyze its quartiles.**

**Question: Use a dataset of your choice (e.g., exam scores of students, employee salaries, or any other numerical data). Create a histogram to visualize the data's distribution. Afterward, plot quartiles (e.g., Q1, Q2, Q3) on the same graph. Answer the following questions:**

1. **What does the histogram reveal about the data's distribution?**
2. **How do the quartiles relate to the histogram?**
3. **Are there any outliers in the data, and if so, how do they affect the quartiles?**

**Output: Provide the histogram and quartile plot along with a written analysis.**

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

marks = np.array([10,18,34,

37, 33, 38,

34, 24, 80,

45, 49, 27,

31, 35, 42])

fig, ax = plt.subplots(figsize =(4, 4))

ax.hist(marks, color = "darkcyan", ec="black", lw=1)

plt.title('Histogram: Exam Score')

plt.ylabel('No. of students')

plt.xlabel('Score')

plt.figure(figsize=(4, 4))

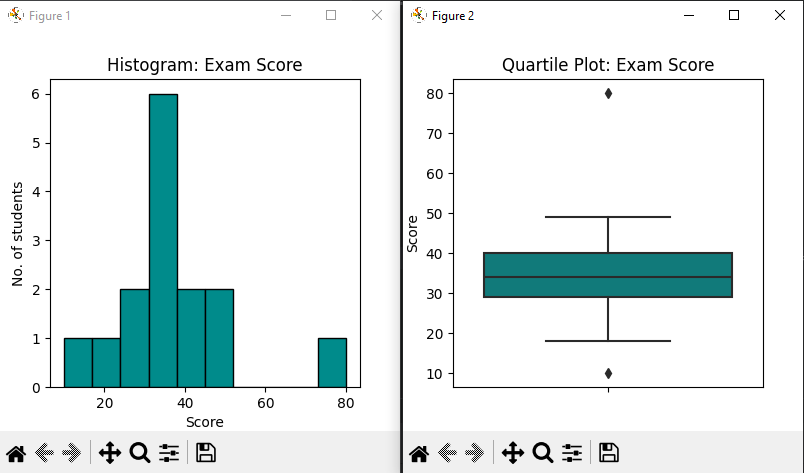
sns.boxplot(y=marks, color='darkcyan')

plt.title('Quartile Plot: Exam Score')

plt.ylabel('Score')

plt.show()

Output:



1. The histogram reveals that the largest distribution of score is between 30 and 38 with 6 students. The distribution between 10-24 and 73-80 is the same with 1 student.
2. The Quartile plot shows that the distribution in lower quartile is less than the upper quartile which reveals a higher score among the students.The InterQuartile range(IQR) between 28 and 40 reveals where the majority scores are distributed. We also obtain a median of 34 for the data.
3. It is revealed that two outliers at 10 and 80 exist which are translated better in Quartile plot than histogram. These outliers remain clearly out of the minimum and maximum of the plot thereby signifying an unusual difference from the usual distribution.

**Question 8:**

**Distribution Chart and Scatter Plot:**

**Objective: Explore the relationship between two variables and visualize their distributions.**

**Question: Choose a dataset that contains two numerical variables (e.g., income vs. education level, temperature vs. ice cream sales). Create a distribution chart for each variable and a scatter plot to visualize their relationship. Answer the following questions:**

1. **What do the distribution charts reveal about each variable?**
2. **Is there a correlation between the two variables based on the scatter plot?**
3. **Can you identify any patterns or trends in the data?**

**Output: Present the distribution charts, scatter plot, and your observations in a report.**

**Heat Map charts:-**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

x\_values = [1, 2, 3, 4, 5]

y\_values = [10, 15, 13, 18, 20]

data\_values = [10, 15, 13, 18, 20]

df = pd.DataFrame({'x': x\_values, 'y': y\_values, 'value': data\_values})

heatmap\_data = df.pivot\_table(index='x', columns='y', values='value')

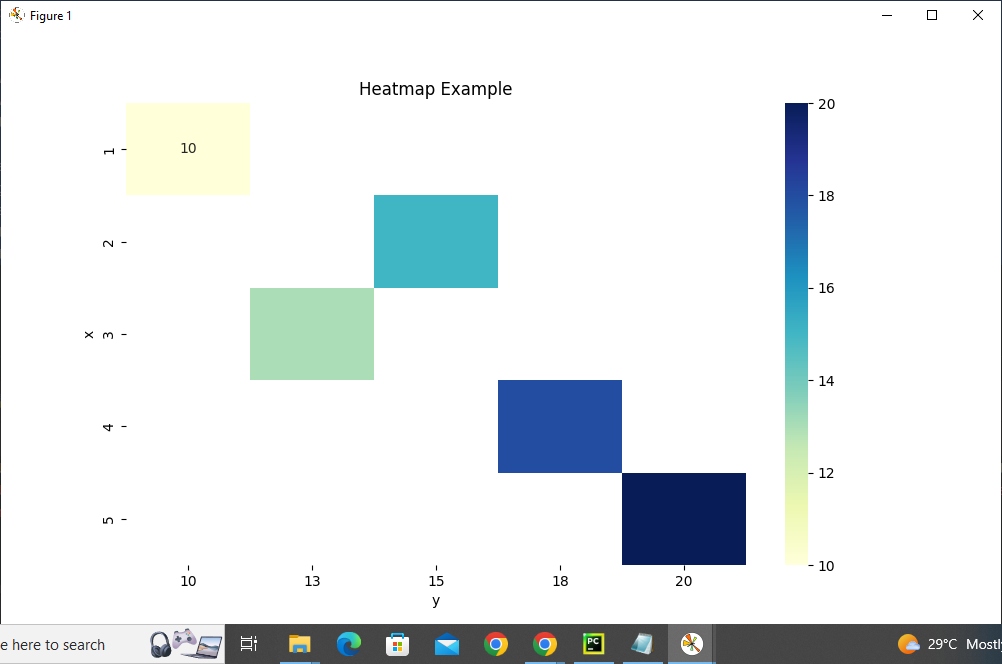
fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, cmap='YlGnBu', cbar=True)

plt.title('Heatmap Example')

plt.show()

**Output:-**

****

**Scatter Plot:-**

x\_values = [1, 2, 3, 4, 5]

data\_values = [10, 15, 13, 18, 20]

df = pd.DataFrame({'x': x\_values, 'value': data\_values})

plt.scatter(df['x'], df['value'], marker='o', color='blue', label='Data Points')

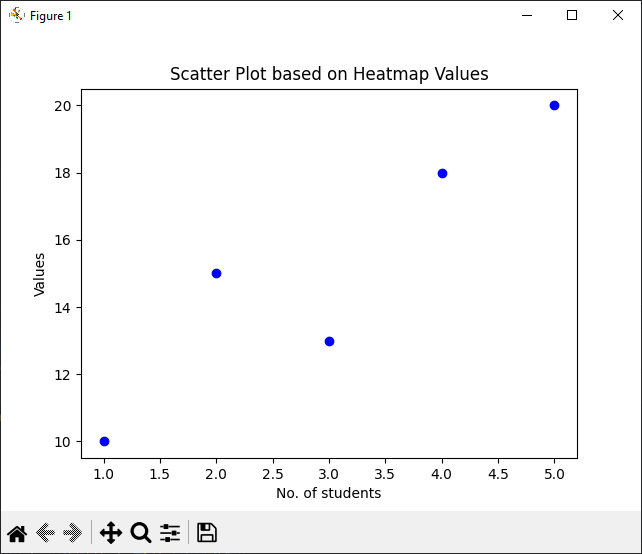
plt.xlabel('No. of students')

plt.ylabel('Values')

plt.title('Scatter Plot based on Heatmap Values')

plt.show()

**Output:-**

****

1. In the heatmap, the values are symmetrically distributed along the x and y axes, forming a diagonal pattern from the bottom-left to the top-right. This indicates that there might be some correlation or relationship between 'x' and 'y' values.

In the scatter plot, there is a clear upward trend in the data points as 'x' increases. This suggests a positive correlation between 'x' and 'value'. As the number of students (x-values) increases, the 'values' tend to increase as well.

The data points are relatively closely clustered around the trendline, indicating a strong correlation. There doesn't appear to be any outliers or significant deviations from the trendline.

1. Based on the scatter plot and the observed trend, it is evident that there is a positive correlation between the two variables ('x' and 'value'). As 'x' increases, 'value' tends to increase as well.
2. The primary pattern or trend identified in the data is a positive linear relationship. As the number of students (x-values) increases, the 'values' also increase, suggesting that there might be a positive impact of increasing the number of students on the 'values'. In summary, the distribution charts and scatter plot reveal that there is a positive correlation between the two variables ('x' and 'value'), and the data follows a clear upward trend. Increasing the number of students is associated with higher 'values'.

**Question 9:**

**Bubble Chart and Density Chart:**

**Objective: Visualize multivariate data using bubble and density charts.**

**Question: Select a dataset with at least three numerical variables (e.g., population, income, and education level by city). Create a bubble chart that represents the data by using bubble sizes and colors to encode information. Additionally, create a density chart (e.g., a 2D density plot) to show the concentration of data points. Answer the following questions:**

1. **How does the bubble chart help in visualizing multivariate data?**
2. **What insights can you gain from the density chart in terms of data concentration?**
3. **Are there any interesting patterns or outliers in the data?**

**Output: Share the bubble chart, density chart, and an interpretation of the visualizations.**

**Density Chart:-**

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

data = np.random.randn(1000)

sns.kdeplot(data, fill=True, color='blue', label='Density Plot')

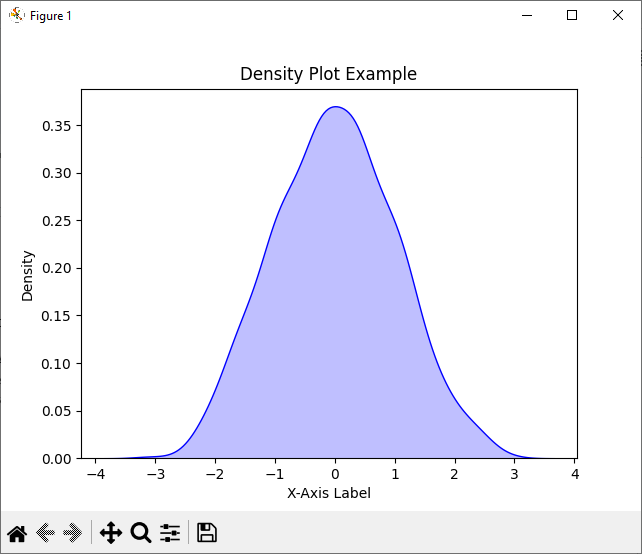
plt.xlabel('X-Axis Label')

plt.ylabel('Density')

plt.title('Density Plot Example')

plt.show()

**Output:-**

****

**Bubble Diagram**

import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]

y = [10, 15, 13, 18, 20]

sizes = [100, 200, 300, 150, 250]

plt.scatter(x, y, s=sizes, alpha=0.5)

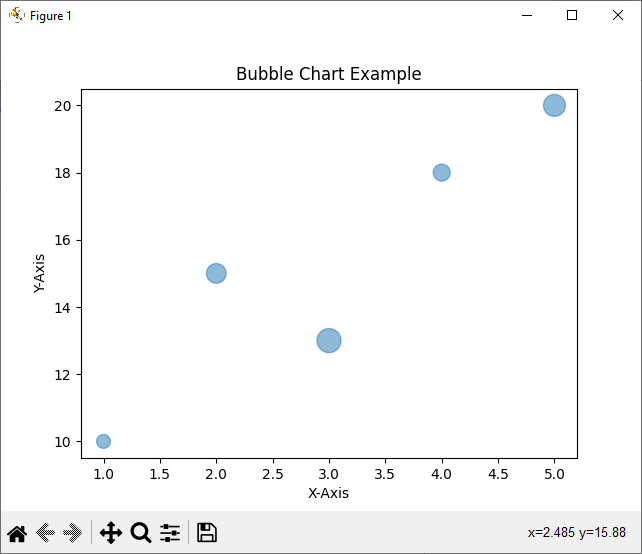
plt.xlabel('X-Axis')

plt.ylabel('Y-Axis')

plt.title('Bubble Chart Example')

plt.show()

Output:-

****

**CO2**

**10.Naive\_Bayes\_Classsification**

import pandas as pd

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

from sklearn.naive\_bayes import GaussianNB

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

data = pd.read\_csv('iris.csv')

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

y = data['species']

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

clf = GaussianNB()

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

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

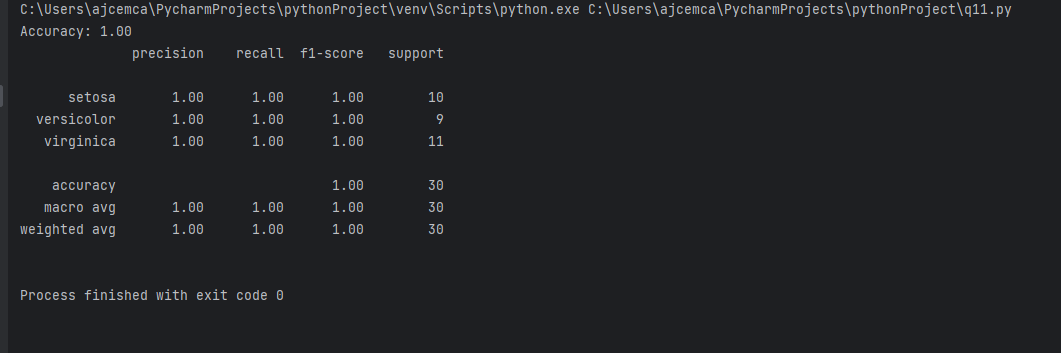
accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy:.2f}')

report = classification\_report(y\_test, y\_pred, target\_names=data['species'].unique())

print(report)

**Output**



**11. KNN-CLASSIFICATION**

import numpy as np

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

from sklearn.neighbors import KNeighborsClassifier

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

# Generate some sample data (replace this with your dataset)

X = np.random.rand(100, 2) # Features

y = np.random.randint(0, 2, 100) # Target labels (binary classification)

# Split the data 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 KNN classifier with k=3

knn = KNeighborsClassifier(n\_neighbors=3)

# Fit the classifier to the training data

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

# Make predictions on the test data

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

# Calculate and print the confusion matrix

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

print("Confusion Matrix:")

print(conf\_matrix)

# Calculate and print the accuracy score

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

print("Accuracy:", accuracy)

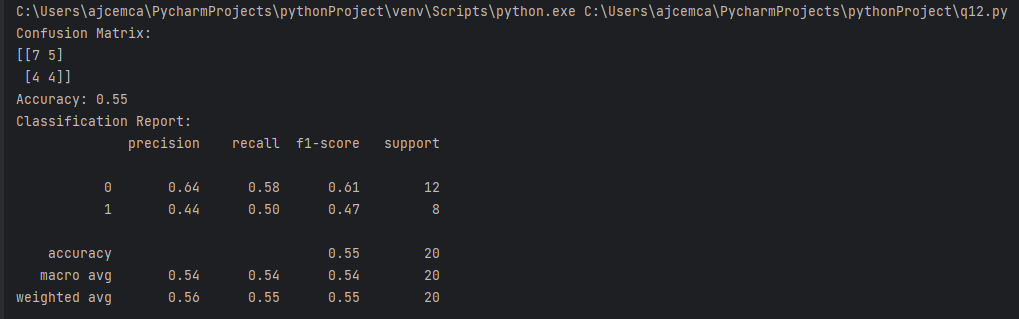
# Print the classification report (including precision, recall, f1-score, and support)

class\_report = classification\_report(y\_test, y\_pred)

print("Classification Report:")

print(class\_report)

**Output**



**12. Linear Regression(without csv)**

import matplotlib.pyplot as plt

import numpy as np

study\_hours = [2, 3, 4, 5, 6]

exam\_scores = [60, 75, 85, 90, 95]

mean\_study\_hours = sum(study\_hours) / len(study\_hours)

mean\_exam\_scores = sum(exam\_scores) / len(exam\_scores)

num = sum((x - mean\_study\_hours) \* (y - mean\_exam\_scores) for x, y in zip(study\_hours, exam\_scores))

deno = sum((x - mean\_study\_hours) \*\* 2 for x in study\_hours)

slope = num / deno

intercept = mean\_exam\_scores - slope \* mean\_study\_hours

def predict\_exam\_score(hours):

return slope \* hours + intercept

hours\_to\_predict = 4

predicted\_score = predict\_exam\_score(hours\_to\_predict)

print(f"Predicted Exam Score for {hours\_to\_predict} hours: {predicted\_score:.2f}")

x\_line = np.array(study\_hours)

y\_line = slope \* x\_line + intercept

plt.scatter(study\_hours, exam\_scores, label='Data Points', color='blue')

plt.plot(x\_line, y\_line, label='Regression Line', color='red')

plt.title('Study Hours vs Exam Score')

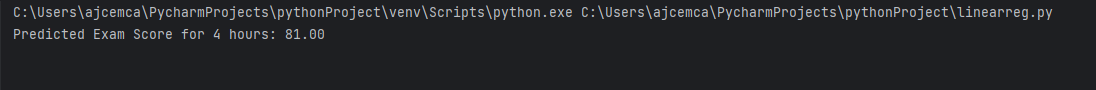
plt.xlabel('Study Hours')

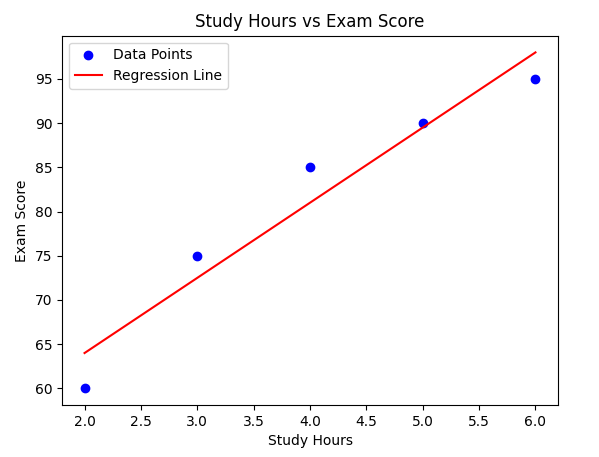
plt.ylabel('Exam Score')

plt.legend()

plt.show()

**Output**





**13. Decision tree classifier(using iris dataset)**

from sklearn.datasets import load\_iris

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

from sklearn.tree import DecisionTreeClassifier

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

import matplotlib.pyplot as plt

from sklearn import tree

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# 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 Decision Tree classifier

clf = DecisionTreeClassifier()

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

# Make predictions on the test set

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

# Calculate the accuracy

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

print(f'Accuracy: {accuracy:.2f}')

# Display the classification report and confusion matrix

print(classification\_report(y\_test, y\_pred))

print(f'Confusion Matrix:\n{confusion\_matrix(y\_test, y\_pred)}')

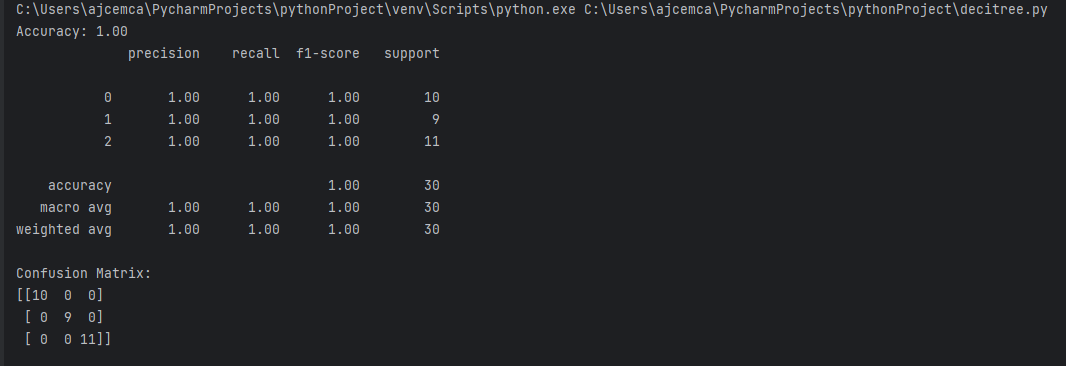
# Plot the decision tree

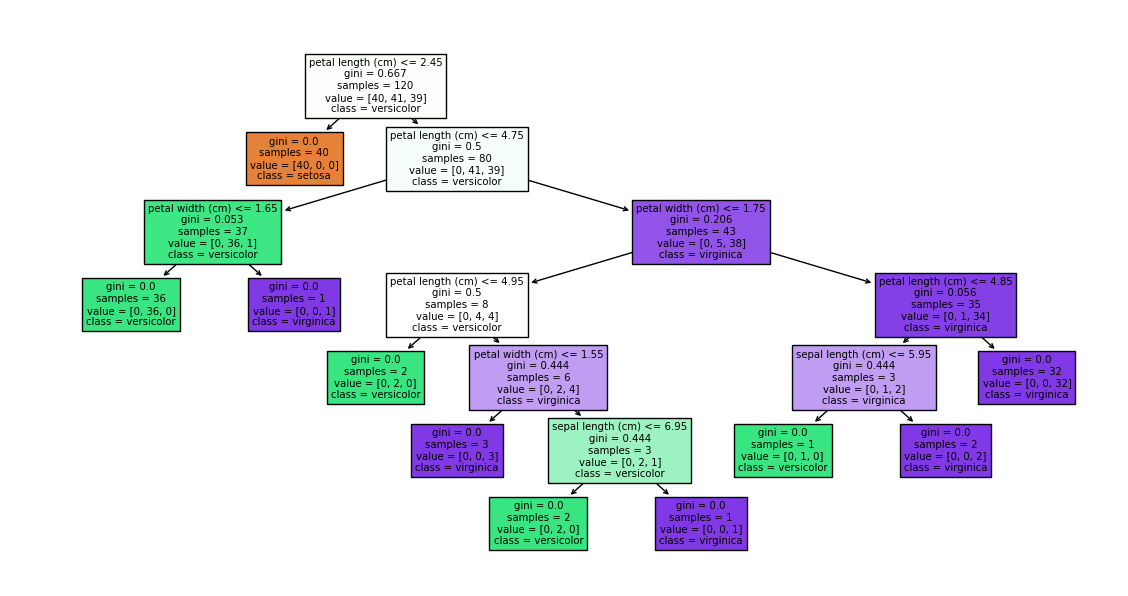
plt.figure(figsize=(12, 8))

tree.plot\_tree(clf, feature\_names=iris.feature\_names, class\_names=iris.target\_names, filled=True)

plt.show()

**Output**





**CO3**

**K-MEANS(without csv)**

import numpy as np

import matplotlib.pyplot as plt

# Generate sample data

data = np.array([[2, 3], [3, 4], [5, 6], [9, 10], [10, 8], [12, 12]])

# Number of clusters (you can change this)

k = 2

# Initialize centroids randomly

centroids = data[np.random.choice(data.shape[0], k, replace=False)]

# Main K-means algorithm

num\_iterations = 100

for iteration in range(num\_iterations):

# Calculate distances between data points and centroids

distances = np.linalg.norm(data[:, np.newaxis] - centroids, axis=2)

# Assign data points to the nearest centroid

labels = np.argmin(distances, axis=1)

print(labels)

# Plot data points and current centroids

plt.scatter(data[:, 0], data[:, 1], c=labels)

plt.scatter(centroids[:, 0], centroids[:, 1], c='red', marker='X', s=200)

plt.xlabel("X-axis")

plt.ylabel("Y-axis")

plt.title(f"Iteration {iteration}")

plt.show()

# Update centroids based on the mean of assigned data points

new\_centroids = np.array([data[labels == i].mean(axis=0) for i in range(k)])

# Check for convergence

if np.array\_equal(centroids, new\_centroids):

break

centroids = new\_centroids

# Print final cluster assignments and centroids

print(distances)

print("Final Cluster Assignments:", labels)

print("Final Cluster Centroids:", centroids)

OUTPUT

