**Name:** Deepanshu Miglani **SAP ID:**500106264  **BTech CSE AIML B4(NH)**

**Lab 1**

**Q1. Array Creation and Manipulation:**

**Create different types of arrays (1D, 2D, 3D) using various methods (np.array, np.arange, np.linspace, etc.).**

**Perform basic operations on arrays (indexing, slicing, reshaping, concatenation).**

**Investigate array attributes like shape, size, dtype, ndim.**

**Use methods like reshape, resize, flatten.**

**CODE:**

import numpy as np

# -------------------------------

# 1. Creating Different Types of Arrays

# -------------------------------

# 1D Array Creation

arr\_1d = np.array([1, 2, 3, 4, 5])

print("1D Array using np.array:\n", arr\_1d)

arr\_arange = np.arange(0, 10, 2) # Start, Stop, Step

print("1D Array using np.arange:\n", arr\_arange)

arr\_linspace = np.linspace(0, 1, 5) # Start, Stop, Number of points

print("1D Array using np.linspace:\n", arr\_linspace)

# 2D Array Creation

arr\_2d = np.array([[1, 2, 3], [4, 5, 6]])

print("\n2D Array using np.array:\n", arr\_2d)

arr\_reshaped = np.arange(1, 10).reshape(3, 3)

print("2D Array using np.reshape:\n", arr\_reshaped)

# 3D Array Creation

arr\_3d = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

print("\n3D Array using np.array:\n", arr\_3d)

# -------------------------------

# 2. Basic Operations on Arrays

# -------------------------------

# Indexing and Slicing

print("\nElement at index 1 in 1D array:", arr\_1d[1])

print("Slicing 2D Array:\n", arr\_2d[:, 1:3]) # Extracts specific columns

# Reshaping, Resizing, and Flattening

reshaped\_array = arr\_1d.reshape(1, -1) # Reshape 1D to 2D as a row vector

print("\nReshaped Array:\n", reshaped\_array)

arr\_resized = np.resize(arr\_1d, (2, 3)) # Resize alters the array's shape in-place

print("Resized Array:\n", arr\_resized)

flattened = arr\_2d.flatten() # Flatten a 2D array into a 1D array

print("Flattened Array:\n", flattened)

# Concatenation

concatenated = np.concatenate((arr\_1d, arr\_arange)) # Concatenating two 1D arrays

print("\nConcatenated Array:\n", concatenated)

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

arr2 = np.array([[5, 6], [7, 8]])

concatenated\_2d = np.concatenate((arr1, arr2), axis=0) # Row-wise concatenation

print("Row-wise Concatenated 2D Array:\n", concatenated\_2d)

# -------------------------------

# 3. Investigate Array Attributes

# -------------------------------

# Array Attributes

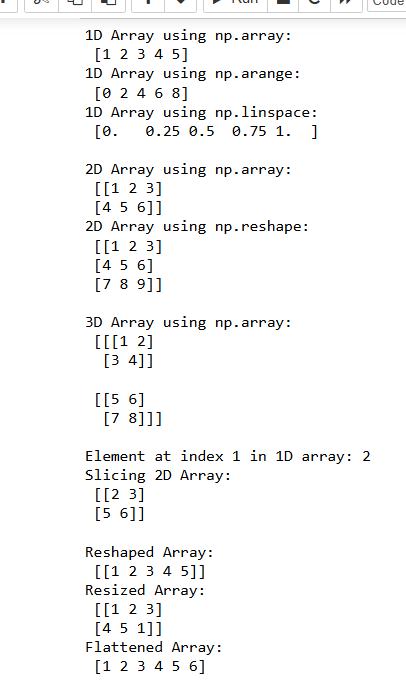
print("\nShape of 2D Array:", arr\_2d.shape)

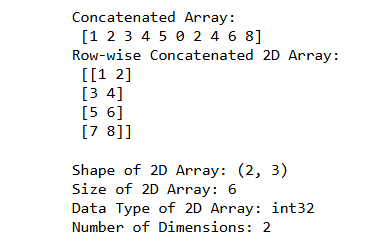
print("Size of 2D Array:", arr\_2d.size)

print("Data Type of 2D Array:", arr\_2d.dtype)

print("Number of Dimensions:", arr\_2d.ndim)

**OUTPUT:**





**Q2. Data Loading and Preprocessing:**

**Load a dataset (e.g., CSV, Excel) using NumPy.**

**Clean and preprocess the data (handling missing values, normalization, standardization).**

**Calculate mean, median, standard deviation, variance, and other statistical measures.**

**CODE:**

import numpy as np

# Suppress all invalid or divide warnings generally, but use specific suppression where needed

np.seterr(invalid='ignore', divide='ignore')

# -------------------------------

# 1. Load a Dataset Using NumPy

# -------------------------------

# Load a CSV file. Adjust the path as needed.

data = np.genfromtxt(r'C:/Users/mehak/Downloads/data.csv', delimiter=',', skip\_header=1)

print("Loaded Data:\n", data)

# -------------------------------

# 2. Clean and Preprocess the Data

# -------------------------------

# Handling Missing Values

# Replace NaN values in each column with the column mean, handling columns that might be entirely NaN

for i in range(data.shape[1]):

# Suppress warnings when calculating the mean of potentially empty columns

with np.errstate(invalid='ignore'):

col\_mean = np.nanmean(data[:, i])

# Check if the column is completely NaN; if so, set a default value (like 0 or another meaningful number)

if np.isnan(col\_mean):

col\_mean = 0 # Replace with 0 or another default value

# Replace NaN values in the column with the computed mean

data[np.isnan(data[:, i]), i] = col\_mean

print("\nData after handling missing values:\n", data)

# Normalization: Scaling values between 0 and 1

# Use np.where to handle division by zero in normalization

normalized\_data = np.where(

np.max(data, axis=0) - np.min(data, axis=0) == 0,

0, # Replace with 0 if division by zero

(data - np.min(data, axis=0)) / (np.max(data, axis=0) - np.min(data, axis=0))

)

print("\nNormalized Data:\n", normalized\_data)

# Standardization: Scaling to zero mean and unit variance

# Use np.where to handle division by zero in standardization

standardized\_data = np.where(

np.std(data, axis=0) == 0,

0, # Replace with 0 if division by zero

(data - np.mean(data, axis=0)) / np.std(data, axis=0)

)

print("\nStandardized Data:\n", standardized\_data)

# -------------------------------

# 3. Calculate Statistical Measures

# -------------------------------

# Mean of each column

mean = np.mean(data, axis=0)

print("\nMean of each column:", mean)

# Median of each column

median = np.median(data, axis=0)

print("Median of each column:", median)

# Standard Deviation of each column

std\_dev = np.std(data, axis=0)

print("Standard Deviation of each column:", std\_dev)

# Variance of each column

variance = np.var(data, axis=0)

print("Variance of each column:", variance)

# Additional Measures: Min, Max, and Sum of each column

min\_values = np.min(data, axis=0)

max\_values = np.max(data, axis=0)

sum\_values = np.sum(data, axis=0)

print("Minimum values of each column:", min\_values)

print("Maximum values of each column:", max\_values)

print("Sum of each column:", sum\_values)

**OUTPUT:**

