Training Day 8 Report

Date: 2 July 2025

Topic: Detailed Exploration of NumPy

Overview:

NumPy (**Numerical Python**) is a **fundamental library for scientific computing in Python**. It provides **multidimensional arrays** and a wide range of mathematical operations to process large datasets efficiently. Unlike normal Python lists, NumPy arrays are **faster, more memory-efficient, and support vectorized operations**, making them ideal for **data science, AI/ML, and numerical simulations**.

Key Concepts Covered

1. NumPy Arrays

- Arrays are created using np.array().
- NumPy arrays can be **1D, 2D, or multi-dimensional**.
- They allow element-wise operations.

Example:

```
import numpy as np
arr1 = np.array([1, 2, 3, 4])
arr2 = np.array([[1, 2, 3], [4, 5, 6]])
print(arr1.ndim)   # Dimension
print(arr2.shape)  # Rows & Columns
print(arr2.size)   # Number of elements
```

2. Array Creation Functions

- np.zeros((m,n)) → Creates an array of zeros.
- np.ones((m,n)) → Creates an array of ones.
- np.arange(start, stop, step) → Creates a sequence of numbers.
- np.linspace(start, stop, num) → Creates evenly spaced numbers.

- np.eye(n) → Identity matrix.

Example:

```
print(np.zeros((3,3)))
print(np.ones((2,4)))
print(np.arange(0,10,2))
print(np.linspace(1,5,5))
print(np.eye(4))
```

3. Indexing and Slicing

NumPy allows powerful indexing and slicing to access specific parts of an array.

```
arr = np.arange(10)
print(arr[2:7])     # Elements from index 2 to 6
print(arr[::-1])    # Reversed array
```

4. Mathematical Operations

- Vectorized operations work on entire arrays without loops.
- Supports operations like +, -, *, /, sqrt, log, sin, etc.

```
a = np.array([10,20,30])
b = np.array([1,2,3])
print(a + b)   # [11 22 33]
print(a * b)   # [10 40 90]
print(np.sqrt(a))
```

5. Aggregate Functions

- np.sum() → Sum of elements
- np.mean() → Average
- np.min(), np.max() → Min & Max values
- np.std() → Standard deviation

Example:

```
arr = np.array([1,2,3,4,5])
print(arr.sum())     # 15
print(arr.mean())    # 3.0
print(arr.max())     # 5
print(arr.std())     # 1.41
```

6. Reshaping and Flattening

- reshape(m,n) → Changes dimensions without changing data.
- flatten() → Converts a multi-dimensional array into 1D.

```
arr = np.arange(12).reshape(3,4)
print(arr)
print(arr.flatten())
```

7. Special Use-Cases

- **Matrix multiplication:** np.dot(a,b) or a @ b
- **Transpose:** arr.T
- **Sorting:** np.sort(arr)
- **Concatenation:** np.concatenate([arr1, arr2])

Summary

On 2 July, we explored **NumPy in detail**:

- Learned about **array creation methods** (zeros, ones, arange, linspace, eye).
- Practiced **indexing, slicing, reshaping, flattening**.
- Applied **mathematical operations** and **aggregate functions**.
- Implemented **matrix operations** like dot product, transpose, and concatenation.

Learning Outcomes

✔ Understood the power of **NumPy arrays vs Python lists**.
✔ Learned multiple **array creation methods** for different use cases.
✔ Practiced **indexing, slicing, and reshaping** effectively.
✔ Gained ability to perform **mathematical and statistical operations** directly on arrays.
✔ Explored **matrix operations** that are useful in **machine learning, AI, and scientific computing**.