# **Learn Numpy & Pandas**

# What is Numpy?

NumPy (Numerical Python) is a powerful library in Python for numerical computing. It provides support for working with large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these data structures efficiently.

**Key Features of NumPy:**

* **N-dimensional array:** Supports efficient storage and manipulation of multi-dimensional arrays.
* **Mathematical functions:** Includes operations like linear algebra, Fourier transform and statistical computations
* **Broadcasting:** Enables performing operations on arrays of different shapes seamlessly
* **Performance Optimization:** Written in C, making computations significantly faster than Python’s built-in list operations
* **Integration:** Works well with other scientific computing libraries like Pandas, SciPy, and Matplotlib.

**Why use NumPy?**

**Speed:** Much faster compared to Python lists when dealing with numerical data.

**Memory Efficiency:** Uses less memory than Python lists for storing data.

**Ease of Computation:** Allows vectorized operations, eliminating the need for explicit loops.

**Interoperability:** Compatible with various data formats and scientific computing frameworks.

## **Installation of NumPy**

Installing NumPy is straightforward and can be done using pip or conda, depending on your preference. Here’s a step-by-step guide for Windows, Linux, and macOS:

### **Windows Installation:**

1. Open Command Prompt
2. Check Python Installation

python – version

If Python is not installed, download and install it from python.org

1. Install NumPy using pip

pip install numpy

1. Verify Installation

Python -c “import numpy; print(numpy.\_\_version\_\_)”

### **Linux Installation**

1. Open Terminal
2. Check Python Installation

python3 –version

If Python is missing, install it using

sudo apt install python3

1. Install pip (if not installed)

sudo apt install python3-pip

1. Install Numpy

pip3 install numpy

1. Verify Installation

Python3 -c “import numpy; print(numpy.\_\_version\_\_)”

### **macOS**

1. Open Terminal
2. Check Python Installation

python3 – version

If Python is missing, install it using

brew install python

1. Install pip (if not installed)

python3 -m ensurepip

1. Install Numpy

pip install numpy

1. Verify installation

python3 -c “import numpy; print(numpy.\_\_version\_\_)”

Alternatively, you can install NumPy using conda

conda install numpy

This method is useful if you are using Anaconda or Miniconda

## **Creating and Manipulating Arrays**

### **1D Array (One-Dimensional)**

**A 1D Array is simply a list-like array with single dimension.**

**Creation:**

import numpy as np

# Creating a 1D array

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

print(arr\_id)

**Manipulation of Array:**

**Access Elements:** arr\_id[2]

**Slicing:** arr\_id[1:3]

**Mathematical Operations:** arr\_id \* 2

### **2D Array (Two-Dimensional)**

A 2D Array represents rows and column, like a table or matrix

**Creation:**

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

print(arr\_2d)

**Manipulation of Array:**

**Access Elements:** arr\_2d[1,2]

**Row slicing:** arr\_2d[0,:]

**Column slicing:** arr\_2d[:,-1]

**Matrix Operations:** np.dot(arr\_2d, np.array([[1], [2], [3]]))

### **3D Array (Three-Dimensional)**

A **3D array** adds depth, like multiple 2D arrays stacked.

**Creation:**

python

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

print(arr\_3d)

**Manipulation:**

* **Access elements:** arr\_3d[1, 0, 1] → Retrieves 6.
* **Slicing:** arr\_3d[:, :, 0] → Extracts all first elements from each depth layer.
* **Reshape:** arr\_3d.reshape(4, 2) → Changes shape while preserving data.

### **4D Array (Four-Dimensional)**

A **4D array** is useful for representing complex datasets like images or time-series data.

**Creation:**

python

arr\_4d = np.random.randint(1, 10, (2, 2, 2, 2)) # Creates a random 4D array

print(arr\_4d)

**Manipulation:**

* **Access elements:** arr\_4d[1, 0, 1, 0] → Retrieves a specific number.
* **Transpose:** arr\_4d.transpose(1, 0, 2, 3) → Rearranges dimensions.
* **Reshape:** arr\_4d.reshape(8, 2) → Flattens some dimensions while keeping structure.

**Summary**

* **1D**: Simple list-like array.
* **2D**: Table/matrix-like format.
* **3D**: Stack of 2D arrays.
* **4D**: Complex structured data, useful for advanced computing.