NumPy (short for Numerical Python) is a powerful Python library used for numerical computations, especially when working with large arrays, matrices, or multi-dimensional data. It forms the foundation of many data science, machine learning, and scientific computing tools in Python.

**How to install NUMPY**

Open VS Code🡪Create new file give any name to file(numpy.py)

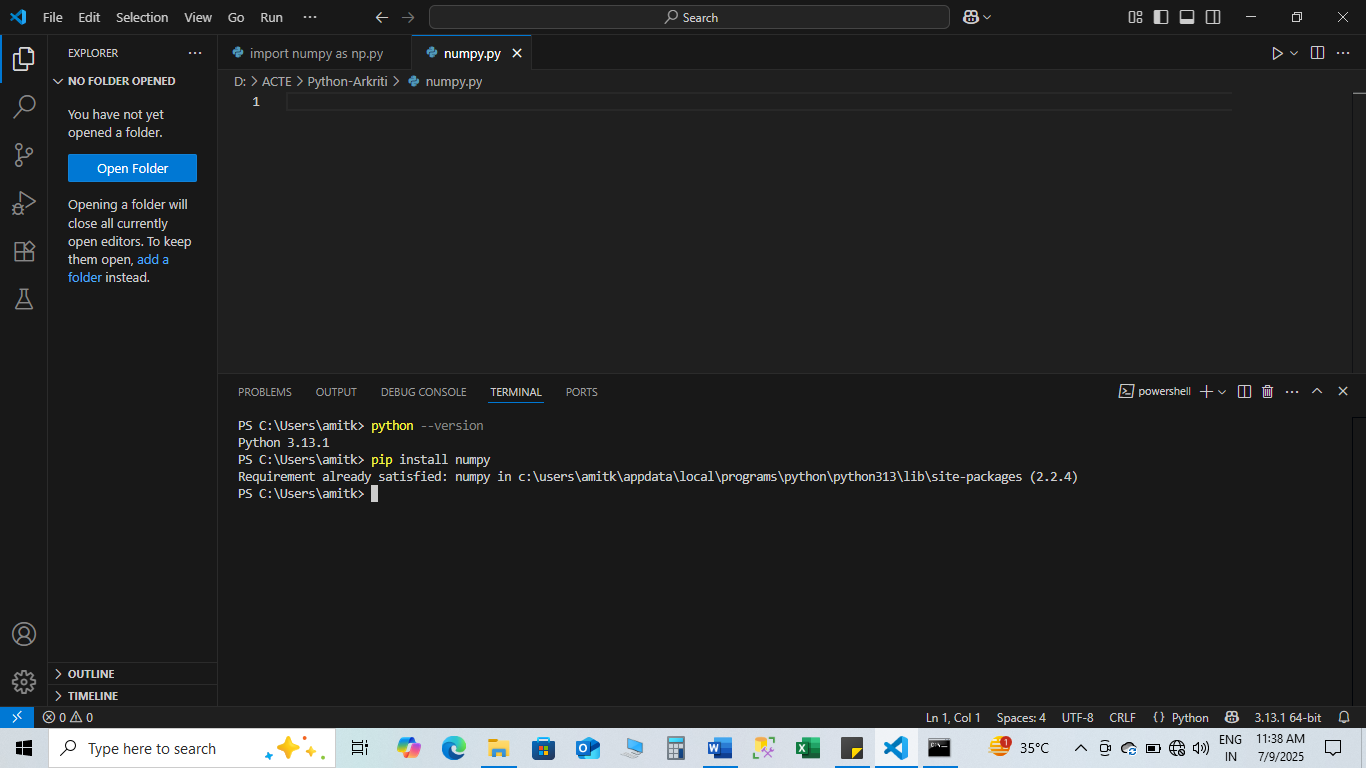
Go to Terminal🡪Go to New Terminal

PS C:\Users\amitk> python --version

Python 3.13.1

PS C:\Users\amitk> pip install numpy (press Enter)

It will automatically install numpy.



**Array Manipulation using NumPy**

NumPy Basics for Data Analysis

1. **Importing NumPy**

import numpy as np

1. **Creating Arrays**

1D, 2D, 3D Arrays:

a = np.array([1, 2, 3]) # 1D array

b = np.array([[1, 2], [3, 4]]) # 2D array

c = np.array([[[1], [2]], [[3], [4]]]) # 3D array

Special arrays:

1. np.zeros((2, 3)) # 2x3 array of zeros

import numpy as np

# Create a 2x3 array filled with zeros

arr = np.zeros((2, 3))

print("2x3 array of zeros:")

print(arr)

1. np.ones((2, 3)) # 2x3 array of ones
2. np.eye(3) # Identity matrix
3. np.arange(10) # [0 1 2 3 ... 9]
4. np.linspace(0, 1, 5) # 5 values from 0 to 1
5. **Array Attributes**

a = np.array([[1, 2, 3], [4, 5, 6]])

print(a.shape) # (2, 3)

print(a.ndim) # 2 (number of dimensions)

print(a.size) # 6 (total elements)

print(a.dtype) # data type, e.g., int64

**4. Array Indexing and Slicing**

**Accessing elements:**

a[0, 1] # row 0, column 1

Slicing:

a[:, 1] # All rows, column 1

a[1, :] # Row 1, all columns

a[0:2, 1:3] # Subarray

**5.Array Operations**

1.Element-wise operations:

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

b = np.array([4, 5, 6])

a + b # [5 7 9]

a \* b # [4 10 18]

a \*\* 2 # [1 4 9]

2.Aggregate functions:

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

np.sum(a) # 10

np.mean(a) # 2.5

np.max(a) # 4

np.min(a) # 1

np.std(a) # Standard deviation

3.Reshaping and Flattening

a = np.array([[1, 2, 3], [4, 5, 6]])

a.reshape((3, 2)) # Change shape

a.flatten() # Convert to 1D

4.Filtering and Boolean Indexing

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

a[a > 3] # [4 5]

5.Axis in Aggregations

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

np.sum(a, axis=0) # Sum columns → [4 6]

np.sum(a, axis=1) # Sum rows → [3 7]

6.Stacking Arrays

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

b = np.array([[5, 6]])

np.vstack((a, b)) # Vertical stack

a = np.array([[1, 2]])

b = np.array([[5, 6]])

np.hstack((a, b)) # Horizontal stack

1. **Matrix product**

In Python, you can perform **matrix product** (also called **matrix multiplication**) using:

**1. Using @ Operator (Python 3.5+)**

import numpy as np

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

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

result = A @ B

print(result) #[[1\*5 + 2\*7, 1\*6 + 2\*8],

#[3\*5 + 4\*7, 3\*6 + 4\*8]]

**2. Using np.dot()**

import numpy as np

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

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

result = np.dot(A, B)

print(result)

1. **Using np.matmul()**

import numpy as np

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

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

result = np.matmul(A, B)

print(result)

1. **Array manipulation**

**Array manipulation** in Python typically involves operations like creating, accessing, updating, reshaping, sorting, and performing arithmetic or logical operations on arrays.

While you can use native Python lists for basic manipulation, the **NumPy** library is most commonly used for efficient array operations.

1. Using Python Lists (Basic)

# Create an array (list)

arr = [10, 20, 30, 40]

# Access elements

print(arr[1]) # 20

# Update element

arr[2] = 99

print(arr[0:4])

# Append a value

arr.append(50)

print(arr[0:5])

# Remove an element

arr.remove(10)

print(arr[0:5])

# Slice the array

print(arr[1:3])

2. Using NumPy (Recommended for Efficiency)

import numpy as np

# Create a NumPy array

arr = np.array([10, 20, 30, 40])

# Access elements

print(arr[2]) # 30

# Update value

arr[1] = 99

# Add arrays

arr2 = np.array([1, 2, 3, 4])

result = arr + arr2 # Element-wise addition

# Reshape array

reshaped = arr.reshape(2, 2)

# Filter elements

filtered = arr[arr > 20]

print("Original:", arr)

print("Result:", result)

print("Reshaped:\n", reshaped)

print("Filtered:", filtered)

3. Common NumPy Functions for Manipulation

| Function | Purpose |
| --- | --- |
| np.reshape() | Change shape of array |
| np.concatenate() | Join arrays |
| np.split() | Split arrays |
| np.sort() | Sort array |
| np.unique() | Get unique values |
| np.delete() | Delete elements |
| np.insert() | Insert elements |
| np.append() | Append elements |

1.)np.concatenate()

import numpy as np

# Create two NumPy arrays

arr1 = np.array([10, 20, 30])

arr2 = np.array([40, 50, 60])

# Concatenate them

result = np.concatenate((arr1, arr2))

print("Concatenated Array:", result)

2.)np.split()

import numpy as np

# Create a NumPy array

arr = np.array([10, 20, 30, 40, 50, 60])

# Split into 3 equal parts

result = np.split(arr, 3)

print(result)

3.)np.sort()

import numpy as np

arr = np.array([30, 10, 50, 20, 40])

sorted\_arr = np.sort(arr)

print("Original Array:", arr)

print("Sorted Array:", sorted\_arr)

4.)np.unique()

import numpy as np

arr = np.array([10, 20, 10, 30, 20, 40, 30])

unique\_values = np.unique(arr)

print("Original Array:", arr)

print("Unique Values:", unique\_values)

5.)np.delete()

import numpy as np

arr = np.array([10, 20, 30, 40, 50])

# Delete the element at index 2 (which is 30)

new\_arr = np.delete(arr, 2)

print("Original Array:", arr)

print("After Deletion:", new\_arr)

6.)np.insert()

import numpy as np

arr = np.array([10, 20, 30, 40])

# Insert value 25 at index 2

new\_arr = np.insert(arr, 2, 25)

print("Original Array:", arr)

print("After Insertion:", new\_arr)

**File handling using NumPy**

File handling using NumPy in Python typically involves reading from and writing to files — especially for numerical data — using NumPy’s built-in functions such as:

* np.loadtxt()
* np.savetxt()
* np.genfromtxt()
* np.save() / np.load() (for binary .npy files)
* np.savez() / np.load() (for compressed .npz files)

**✅ 1. np.loadtxt() – Reading from a Text File**

import numpy as np

# Reading data from a text file

data = np.loadtxt('data.txt')

print(data)

**Example content of data.txt:**

**1.0 2.0 3.0**

**4.0 5.0 6.0**

**✅ 2. np.savetxt() – Writing to a Text File**

import numpy as np

arr = np.array([[10, 20, 30], [40, 50, 60]])

np.savetxt('output.txt', arr, fmt='%d')

**This creates a file output.txt like:**

**10 20 30**

**40 50 60**

**✅ 3. np.genfromtxt() – Handling Missing Data**

data = np.genfromtxt('data\_with\_missing.txt', delimiter=',', filling\_values=0)

print(data)

**File (data\_with\_missing.txt):**

**1,2,3**

**4,,6**

**7,8,9**

**✅ 4. np.save() and np.load() – For Binary .npy Files**

arr = np.array([100, 200, 300])

np.save('array\_data.npy', arr)

loaded\_arr = np.load('array\_data.npy')

print(loaded\_arr)

**✅ 5. np.savez() and np.load() – For Multiple Arrays (Compressed)**

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

b = np.array([4, 5, 6])

np.savez('data.npz', arr1=a, arr2=b)

loaded = np.load('data.npz')

print(loaded['arr1'])

print(loaded['arr2'])

**Summary Table**

| **Function** | **Purpose** | **File Type** |
| --- | --- | --- |
| **np.loadtxt()** | **Load text files** | **.txt, .csv** |
| **np.savetxt()** | **Save arrays to text files** | **.txt, .csv** |
| **np.genfromtxt()** | **Load text files with missing data** | **.txt, .csv** |
| **np.save()** | **Save binary array (efficient)** | **.npy** |
| **np.load()** | **Load .npy or .npz files** | **.npy, .npz** |
| **np.savez()** | **Save multiple arrays in one file** | **.npz** |