# NumPy: Advanced Array Operations

IIT Ropar - Minor in AI

February 21, 2025

## 1 Weather Data Analysis Using NumPy

A meteorological department collects daily temperature readings across 10 cities for a month. The department wants to analyze trends, including average temperatures, hottest and coldest days, and temperature fluctuations. Using NumPy, perform operations such as indexing, aggregation, reshaping, and boolean filtering to extract meaningful insights.

NumPy is a powerful library for numerical computing in Python. In this guide, we'll explore advanced array operations that are crucial for data manipulation in machine learning tasks.

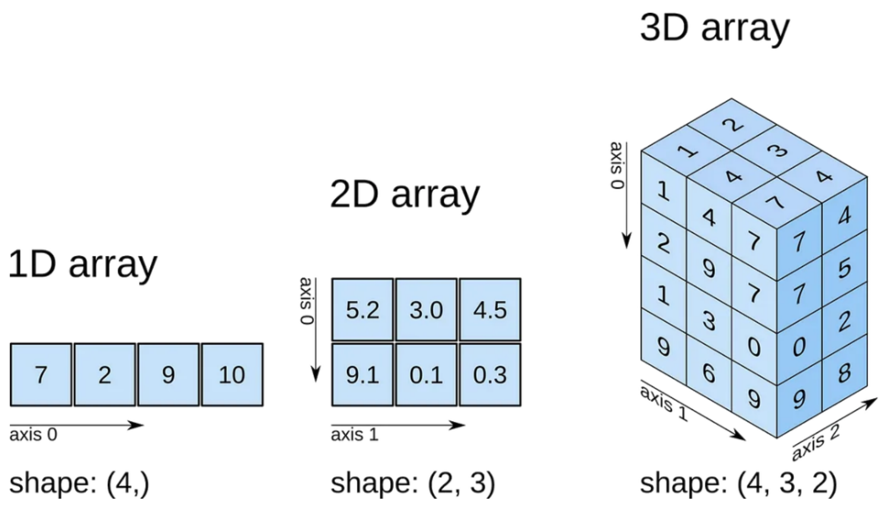

Figure 1: Illustration of 1D, 2D, and 3D NumPy arrays

## 2 Creating and Manipulating NumPy Arrays

Let's start by creating a simple NumPy array:

```
import numpy as np
np1 = np.array([3,4,5,6,7,8])
print(np1)
# Ouput:
# [3 4 5 6 7 8]
```

This creates a 1-dimensional array with 6 elements.

## 3 Array Indexing and Slicing

NumPy provides powerful indexing capabilities:

### 3.1 Positive Indexing

You can use positive indices to access elements from the start of the array:

```
print(np1[3])
# Output: 6
```

This returns the 4th element(i.e. element at 3rd index) from the start of the array.

### 3.2 Negative Indexing

You can use negative indices to access elements from the end of the array:

```
print(np1[-3]) # Output: 6
```

This returns the third element(i.e. element at -3 index) from the end of the array.

### 3.3 Array Slicing

Slicing allows you to extract a portion of the array:

```
print(np1[0:-1])
# Output: [3 4 5 6 7]
print(np1[2:])
# Output: [5 6 7 8]
print(np1[:5])
# Output: [3 4 5 6 7]
print(np1[:])
# Output: [3 4 5 6 7 8]
```

These operations demonstrate various ways to slice the array, including from the start, to the end, and with specific ranges.

### 3.4 Using Step Size while Array Slicing

You can also specify a step size when slicing:

```
print(np1[0:-2:2])
# Output: [3 5]
print(np1[::2])
# Output: [3 5 7]
```

This allows you to select every nth element of the array.

## 4 Multidimensional Arrays

NumPy excels at handling multidimensional arrays:

```
np2 = np.array([ [1,2,3,4,5,6,7], [8,9,10,11,12,13,14] ])
print(np2)
# Ouput:
# [[ 1 2 3 4 5 6 7]
# [ 8 9 10 11 12 13 14]]
```

This creates a 2-dimensional array with 2 rows and 7 columns1.

### 4.1 Slicing 2D Arrays

You can slice 2D arrays using comma-separated indices:

```
print(np2[:,2:5])
# Output:
# [[ 3 4 5]
# [10 11 12]]
```

This selects all rows and columns 2 through 4.

## 5 Modifying Array Elements

NumPy allows for easy modification of array elements:

```
np2[1,4] = 20
print(np2)
# Output:
# [[ 1 2 3 4 5 6 7]
# [ 8 9 10 11 20 13 14]]

np2[:, 1] = [20,30]
print(np2)
# Output:
# [[ 1 20 3 4 5 6 7]
# [ 8 30 10 11 20 13 14]]

np2[0, :] = [1,1,1,1,1,1,1]
print(np2)
# Output:
# [[ 1 1 1 1 1 1 1]
# [ 8 30 10 11 20 13 14]]
```

These operations demonstrate how to modify single elements, columns, and rows of a 2D array.

## 6 Creating Special Arrays

NumPy provides functions to create arrays with specific properties:

```
outer = np.ones((5,5))
print(outer)
# Output:
# [[1. 1. 1. 1. 1.]
# [1. 1. 1. 1. 1.]
# [1. 1. 1. 1. 1.]
# [1. 1. 1. 1. 1.]
# [1. 1. 1. 1. 1.]]

inner = np.zeros((3,3))
print(inner)
# Output:
# [[0. 0. 0.]
# [0. 0. 0.]
# [0. 0. 0.]]

inner[1,1] = 9
print(inner)
# Output:
# [[0. 0. 0.]
# [0. 9. 0.]
# [0. 0. 0.]]

outer[1:4, 1:4] = inner
print(outer)
# Output:
# [[1. 1. 1. 1. 1.]
# [1. 0. 0. 0. 1.]
# [1. 0. 9. 0. 1.]
# [1. 0. 0. 0. 1.]
# [1. 1. 1. 1. 1.]]
```

This creates a 5x5 array of ones and inserts a 3x3 array of zeros (with a 9 in the center) into numpy array named outer.

## 7 Reshaping Arrays

Reshaping allows you to change the dimensions of an array:

```
np2 = np.array([1,2,3,4,5,6,7,8])
np_2d = np2.reshape(2,2,2)
print(np_2d)
# Outer:
# [[[1 2]
# [3 4]]
#
# [[5 6]
# [7 8]]]

np_2d = np2.reshape(2,-1,2)
print(np_2d)
# Output:
# [[[1 2]
# [3 4]]
#
# [[5 6]
# [7 8]]]
```

np2.reshape(2,2,2) reshapes the 1D array into a 3D array with 2 ”layers”, each containing 2 rows and 2 columns.
np2.reshape(2,-1,2) does the same thing as above, it reshapes the 1D array, in such a way that outermost layer has 2 elements and innermost layer has 2 elements, but in this case the middle layer's no. of elements were figured out by Numpy itself(since -1 written) rather than given by us.

## 8 Copying Arrays

When working with arrays, it's important to understand the difference between views and copies:

```
a = np.array([1,2,3,4,5])
b = a.copy()
b[1] = 100

print(b)
# Output: [ 1 100 3 4 5]
```

```
print(a)
# Output: [1 2 3 4 5]

c = a
c[1] = 200
print(c)
# Output: [ 1 200 3 4 5]
print(a)
# Output: [ 1 200 3 4 5]
```

This demonstrates that modifying a copy of an array (i.e. b in this case) doesn't affect the original array.
Whereas, modifying view of an array (i.e. c in this case) does affect the original array.

## 9 Conclusion

These advanced NumPy operations form the foundation for efficient data manipulation in machine learning tasks. By mastering these techniques, you'll be well-equipped to handle complex numerical computations and data pre-processing for your machine learning models.