Numpy Essentials For Data Science

Saurab Gyawali

November 26, 2025

Contents

1 Introduction **2**
1.1 What is NumPy? . . . 2
1.2 Installation and Setup . . . 2

2 Array Fundamentals **2**
2.1 Dimensions and Shape . . . 2
2.2 Creating Arrays . . . 3
2.2.1 Manual Creation . . . 3
2.2.2 Special Arrays . . . 3
2.2.3 Sequential Arrays . . . 3
2.2.4 Random Arrays . . . 3

3 Data Types and Type Casting **4**
3.1 NumPy Data Types . . . 4
3.2 Type Specification and Casting . . . 4

4 Array Manipulation **4**
4.1 Reshaping Arrays . . . 4
4.2 Flattening Arrays . . . 4
4.3 Transposing Arrays . . . 5

5 Indexing and Slicing **5**
5.1 Basic Indexing . . . 5
5.2 Slicing . . . 5
5.3 Advanced Indexing . . . 6
5.3.1 Boolean Indexing . . . 6

6 Array Operations **6**
6.1 Arithmetic Operations . . . 6
6.2 Broadcasting . . . 6
6.3 Aggregate Functions . . . 6

7 Practical Applications **7**
7.1 Linear Algebra . . . 7

1 Introduction

1.1 What is NumPy?

NumPy (Numerical Python) is the fundamental package for scientific computing in Python. It provides:

A powerful N-dimensional array object

Sophisticated (broadcasting) functions

Useful linear algebra, Fourier transform, and random number capabilities

NumPy arrays are more efficient than Python lists for numerical operations due to:

Memory efficiency: Homogeneous data types allow compact storage

Vectorization: Operations applied to entire arrays without explicit loops

Optimized C implementation: Core operations executed in compiled code

1.2 Installation and Setup

Most scientific Python distributions include NumPy by default. To install manually:

```
# Using pip
pip install numpy

# Using conda
conda install numpy
```

Basic import convention:

```
import numpy as np  # Standard alias
```

2 Array Fundamentals

2.1 Dimensions and Shape

NumPy supports arrays of various dimensions:

0D Array (Scalar): Single value (e.g., 5, 2.3)

1D Array (Vector): Sequence of values (e.g., [1, 2, 3])

2D Array (Matrix): Grid of values (e.g., [[1, 2], [3, 4]])

3D+ Array (Tensor): Multi-dimensional structure

Key properties of arrays:

`ndim`: Number of dimensions

`shape`: Tuple indicating size along each dimension

`size`: Total number of elements

`dtype`: Data type of elements

```python
# Array dimension examples
import numpy as np
scalar = np.array(5)  # 0D
vector = np.array([1, 2, 3])  # 1D
matrix = np.array([[1, 2, 3], [4, 5, 6]])  # 2D

print(f"Scalar dimensions: {scalar.ndim}, shape: {scalar.shape}")
print(f"Vector dimensions: {vector.ndim}, shape: {vector.shape}")
print(f"Matrix dimensions: {matrix.ndim}, shape: {matrix.shape}")
```

2.2 Creating Arrays

2.2.1 Manual Creation

```python
# From Python lists/tuples
np.array([1, 2, 3])  # 1D array
np.array([[1, 2], [3, 4]])  # 2D array
```

2.2.2 Special Arrays

```python
# Zero arrays
np.zeros(5)  # 1D array of zeros
np.zeros((2, 3))  # 2x3 array of zeros

# One arrays
np.ones(4)  # 1D array of ones
np.ones((3, 2), dtype=float)  # 3x2 array of float ones

# Identity matrix
np.eye(3)  # 3x3 identity matrix
```

2.2.3 Sequential Arrays

```python
# Arithmetic sequences
np.arange(0, 10)  # [0, 1, 2, ..., 9]
np.arange(0, 10, 2)  # [0, 2, 4, 6, 8]

# Linear spacing
np.linspace(0, 1, 5)  # [0.0, 0.25, 0.5, 0.75, 1.0]

# Logarithmic spacing
np.logspace(0, 2, 3)  # [1.0, 10.0, 100.0] (10^0, 10^1, 10^2)
```

2.2.4 Random Arrays

```python
# Uniform distribution [0, 1)
np.random.rand(3)  # 1D array of 3 random values
np.random.rand(2, 3)  # 2x3 array of random values
```

```
4
5 # Standard normal distribution (mean=0, std=1)
6 np.random.randn(2, 2)  # 2x2 array of standard normal values
7
8 # Random integers
9 np.random.randint(0, 10, size=5)  # 5 random integers in [0, 9]
```

3 Data Types and Type Casting

3.1 NumPy Data Types

NumPy provides specific data types for memory optimization:

Type	Description	Example
`int32`, `int64`	Signed integers	-128 to 127 (`int8`)
`float32`, `float64`	Floating point numbers	3.14159 (`float64`)
`bool`	Boolean	True, False
`complex64`, `complex128`	Complex numbers	1+2j

Table 1: Common NumPy Data Types

3.2 Type Specification and Casting

```
1 # Specifying dtype during creation
2 arr_int = np.array([1, 2, 3], dtype=np.int32)
3 arr_float = np.array([1, 2, 3], dtype=np.float64)
4
5 # Type casting with astype()
6 arr = np.array([1.7, 2.3, 3.9])
7 arr_int = arr.astype(np.int32)  # Truncates to [1, 2, 3]
```

4 Array Manipulation

4.1 Reshaping Arrays

```
1 arr = np.arange(12)  # [0, 1, 2, ..., 11]
2
3 # Reshape to different dimensions
4 reshaped_2d = arr.reshape(3, 4)  # 3x4 array
5
6 # -1 lets NumPy figure out the dimension
7 auto_reshape = arr.reshape(4, -1)  # 4x3 array
```

4.2 Flattening Arrays

```python
matrix = np.array([[1, 2, 3], [4, 5, 6]])

# Different flattening methods
ravel_view = matrix.ravel()  # Returns a view when possible
flatten_copy = matrix.flatten()  # Always returns a copy

print(f"ravel(): {ravel_view}")  # [1, 2, 3, 4, 5, 6]

# Modifying ravel_view affects matrix if it's a view
ravel_view[0] = 99
print(f"After modification:\n{matrix}")
```

4.3 Transposing Arrays

```python
matrix = np.array([[1, 2, 3], [4, 5, 6]])

# Transpose methods
transposed = matrix.T  # Preferred method

print(f"Original:\n{matrix}")
print(f"Transposed:\n{transposed}")
```

5 Indexing and Slicing

5.1 Basic Indexing

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

# Zero-based indexing
print(arr[0])  # 10
print(arr[-1])  # 50 (last element)

# 2D array indexing
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(matrix[0, 1])  # 2 (row 0, column 1)
```

5.2 Slicing

```python
arr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

# Basic slicing: start:stop:step
print(arr[2:6])  # [2, 3, 4, 5] (stop index excluded)
print(arr[:5])  # [0, 1, 2, 3, 4] (from start to index 4)
print(arr[::2])  # [0, 2, 4, 6, 8] (every other element)

# 2D slicing
matrix = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print(matrix[0:2, 1:3])  # [[2, 3], [6, 7]] (rows 0-1, columns 1-2)
```

5.3 Advanced Indexing

5.3.1 Boolean Indexing

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# Create boolean mask
mask = arr > 5
print(mask)  # [False, False, False, False, False, True, True, ...]
print(arr[mask])  # [6, 7, 8, 9, 10]

# Multiple conditions
mask2 = (arr % 2 == 0) & (arr > 3)  # Even numbers greater than 3
print(arr[mask2])  # [4, 6, 8, 10]
```

6 Array Operations

6.1 Arithmetic Operations

NumPy supports element-wise arithmetic operations:

```
a = np.array([1, 2, 3, 4])
b = np.array([5, 6, 7, 8])

# Basic arithmetic
add = a + b  # [6, 8, 10, 12]
sub = a - b  # [-4, -4, -4, -4]
mul = a * b  # [5, 12, 21, 32]
div = a / b  # [0.2, 0.333..., 0.428..., 0.5]
```

6.2 Broadcasting

Broadcasting allows operations between arrays of different shapes:

```
# Example: Matrix + Vector
matrix = np.array([[1, 2, 3], [4, 5, 6]])  # Shape (2, 3)
vector = np.array([10, 20, 30])  # Shape (3,)
result = matrix + vector  # Shape (2, 3)
# [[11, 22, 33],
#  [14, 25, 36]]
```

6.3 Aggregate Functions

Functions that reduce array dimensions by computing summary statistics:

```
arr = np.array([1, 2, 3, 4, 5])
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Basic aggregates
print(f"Sum: {np.sum(arr)}")  # 15
print(f"Mean: {np.mean(arr)}")  # 3.0
print(f"Standard deviation: {np.std(arr)}")  # 1.414
```

```
print(f"Minimum: {np.min(arr)}")  # 1
print(f"Maximum: {np.max(arr)}")  # 5

# 2D aggregates (with axis parameter)
print(f"Sum along rows: {np.sum(matrix, axis=0)}")  # [12, 15, 18]
print(f"Sum along columns: {np.sum(matrix, axis=1)}")  # [6, 15, 24]
```

7 Practical Applications

7.1 Linear Algebra

NumPy provides comprehensive linear algebra capabilities:

```
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])

# Matrix multiplication
dot_product = np.dot(a, b)
matmul = a @ b

# Matrix properties
det = np.linalg.det(a)  # Determinant
```

Credits and Resources

This NumPy reference was created by **Saurab Gyawali**. For more projects and tutorials:

GitHub: `https://github.com/soorabcode/`

Instagram: `https://www.instagram.com/saurab_gyawalii`

LinkedIn: `https://www.linkedin.com/in/saurab-gyawali-420897233`