Introducción práctica a NumPy: arrays y operaciones básicas

Ciencia de datos con NumPy

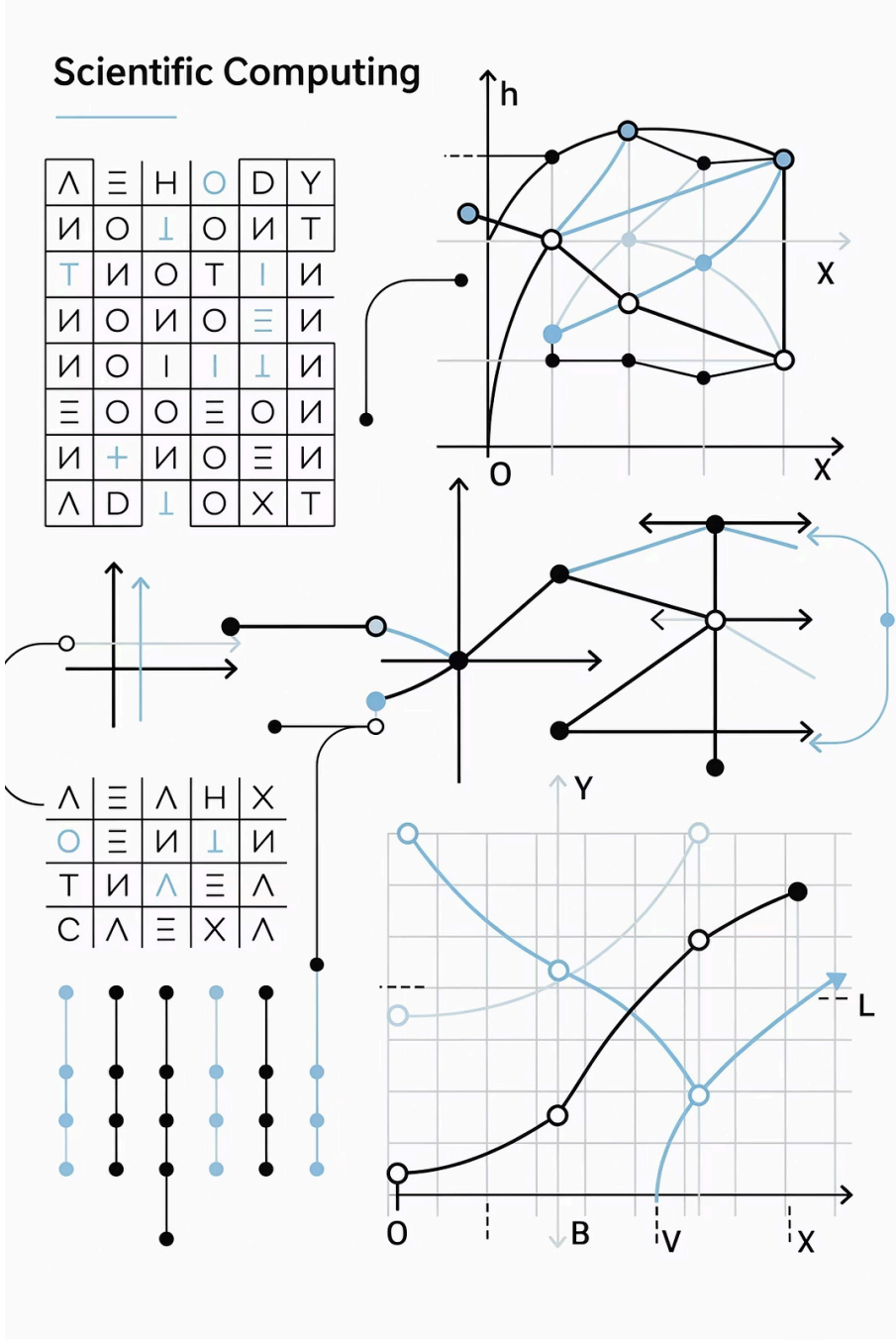
CONCEPTOS BÁSICOS

¿Qué es NumPy y para qué sirve?

Librería para cálculos numéricos

Más rápida que las listas de Python

Base de la ciencia de datos en Python



Listas vs Arrays: una comparación visual

Listas de Python

- Pueden contener tipos mixtos
- Flexibles pero lentas
- Operaciones elemento por elemento requieren bucles

Arrays de NumPy

- Todos los elementos del mismo tipo
- Extremadamente rápidos
- Operaciones vectorizadas automáticas

Los arrays de NumPy están optimizados para operaciones matemáticas masivas, haciéndolos ideales para análisis de datos, machine learning y computación científica.

Con NumPy evitamos bucles y el código es más corto y rápido.

PRÁCTICA

Importar NumPy y crear arrays

Antes de crear arrays, es fundamental importar la librería NumPy. Luego, podemos generar arrays de diversas formas, incluyendo a partir de listas de Python o con funciones dedicadas como `np.zeros()`, `np.ones()`, y `np.arange()`.

Crear desde lista (`np.array()`)

```
import numpy as np

# Array unidimensional
arr1 = np.array([1, 2, 3])
print(arr1)
# [1 2 3]

# Array bidimensional
arr2 = np.array([[1, 2], [3, 4]])
print(arr2)
# [[1 2]
#  [3 4]]
```

Arrays de ceros (`np.zeros()`)

```
# Array 1D de 5 ceros
zeros_1d = np.zeros(5)
print(zeros_1d)
# [0. 0. 0. 0. 0.]

# Array 2D de 2x3 ceros
zeros_2d = np.zeros((2, 3))
print(zeros_2d)
# [[0. 0. 0.]
#  [0. 0. 0.]]
```

Arrays de unos (`np.ones()`)

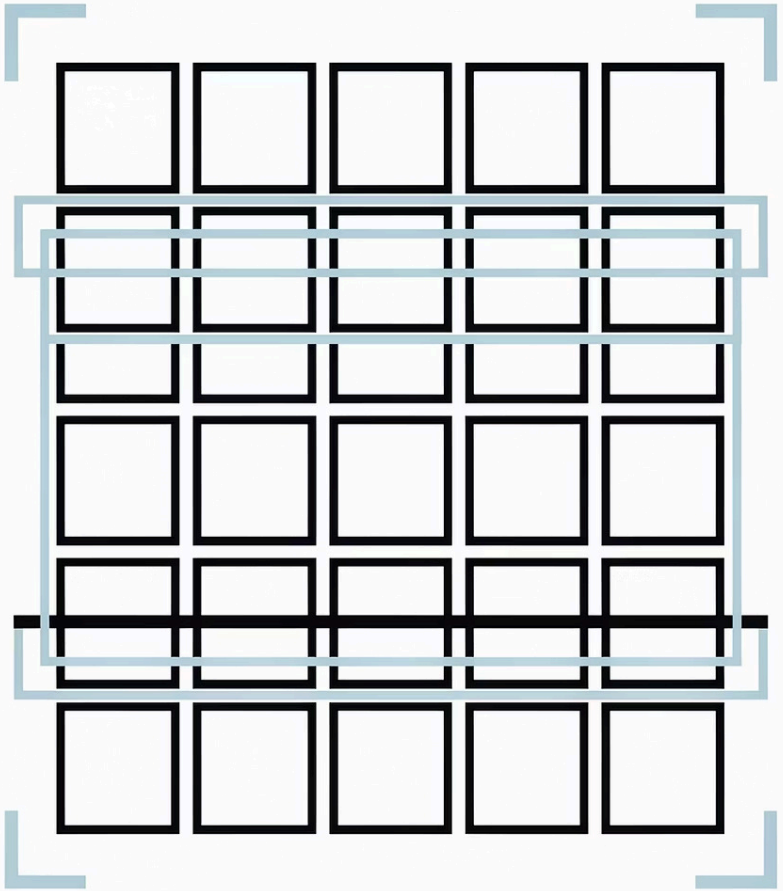
```
# Array 1D de 4 unos
ones_1d = np.ones(4)
print(ones_1d)
# [1. 1. 1. 1.]

# Array 2D de 3x2 unos
ones_2d = np.ones((3, 2))
print(ones_2d)
# [[1. 1.]
#  [1. 1.]
#  [1. 1.]]
```

Rango de valores (`np.arange()`)

```
# Números del 0 al 4
range_5 = np.arange(5)
print(range_5)
# [0 1 2 3 4]

# Números del 2 al 9 (paso 2)
range_step = np.arange(2, 10, 2)
print(range_step)
# [2 4 6 8]
```

Arrays multidimensionales

Cuando trabajamos con datos más complejos, como tablas o imágenes, los arrays unidimensionales ya no son suficientes. NumPy nos permite crear arrays con múltiples dimensiones, siendo los arrays 2D (matrices) los más comunes para representar datos tabulares.



Crear un array 2D (matriz)

```
import numpy as np

# Crear un array 2D (matriz) de 2 filas y 3 columnas
matriz_2d = np.array([
    [10, 20, 30],  # Fila 0
    [40, 50, 60]   # Fila 1
])

print("Matriz 2D:")
print(matriz_2d)

print("Forma de la matriz (filas, columnas):", matriz_2d.shape)
# Salida: (2, 3)
```

Cada elemento en un array 2D se identifica por su índice de fila y su índice de columna, permitiendo una organización y acceso intuitivos.

PROPIEDADES IMPORTANTES

Atributos importantes de un array

Los arrays de NumPy poseen atributos clave que definen su estructura. Entenderlos es esencial para manipular datos eficazmente:

```
import numpy as np

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

print(arr.shape)   # (2, 3)  - Filas y columnas
print(arr.ndim)    # 2     - Número de dimensiones
print(arr.dtype)   # int64 - Tipo de datos
```

- **shape**: Indica las dimensiones del array, por ejemplo (filas, columnas).
- **ndim**: Muestra el número de dimensiones del array.
- **dtype**: Define el tipo de datos de los elementos almacenados en el array.

Reorganizar arrays

NumPy ofrece potentes herramientas para manipular la estructura de los arrays, así como para acceder y extraer datos de manera eficiente. Dominar estas operaciones es clave para la preparación de datos.

1

`reshape()`: Cambiar la forma de un array

Permite modificar las dimensiones de un array sin cambiar sus datos. Es útil para adaptar el array a las expectativas de ciertas funciones o modelos.

```
import numpy as np

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

Tip: `reshape` NO cambia los datos, solo la forma.

2

Acceso a elementos

Puedes acceder a elementos individuales de un array usando índices, similar a las listas de Python pero adaptado a múltiples dimensiones.

```
import numpy as np

arr_2d = np.array([[10, 11, 12],
                   [13, 14, 15]])
# Acceder a un elemento:
print(arr_2d[0, 1]) # Salida: 11 (fila 0, columna 1)
```

3

Slicing básico

El slicing permite seleccionar subconjuntos de un array (filas, columnas o rangos), lo que es fundamental para extraer partes específicas de tus datos.

```
import numpy as np

data = np.array([1, 2, 3, 4, 5, 6, 7, 8])
# Slicing en 1D:
subset_1d = data[2:5] # [3 4 5]

matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])
# Slicing en 2D (todas las filas, columna 1):
subset_2d = matrix[:, 1] # [2 5 8]

# Slicing de un sub-matriz (filas 0-1, columnas 0-1):
sub_matrix = matrix[0:2, 0:2]
# [[1 2]
#  [4 5]]
```

OPERACIONES

Operaciones matemáticas vectorizadas

Una de las grandes ventajas de NumPy es la capacidad de realizar operaciones como **suma**, **resta** y **multiplicación** sobre arrays completos, incluyendo **operaciones con escalares**, sin necesidad de bucles explícitos. Esto se conoce como **vectorización** y es extremadamente eficiente, ofreciendo una notable **ventaja en velocidad** comparado con los bucles tradicionales de Python.

Operaciones con escalares

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

print(arr + 10)
# [11 12 13 14 15]

print(arr * 2)
# [2 4 6 8 10]

print(arr ** 2)
# [1 4 9 16 25]

print(arr / 2)
# [0.5 1.  1.5 2.  2.5]
```

Operaciones entre arrays

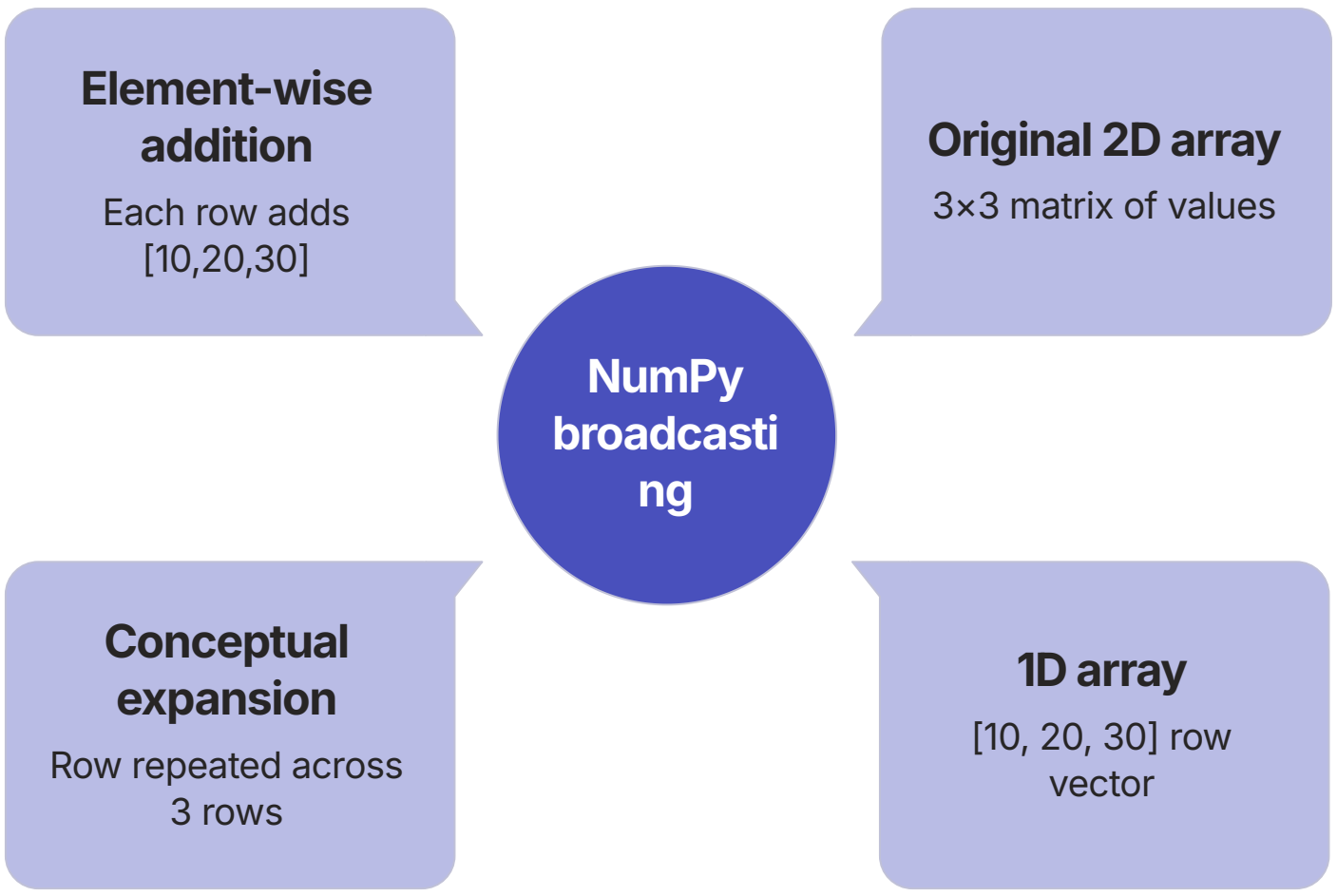
```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

print(a + b)
# [5 7 9]

print(a * b)
# [4 10 18]

print(b - a)
# [3 3 3]
```

Los arrays deben tener el mismo tamaño para operar elemento a elemento

CONCEPTOS CLAVE

Broadcasting (idea intuitiva)

Broadcasting es un mecanismo muy potente en NumPy que permite realizar operaciones entre arrays de diferentes formas o dimensiones de una manera muy eficiente, sin necesidad de duplicar datos explícitamente. Es como si el array más pequeño se "estirara" o "expandiera" para que sus dimensiones coincidan con el array más grande.



En el ejemplo visual, un vector 1D se "duplica" virtualmente a lo largo de las filas de una matriz 2D para permitir una suma elemento a elemento. Esto ocurre internamente en NumPy, optimizando el uso de memoria y la velocidad de cálculo.

Ejemplo: Sumar un vector a cada fila de una matriz

```
import numpy as np

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

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

# NumPy "broadcast" el vector para que coincida con cada fila de la matriz
resultado = matriz + vector

print("Matriz original:")
print(matriz)
print("\nVector a sumar:")
print(vector)
print("\nResultado (matriz + vector):")
print(resultado)
# [[11 22 33]
#  [14 25 36]
#  [17 28 39]]
```

En este código, NumPy entiende que queremos sumar el `vector` a **cada fila** de la `matriz`. Aunque el `vector` tiene una dimensión diferente, NumPy lo "expande" lógicamente (sin copiar datos) para que la operación sea posible. El resultado es una nueva matriz donde cada elemento del vector ha sido sumado a los elementos correspondientes de cada fila de la matriz original.

No se necesitan bucles explícitos, lo que hace el código más legible y la ejecución mucho más rápida, especialmente con arrays grandes. Esta es una de las razones principales por las que NumPy es tan eficiente para el cálculo numérico.

📈 ESTADÍSTICA BÁSICA

Estadística básica con NumPy

NumPy ofrece funciones eficientes para calcular estadísticas descriptivas esenciales. Comprender su uso es fundamental para el análisis de datos, especialmente con la capacidad de operar sobre ejes específicos en arrays multidimensionales.

Funciones estadísticas clave

- **np.mean()**: Calcula la media aritmética.
- **np.min()**: Encuentra el valor mínimo.
- **np.max()**: Encuentra el valor máximo.
- **np.sum()**: Calcula la suma de todos los elementos.

Para arrays bidimensionales (matrices), el parámetro `axis` permite especificar la dirección de la operación:

- `axis=0`: Opera a lo largo de las columnas.
- `axis=1`: Opera a lo largo de las filas.

```
import numpy as np

# Array unidimensional
datos_1d = np.array([5, 10, 15, 20, 25, 30])

print("Datos 1D:")
print(f"Media: {np.mean(datos_1d)}")    # 17.5
print(f"Mínimo: {np.min(datos_1d)}")    # 5
print(f"Máximo: {np.max(datos_1d)}")    # 30
print(f"Suma: {np.sum(datos_1d)}")    # 105

# Array bidimensional (matriz)
matriz = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])

print("\nMatriz 2D:")
print(matriz)
print(f"Media total: {np.mean(matriz)}")            # 5.0 (media de todos los elementos)
print(f"Suma por columnas (axis=0): {np.sum(matriz, axis=0)}") # [12 15 18]
print(f"Máximo por filas (axis=1): {np.max(matriz, axis=1)}")   # [3 6 9]
```

PRÁCTICA

Mini práctica guiada

Pongamos en práctica lo aprendido con un pequeño ejercicio que integra varias funciones de NumPy para la creación, manipulación y análisis básico de arrays.

Pasos a seguir:

01

Crear matriz aleatoria

Genera un array de NumPy 2D de 3x4 con valores aleatorios entre 0 y 100.

02

Calcular media

Obtén la media de todos los elementos del array.

03

Filtrar valores

Crea un nuevo array con solo los valores mayores de 50.

04

Normalizar valores

Normaliza el array original para que todos sus valores estén entre 0 y 1.

Aquí tienes el código completo que realiza todos los pasos:

```
import numpy as np

# 1. Crear matriz aleatoria (3 filas, 4 columnas, valores entre 0 y 100)
matriz_aleatoria = np.random.uniform(0, 100, (3, 4))
print("Matriz aleatoria original:")
print(matriz_aleatoria)

# 2. Calcular la media de todos los elementos
media_total = np.mean(matriz_aleatoria)
print(f"\nMedia de la matriz: {media_total:.2f}")

# 3. Filtrar valores mayores de 50
valores_altos = matriz_aleatoria[matriz_aleatoria > 50]
print("\nValores mayores de 50:")
print(valores_altos)

# 4. Normalizar los valores del array (Min-Max Normalization)
# Formula: (X - min(X)) / (max(X) - min(X))
min_val = np.min(matriz_aleatoria)
max_val = np.max(matriz_aleatoria)

matriz_normalizada = (matriz_aleatoria - min_val) / (max_val - min_val)
print("\nMatriz normalizada (valores entre 0 y 1):")
print(matriz_normalizada)
```

RESUMEN

Resumen: Lo que has aprendido

Hemos recorrido los fundamentos esenciales de NumPy, una librería indispensable para el cálculo numérico en Python. Durante esta presentación, hemos explorado cómo NumPy simplifica y acelera el trabajo con datos estructurados.

Arrays de NumPy

Los arrays son la estructura de datos central de NumPy, ofreciendo operaciones vectorizadas mucho más rápidas y eficientes que las listas de Python, especialmente para grandes volúmenes de datos.

Creación y Manipulación

Aprendimos a crear arrays de diversas formas (`np.array()`, `np.zeros()`, `np.arange()`), y técnicas clave de manipulación como `reshape`, indexación avanzada y slicing para acceder y modificar elementos.

Operaciones Matemáticas

Exploramos cómo realizar operaciones elemento por elemento, el concepto de `broadcasting` para trabajar con arrays de diferentes formas, y operaciones de álgebra lineal básica de manera eficiente.

Análisis Estadístico

Vimos cómo aplicar funciones estadísticas como `np.mean()`, `np.min()`, `np.max()` y `np.sum()`, utilizando el parámetro `axis` para calcular estadísticas a lo largo de filas o columnas.

Próximos Pasos

Con esta base, estás listo para adentrarte en librerías más avanzadas como Pandas para análisis de datos tabulares, Matplotlib para visualización, y el emocionante mundo de la ciencia de datos y el aprendizaje automático.

¡Felicidades! Si ya sabes crear arrays y operar con ellos de forma eficiente sin recurrir a bucles explícitos, ¡ya dominas NumPy y estás listo para el siguiente nivel en la ciencia de datos!

RECURSOS Y FUENTES DE INFORMACIÓN

Recursos y fuentes de información

Para profundizar en NumPy y seguir aprendiendo, aquí tienes una selección de recursos oficiales, tutoriales y materiales de referencia que te ayudarán en tu camino.

1

Documentación oficial

- Documentación oficial de NumPy
- Guía de inicio rápido
- Referencia de API completa

2

Tutoriales y cursos

- NumPy Tutorial (W3Schools)
- Real Python - NumPy Tutorial
- DataCamp - NumPy Cheat Sheet

3

Libros recomendados

- "Python for Data Analysis" por Wes McKinney
- "NumPy Beginner's Guide" por Ivan Idris
- "Guide to NumPy" por Travis Oliphant

4

Comunidad y ayuda

- Stack Overflow (etiqueta: numpy)
- GitHub oficial de NumPy
- Foro de discusión de NumPy