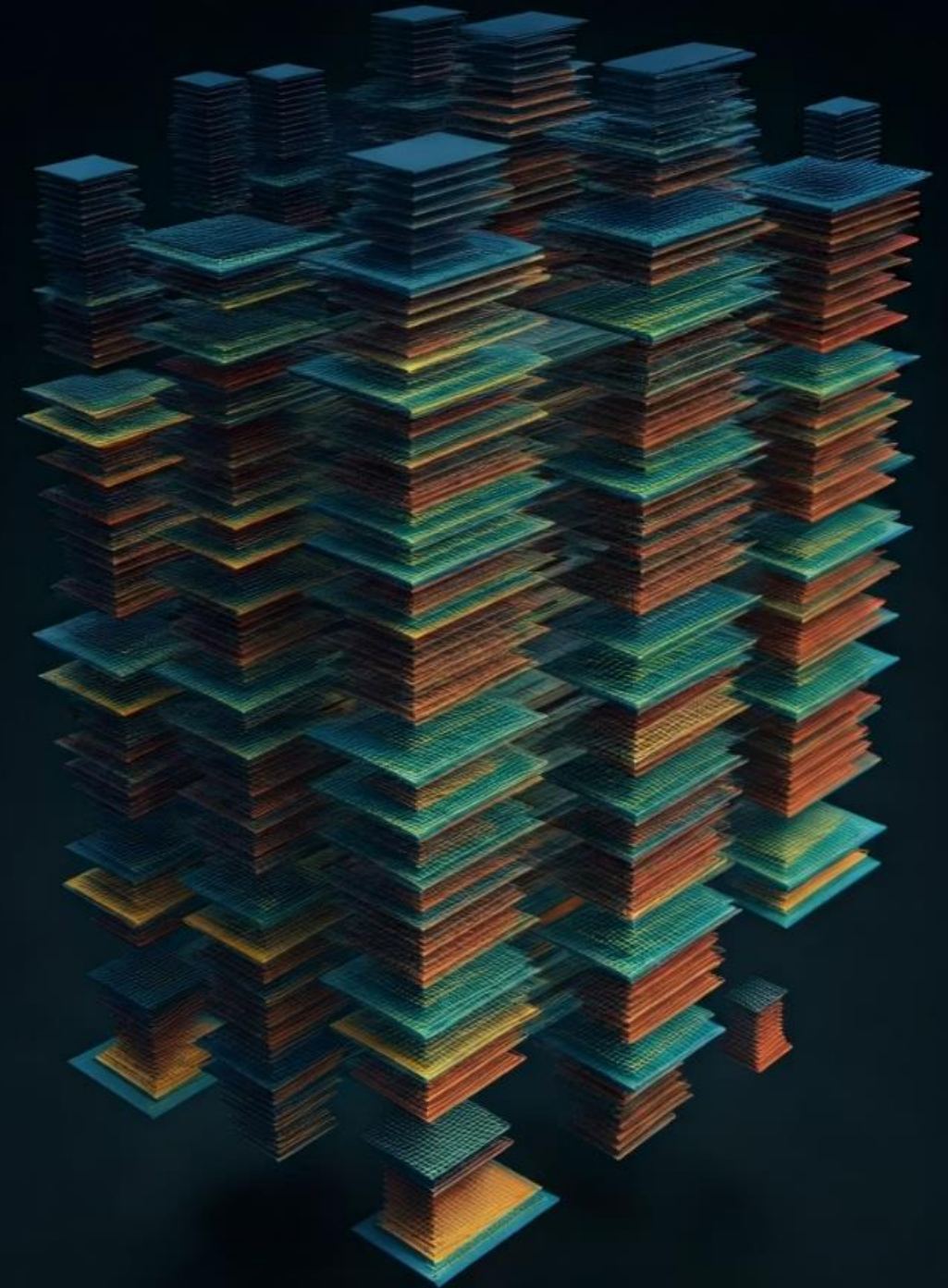


Manipulación de Estructuras de Datos Vectoriales y Matriciales con NumPy

La ingeniería de datos y la ciencia de datos modernas dependen de la capacidad para manipular y procesar eficientemente grandes volúmenes de información estructurada. Los vectores y matrices constituyen la base conceptual y computacional sobre la cual se sustentan muchos de los algoritmos y procedimientos utilizados en machine learning, inteligencia artificial, optimización y análisis estadístico. Python, junto con la biblioteca NumPy, proporciona las herramientas necesarias para crear, manipular y operar con estos objetos matemáticos de manera eficiente, elegante y escalable. Sin embargo, para utilizar NumPy a nivel profesional, es imprescindible comprender a fondo tanto los aspectos prácticos de su sintaxis como los principios matemáticos que la sustentan.

 **por Kibernetum Capacitación S.A.**



[illegible]

Base para Procesamiento de Datos

NumPy es esencial en la cadena de procesamiento de datos, siendo el fundamento sobre el que se construyen otras herramientas.

Pilar del Machine Learning

Constituye la base para bibliotecas como scikit-learn, TensorFlow y PyTorch, fundamentales en el desarrollo de modelos predictivos.

Automatización Industrial

Facilita la implementación de soluciones para la automatización de procesos industriales mediante el análisis eficiente de datos.

Análisis Estadístico

Proporciona las estructuras necesarias para realizar análisis estadísticos complejos de manera eficiente y escalable.



Preguntas de Activación

Eficiencia en Grandes Volúmenes de Datos

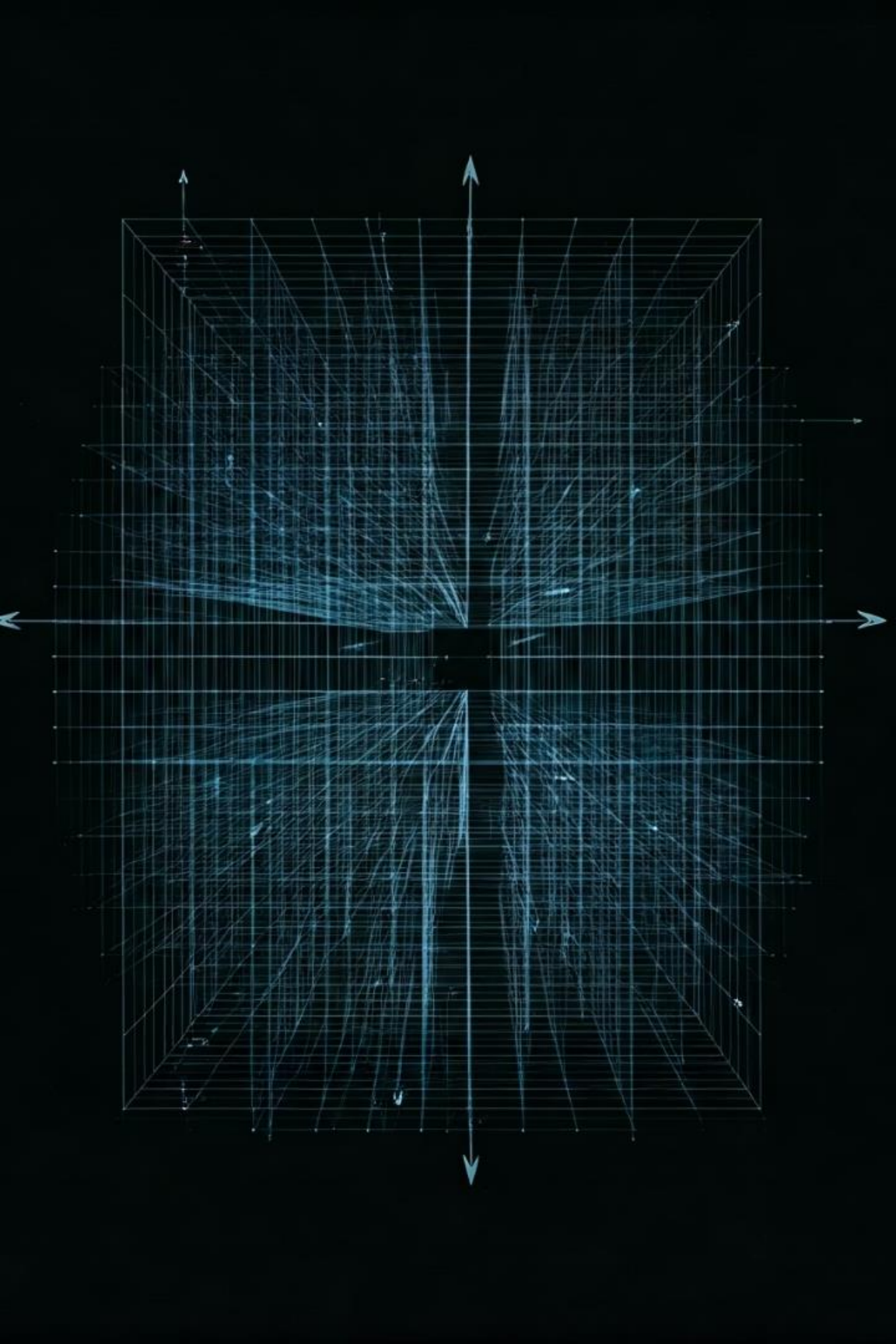
¿Por qué crees que la manipulación eficiente de grandes volúmenes de datos es uno de los pilares de la inteligencia artificial y la ingeniería moderna?

Ventajas de NumPy

¿En qué situaciones prácticas crees que es más ventajoso usar NumPy respecto a las listas tradicionales de Python?

Aplicaciones de Vectores y Matrices

¿Cuántas operaciones puedes imaginar que dependen directa o indirectamente de los vectores y matrices en ciencia de datos o tu área de interés?



Fundamentos Matemáticos: Álgebra Lineal para Ingeniería de Datos

"El álgebra lineal es el lenguaje universal de los datos multidimensionales. Si alguna vez has trabajado con datos tabulares, imágenes, señales o cualquier tipo de registro compuesto por múltiples variables, has interactuado con objetos que se representan naturalmente como vectores y matrices."



Vectores

Entidades matemáticas representadas como secuencias ordenadas de números



Matrices

Tablas rectangulares de números organizadas en filas y columnas



Sistemas de Ecuaciones

Representación matricial de problemas matemáticos



Normas y Distancias

Medidas de magnitud y similitud entre vectores

Vectores: Definición, Propiedades y Operaciones

Definición Formal

"Un vector es una entidad matemática representada como una secuencia ordenada de números (componentes), habitualmente denotada como una columna o fila.

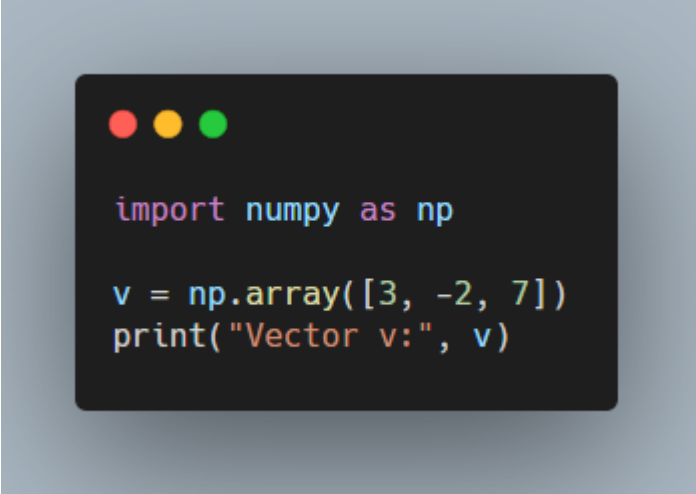
Formalmente, si n es un entero positivo, un vector en \mathbb{R}^n es una tupla (x_1, x_2, \dots, x_n) donde cada x_i es un número real."

Interpretaciones

- Geometría: un punto en n dimensiones
- Física: magnitud y dirección (fuerza, velocidad)
- Datos: instancia o registro multivariable

Ejemplo en Ingeniería

Un vector de 3 componentes puede ser la aceleración en el espacio 3D, una mezcla de señales, o parámetros de un proceso industrial.



```
import numpy as np

v = np.array([3, -2, 7])
print("Vector v:", v)
```

Operaciones con Vectores

+

Suma de Vectores

"La suma (y resta) de vectores es una operación elemento a elemento:"

```
import numpy as np

v = np.array([1, 2, 3])
w = np.array([4, 5, 6])
suma = v + w
print("Suma:", suma) # [5, 7, 9]
```

Ⓜ

Multiplicación por Escalar

```
import numpy as np

a = np.array([2, 3])
escalar = 5
print("Escalar * vector:", escalar * a) # [10, 15]
```

📏

Norma (longitud) de un Vector

"La norma o longitud de un vector mide su 'tamaño' en el espacio:"

```
import numpy as np

v = np.array([3, 4])
norma = np.linalg.norm(v)
print("Norma:", norma) # 5.0
```

⊙

Producto Punto (dot product)

"El producto escalar entre dos vectores de igual dimensión es la suma del producto de sus componentes:"

```
import numpy as np

v = np.array([1, 2, 3])
w = np.array([4, 5, 6])
prod_punto = np.dot(v, w)
print("Producto punto:", prod_punto) # 32
```

Aplicación: cálculo de ángulos, proyección de fuerzas, similitud en machine learning.

Matrices: Definición, Propiedades y Operaciones

Definición

"Una matriz es una tabla rectangular de números organizada en filas y columnas. Matemáticamente, una matriz de tamaño $m \times n$ es una función que asocia a cada par ordenado (i, j) un número real, representando así m filas y n columnas."

```
import numpy as np

M = np.array([[1, 2], [3, 4]])
print("Matriz M:\n", M)
```

Operaciones Básicas

Transpuesta de una matriz:

```
print("Transpuesta de M:\n", M.T)
```

Multiplicación por escalar:

```
print("2 * M:\n", 2 * M)
```

Matrices Especiales

"Matriz identidad: matriz cuadrada con unos en la diagonal y ceros en el resto. Es el equivalente matricial del número 1:"

```
I = np.eye(2)
print("Identidad 2x2:\n", I)
```

Multiplicación de matrices:

```
A = np.array([[1, 2], [3, 4]])
B = np.array([[2, 0], [1, 2]])
prod = np.dot(A, B)
print("Producto matricial:\n", prod)
```

Sistemas de Ecuaciones Lineales



Representación Matricial

"Un sistema de ecuaciones lineales puede escribirse como $Ax = b$, donde A es la matriz de coeficientes, x el vector incógnita y b el vector de resultados."



Resolución con NumPy

Resolver sistemas de ecuaciones lineales utilizando álgebra matricial



Implementación

```
A = np.array([[2, 3], [5, 7]])  
b = np.array([8, 19])  
sol = np.linalg.solve(A, b)  
print("Solución del sistema:", sol)
```

Este ejemplo muestra cómo resolver el sistema de ecuaciones $2x + 3y = 8$ y $5x + 7y = 19$ utilizando NumPy, transformando un problema matemático en una solución computacional eficiente.

Espacios Vectoriales, Rang o y Subespacios

Espacio Vectorial

"Un espacio vectorial es un conjunto de vectores que cumple ciertas propiedades (cierre bajo suma y multiplicación por escalar)."

Rang o de una Matriz

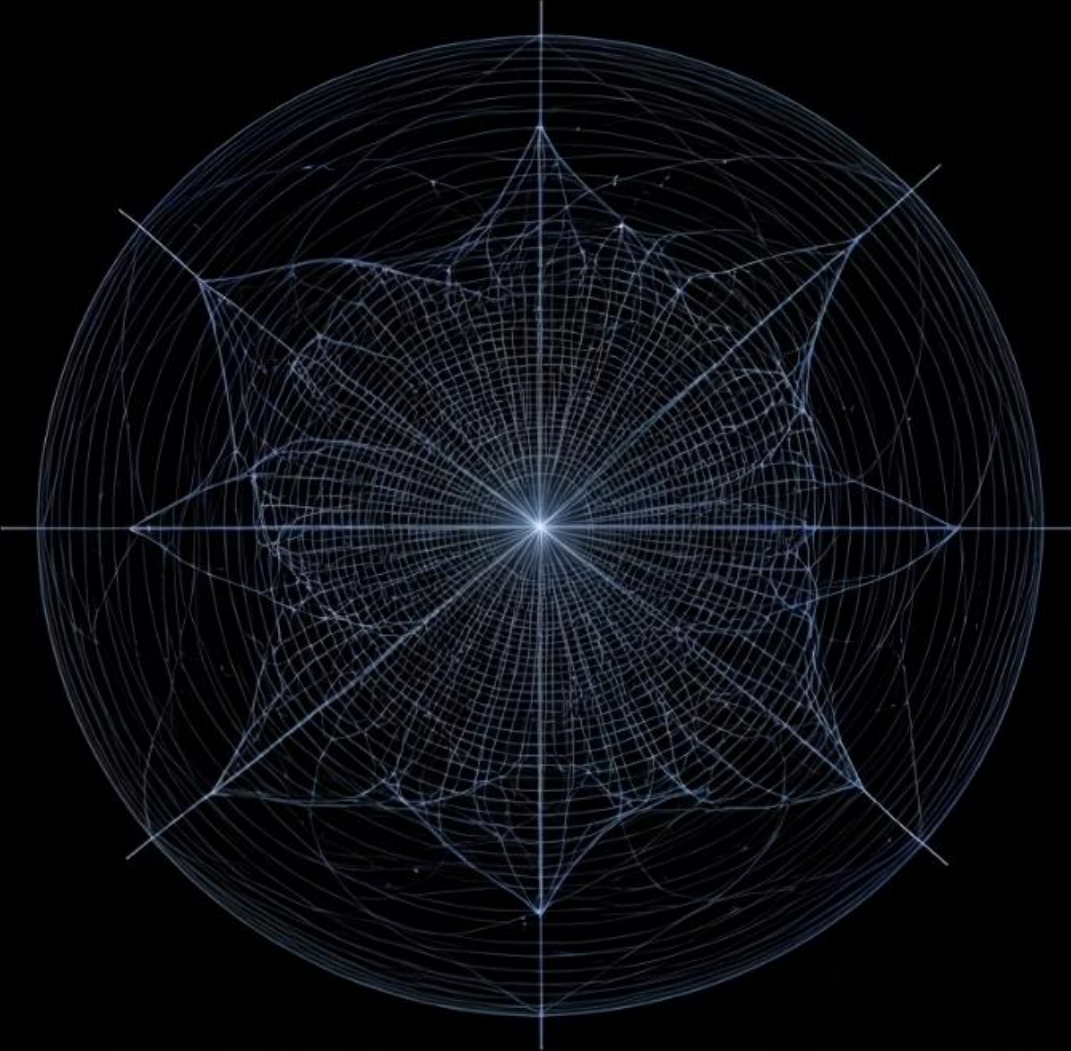
```
M = np.array([[1, 2], [2, 4]])  
rango = np.linalg.matrix_rank(M)  
print("Rango de M:", rango) # 1
```

Interpretación

Si el rango es menor que el número de columnas, hay dependencias lineales.

Aplicaciones

El concepto de rango es fundamental en análisis de componentes principales, compresión de datos y resolución de sistemas de ecuaciones.



Normas, Distancias y Similitud



Norma de un Vector

"La norma de un vector cuantifica su longitud (magnitud), mientras que la distancia euclidiana mide qué tan alejados están dos puntos en el espacio de datos."



Distancia Euclidiana

```
a = np.array([1, 2])
b = np.array([4, 6])
distancia = np.linalg.norm(a - b)
print("Distancia euclídea:", distancia)
```



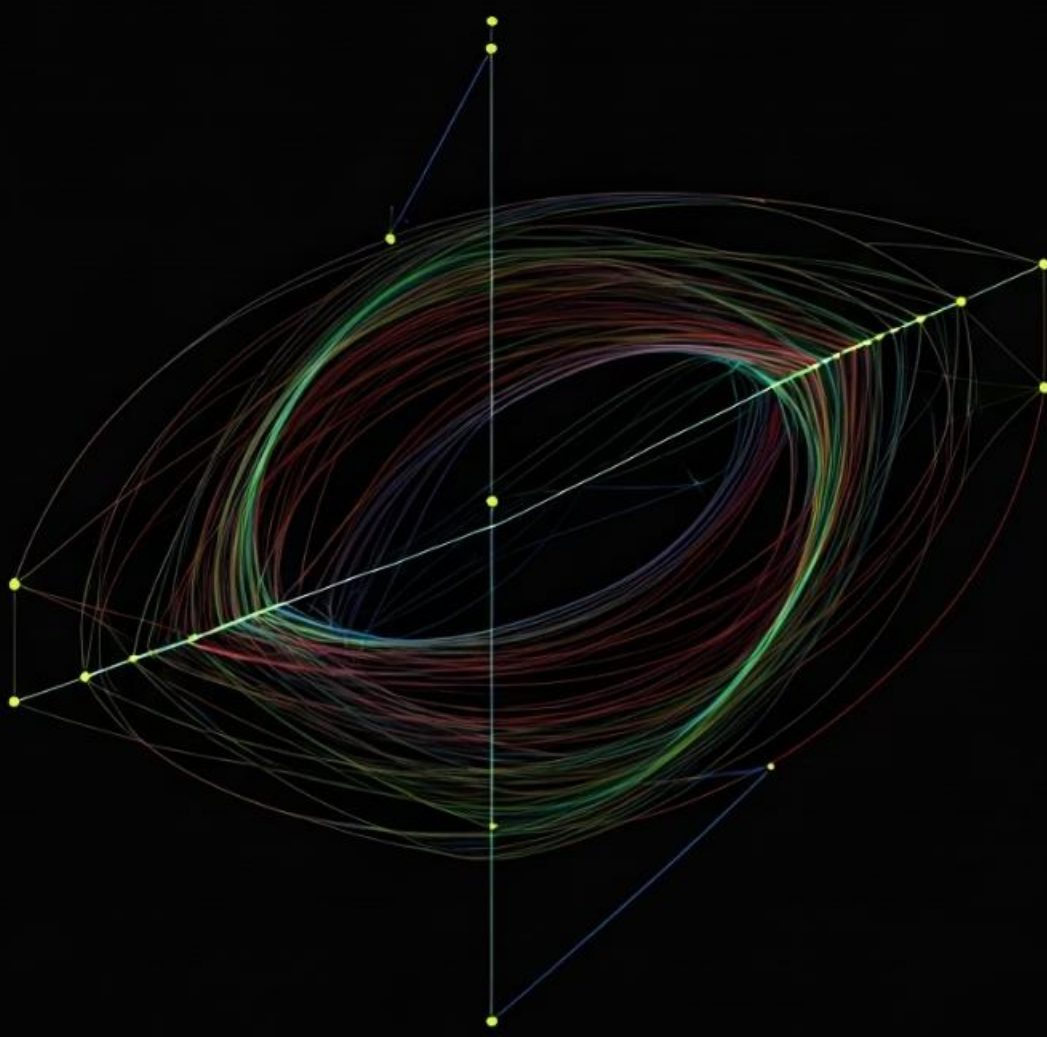
Similitud Coseno

Medida de similitud entre vectores basada en el ángulo entre ellos, muy utilizada en procesamiento de lenguaje natural y recuperación de información.

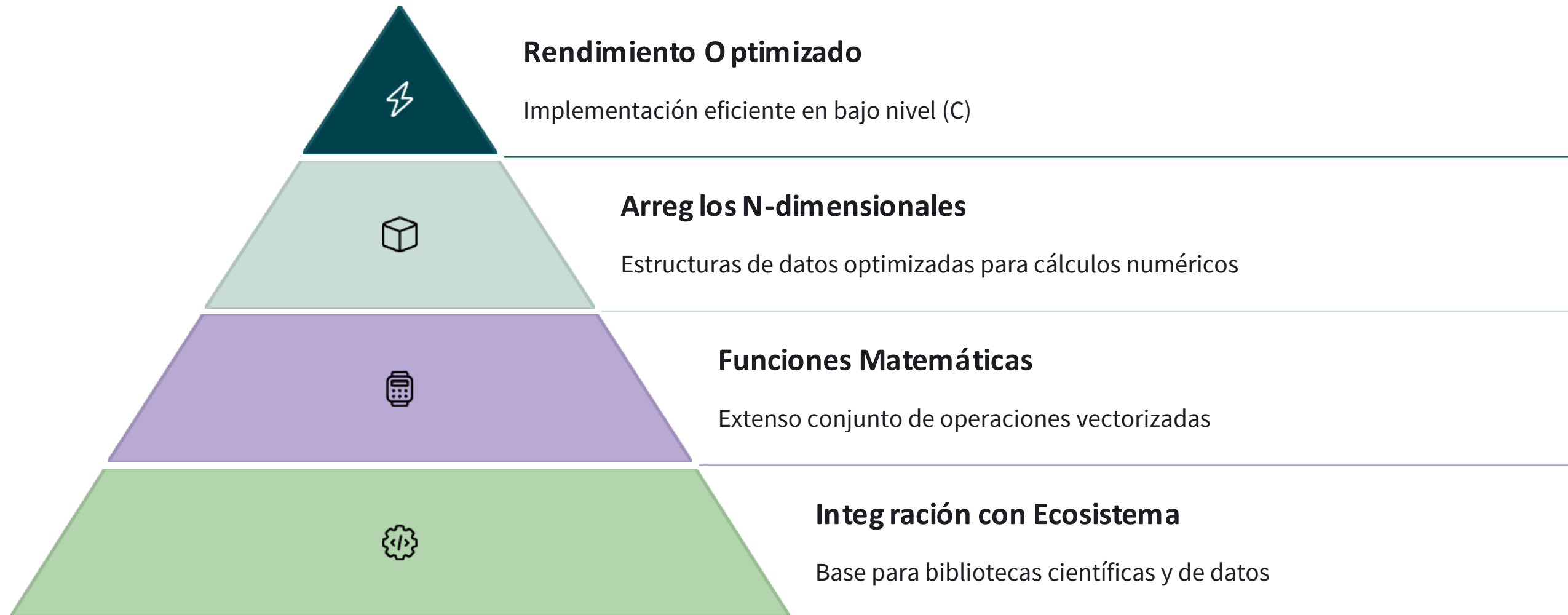


Aplicaciones

Fundamental en algoritmos de clustering, sistemas de recomendación y detección de anomalías.



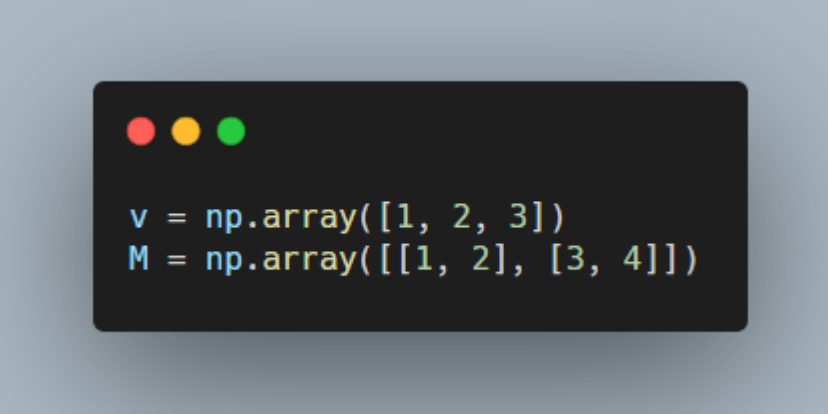
¿Por qué NumPy?



"NumPy provee una implementación eficiente de los arreglos n-dimensionales (ndarrays), junto con un extenso conjunto de funciones matemáticas optimizadas en bajo nivel (C)."

Creación de Arreglos en NumPy

Desde Listas

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. It displays two lines of Python code: `v = np.array([1, 2, 3])` and `M = np.array([[1, 2], [3, 4]])`.

```
v = np.array([1, 2, 3])
M = np.array([[1, 2], [3, 4]])
```

Funciones Preconstruidas

- `np.arange(0, 10, 2) → [0 2 4 6 8]`
- `np.zeros((3, 4)) → matriz de ceros`
- `np.ones((2, 5)) → matriz de unos`
- `np.linspace(0, 1, 5) → [0. 0.25 0.5 0.75 1.]`
- `np.eye(3) → matriz identidad 3x3`
- `np.random.rand(2, 3) → matriz de valores aleatorios entre 0 y 1`

Redimensionado de Arreglos

Cambio de Forma

"Modificar la forma (shape) de un arreglo es fundamental para adaptar los datos a diferentes etapas de un pipeline o modelo."

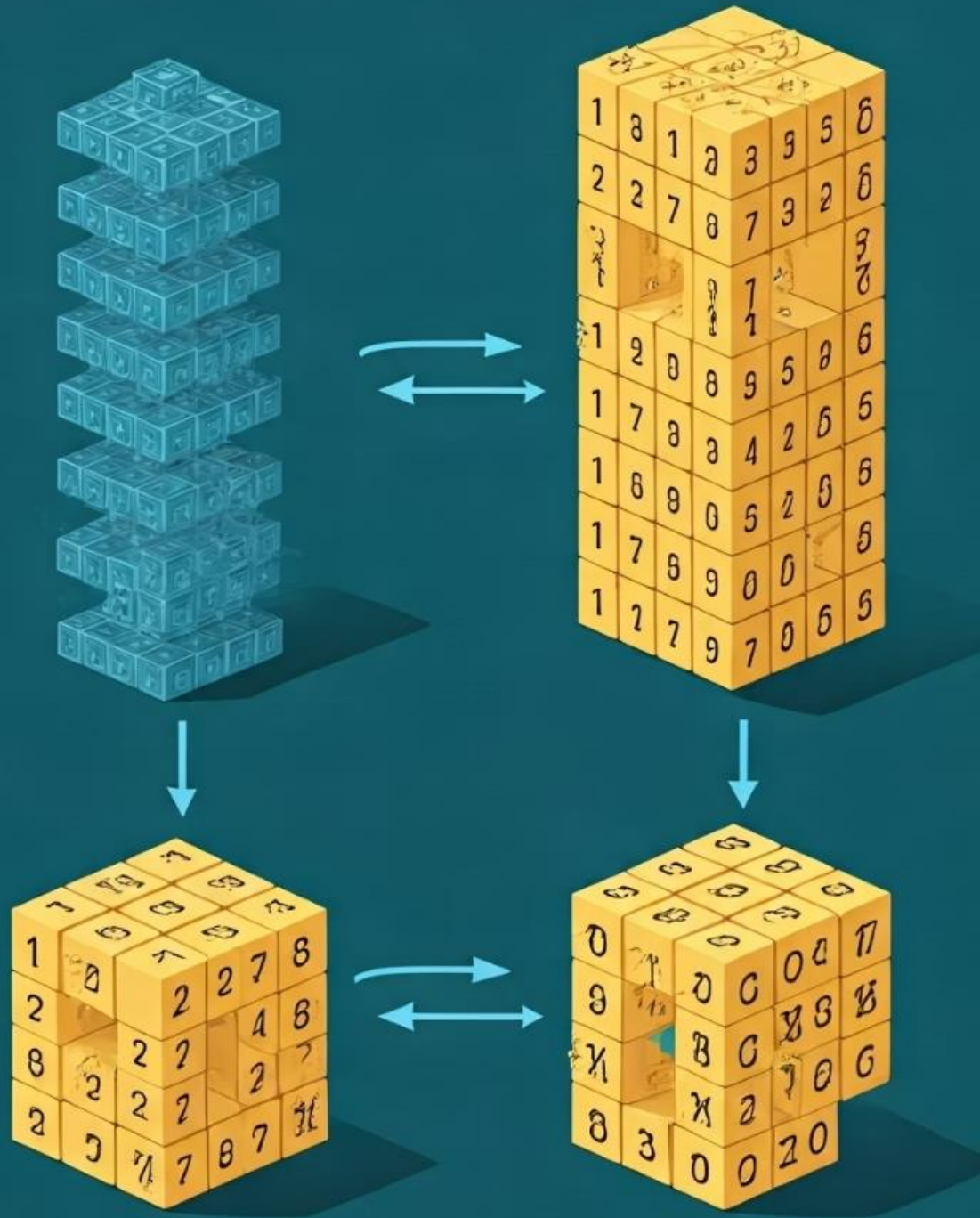
Reshape

```
arr = np.arange(12)
mat = arr.reshape((3, 4)) # matriz 3x4
print("Redimensionado a 3x4:\n", mat)
```

Flatten

Volver a 1D:

```
plano = mat.flatten()
print("Aplanado:", plano)
```



Indexación y Selección de Elementos

Por Índice y Slicing

```
M = np.array([[10, 20, 30],
              [40, 50, 60],
              [70, 80, 90]])
print(M[1, 2]) # 60 (fila 1, columna 2)
print(M[:, 1]) # [20 50 80] (columna 1 de todas las filas)
```

Condicional (Boolean Indexing)

```
edades = np.array([16, 21, 30, 40, 15, 60])
mayores = edades[edades >= 18]
print(mayores) # [21 30 40 60]
```

Esta técnica es extremadamente útil para filtrar datos basados en condiciones específicas, permitiendo operaciones de selección complejas con una sintaxis concisa.

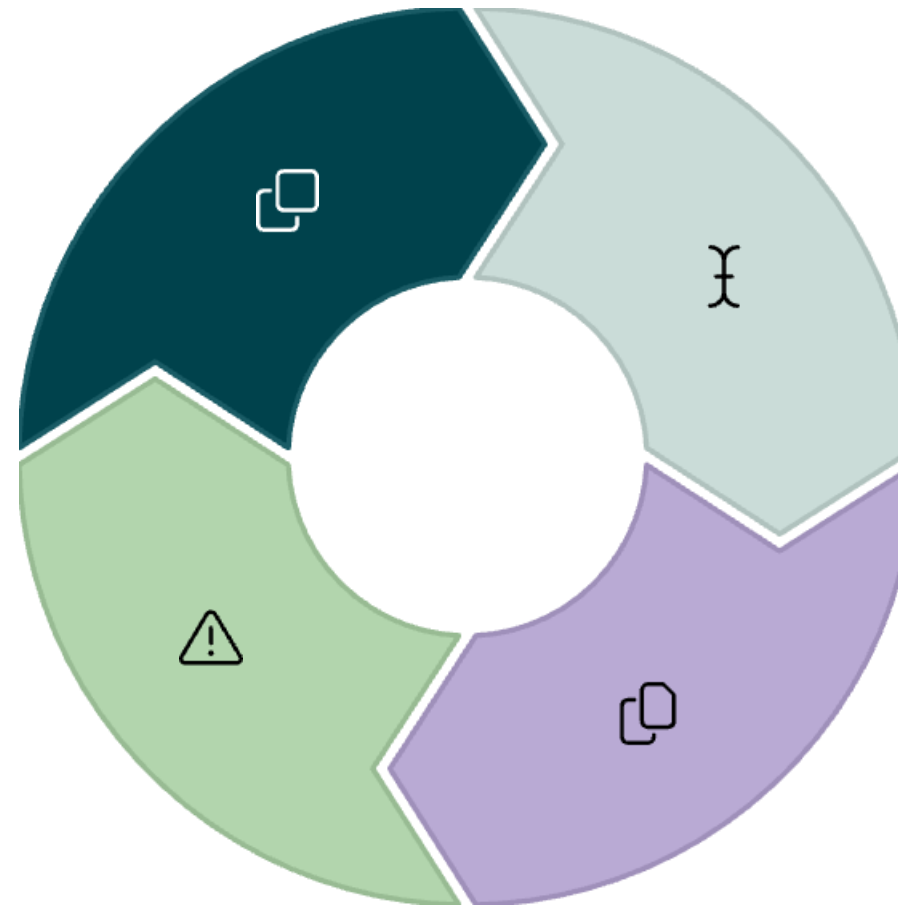
Referencia y Copia de Arreglos

Vistas (Views)

"Por defecto, NumPy crea vistas (views) sobre los datos. Para evitar efectos colaterales, se recomienda usar `.copy()` al generar subconjuntos modificables."

Importancia

Comprender la diferencia entre vistas y copias es crucial para evitar errores sutiles en el procesamiento de datos.



Modificación de Vistas

```
original = np.array([1, 2, 3, 4])
vista = original[1:3]
vista[0] = 100
print(original) # [1 100 3 4]
```

Creación de Copias

```
copia = original[1:3].copy()
copia[0] = 200
print(original) # [1 100 3 4]
```

Operaciones entre Arreglos y con Escalares

Operaciones Elemento a Elemento

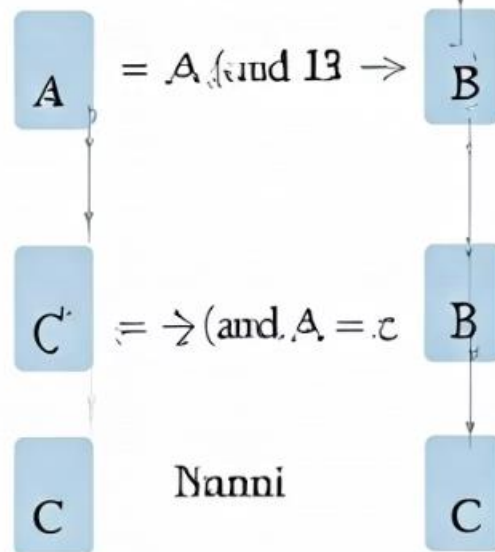
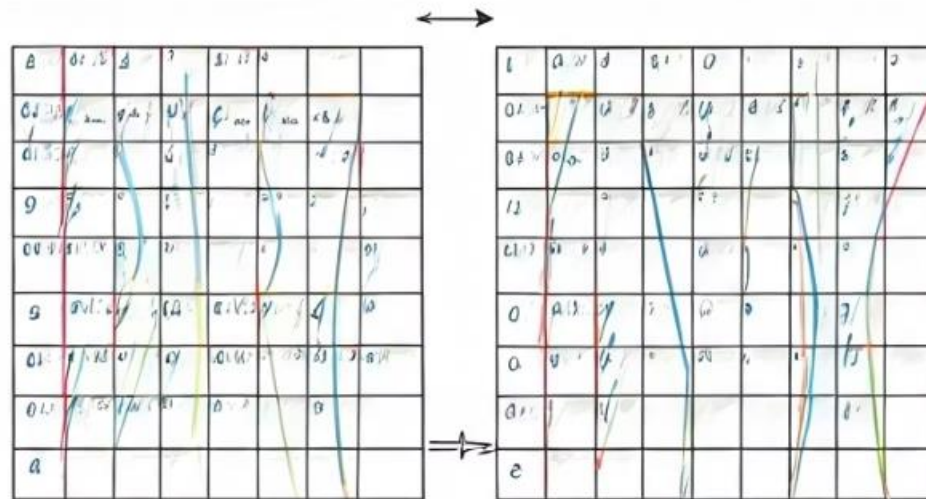
```
x = np.array([1, 2, 3])
y = np.array([10, 20, 30])
print(x + y) # [11 22 33]
print(x * y) # [10 40 90]
print(x * 2) # [2 4 6]
print(x + 1) # [2 3 4]
```

Funciones Universales (ufuncs)

```
a = np.array([1, 4, 9])
print(np.sqrt(a)) # [1. 2. 3.]
print(np.exp(a)) # [2.71828183 54.59815003 8103.08392758]
print(np.sin(a)) # [ 0.84147098 -0.7568025  0.41211849]
```

Las funciones universales de NumPy operan elemento a elemento de manera optimizada, permitiendo cálculos vectorizados que son mucho más rápidos que los bucles tradicionales.

Matriz Multiplication



C. 6. Summed

Operaciones Matriciales

Producto Punto de Vectores

```
a = np.array([1, 2, 3])
b = np.array([0, 1, 0])
print(np.dot(a, b)) # 2
```

Multiplicación de Matrices

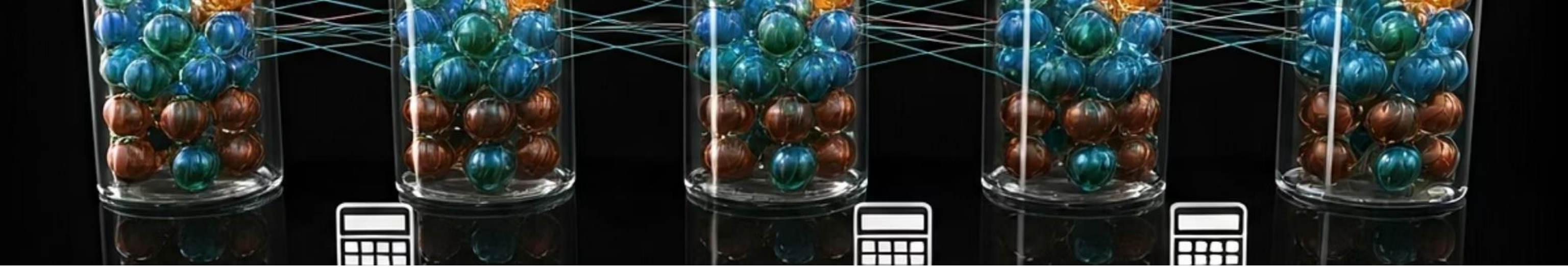
```
A = np.array([[1, 2], [3, 4]])
B = np.array([[2, 0], [1, 2]])
print(np.dot(A, B)) # [[ 4  4]
                    # [10  8]]
print(A @ B)       # equivalente
```

Aplicaciones

Estas operaciones son fundamentales en algoritmos de machine learning, procesamiento de imágenes y simulaciones físicas.

Eficiencia

NumPy utiliza bibliotecas optimizadas como BLAS y LAPACK para realizar estas operaciones de manera extremadamente eficiente.



Aplicando Funciones y Operaciones Agregadas

75

Suma Total

```
datos = np.array([5, 10, 15, 20, 25])  
print(np.sum(datos)) # 75
```

15.0

Media

```
print(np.mean(datos)) # 15.0
```

25

Máximo

```
print(np.max(datos)) # 25
```

7.071

Desviación Estándar

```
print(np.std(datos)) # 7.0710678118654755
```

Selección Condicional y Filtrado



Importancia

"La selección condicional es fundamental para limpieza, exploración y transformación de datos."



Ejemplo Simple

```
temperaturas = np.array([12, 19, 22, 30, 35, 40, 29, 15])
filtro = (temperaturas >= 15) & (temperaturas <= 35)
valores_filtrados = temperaturas[filtro]
print(valores_filtrados) # [19 22 30 35 29 15]
```



Múltiples Condiciones

```
temp = np.random.randint(10, 45, (12, 30))
hum = np.random.randint(20, 80, (12, 30))
dias = np.where((temp > 35) & (hum < 40))
print("Meses:", dias[0])
print("Días:", dias[1])
```

Referencia y Copia de Arreglos (Avanzado)

Vistas vs Copias

Modificar una vista afecta el arreglo original; una copia no.

```
original = np.array([1, 2, 3, 4])
vista = original[1:3]
vista[0] = 100
print(original) # [1 100 3 4]
```

Creación Explícita de Copias

```
copia = original[1:3].copy()
copia[0] = 200
print(original) # [1 100 3 4]
```

Implicaciones en Rendimiento

Las vistas son más eficientes en términos de memoria, pero pueden causar efectos secundarios inesperados si no se manejan correctamente. Las copias consumen más memoria pero garantizan la independencia de los datos.

Multiplicación de Matrices y Producto Punto

Ejemplo de Aplicación

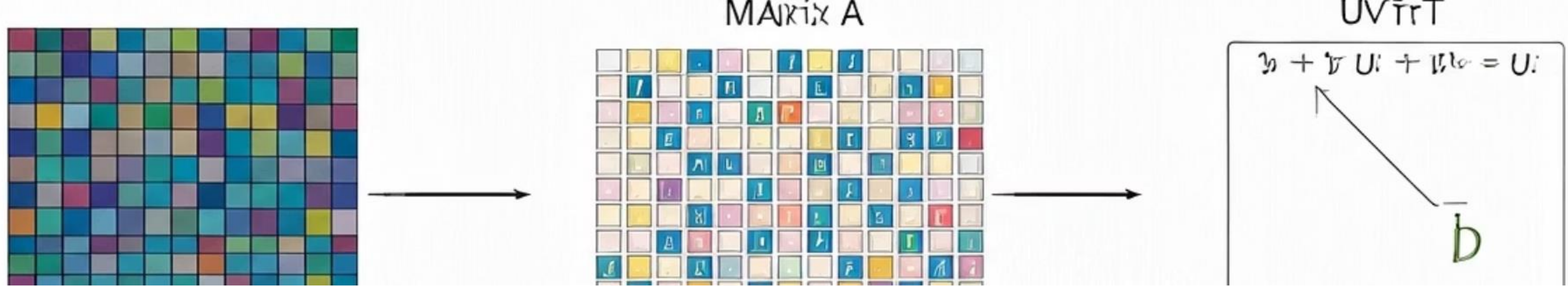
```
X = np.array([[1, 2, 3],  
              [4, 5, 6],  
              [7, 8, 9],  
              [10, 11, 12]])  
w = np.array([0.1, 0.2, 0.3])  
resultados = np.dot(X, w)  
print(resultados) # Salida: [1.4 3.2 5.0 6.8]
```

Este ejemplo muestra cómo se puede aplicar un vector de pesos a un conjunto de características, operación fundamental en modelos lineales de machine learning.

Matrices Especiales

```
identidad = np.eye(3)  
print(identidad)  
unos = np.ones((3, 4))  
print(unos)  
aleatoria = np.random.rand(2, 3)  
print(aleatoria)
```

NumPy proporciona funciones para crear matrices especiales como la identidad, matrices de unos o matrices aleatorias, que son útiles en diversas aplicaciones.



Reducción de Dimensionalidad y Descomposiciones



Inversa

```
A = np.array([[1, 2], [3, 4]])
print("Inversa:\n", np.linalg.inv(A))
```



Determinante

```
print("Determinante:", np.linalg.det(A))
```



Autovalores y
Autovectores

```
valores, vectores = np.linalg.eig(A)
print("Autovalores:", valores)
print("Autovectores:\n", vectores)
```



Aplicaciones

Estas operaciones son fundamentales en técnicas como PCA (Análisis de Componentes Principales), compresión de imágenes y sistemas de recomendación.

Ejemplo Integrador: Aplicación Completa

Simulación de Datos

"Imagina un caso real en ingeniería de datos: tienes registros horarios de consumo energético de 1000 usuarios durante una semana (168 horas)."

```
np.random.seed(42)
consumos = np.random.normal(loc=50, scale=10, size=(1000, 168))
```

Filtrado de Usuarios

```
promedios = np.mean(consumos, axis=1)
filtro = promedios > 55
usuarios_filtrados = consumos[filtro]
```

Normalización por Usuario

```
minimos = np.min(usuarios_filtrados, axis=1, keepdims=True)
maximos = np.max(usuarios_filtrados, axis=1, keepdims=True)
consumos_normalizados = (usuarios_filtrados - minimos) / (maximos - minimos)
```

Matriz de Correlación Horaria

```
matriz_corr = np.corrcoef(consumos_normalizados.T)
print("Matriz de correlación de dimensiones:", matriz_corr.shape)
```



Resumen de Aplicaciones Industriales



Modelos Predictivos

Entrenamiento = multiplicación de matrices + funciones sobre arreglos



Análisis Exploratorio

Filtrar, seleccionar, transformar datos para descubrir patrones y tendencias



Optimización

Descenso por gradiente (productos/sumas vectorizadas) para encontrar soluciones óptimas



Visualización

Preparar datos para gráficos, análisis de tendencias y presentación de resultados

Ejercicio 1: Manipulación de Arreglos



Enunciado

Crea un arreglo de 100 números aleatorios, normalízalo y selecciona los valores entre -1 y 1.



Solución

```
import numpy as np
np.random.seed(0)
arr = np.random.randn(100)
arr_norm = (arr - np.mean(arr)) / np.std(arr)
valores = arr_norm[(arr_norm >= -1) & (arr_norm <= 1)]
print("Cantidad de valores entre -1 y 1:", len(valores))
```



Conceptos Aplicados

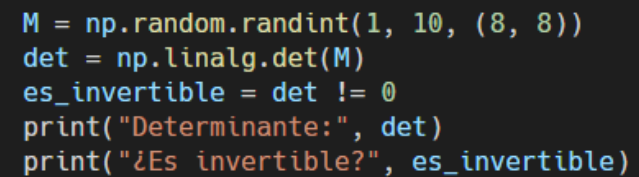
Generación de números aleatorios, normalización estadística y filtrado condicional.

Ejercicio 2: Álgebra Lineal Aplicada

Enunciado

Genera una matriz 8x8 con números enteros aleatorios, calcula su determinante y verifica si es invertible.

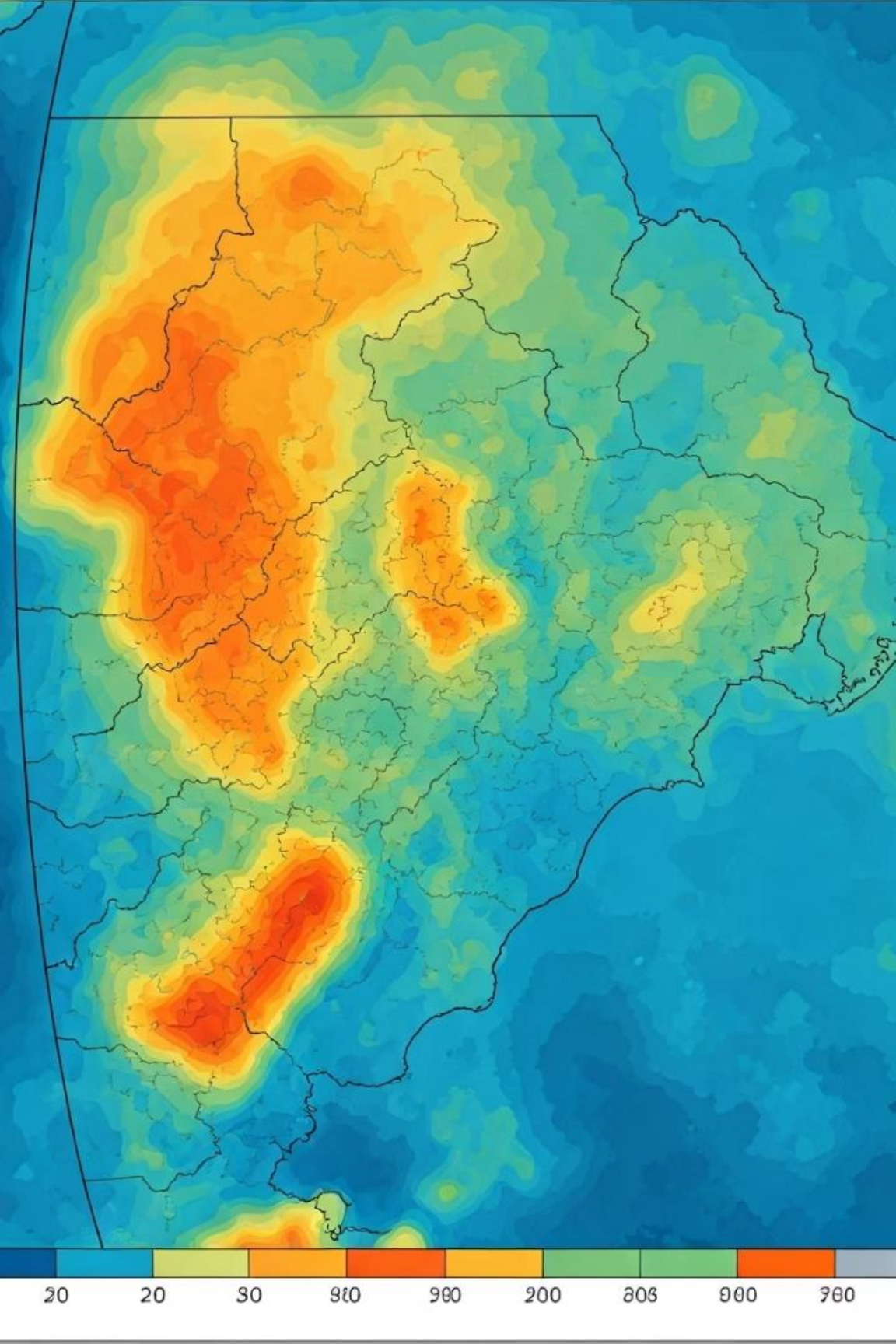
Solución



```
M = np.random.randint(1, 10, (8, 8))
det = np.linalg.det(M)
es_invertible = det != 0
print("Determinante:", det)
print("¿Es invertible?", es_invertible)
```

Importancia

El determinante de una matriz es crucial para determinar si tiene inversa. Una matriz es invertible si y solo si su determinante es diferente de cero. Las matrices invertibles son fundamentales en la resolución de sistemas de ecuaciones lineales y en muchas aplicaciones de álgebra lineal.



Ejercicio 3: Selección Condicional Compleja

Enunciado

Dados datos meteorológicos (matriz de 12 meses \times 30 días), selecciona todos los días donde la temperatura fue superior a 35°C y la humedad inferior a 40%.

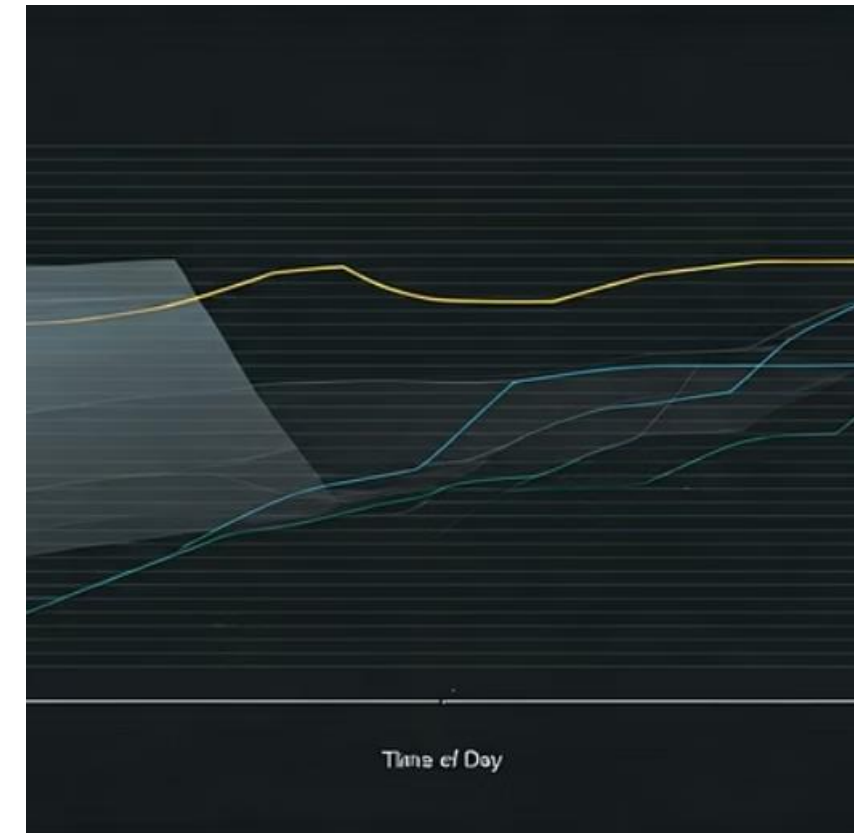
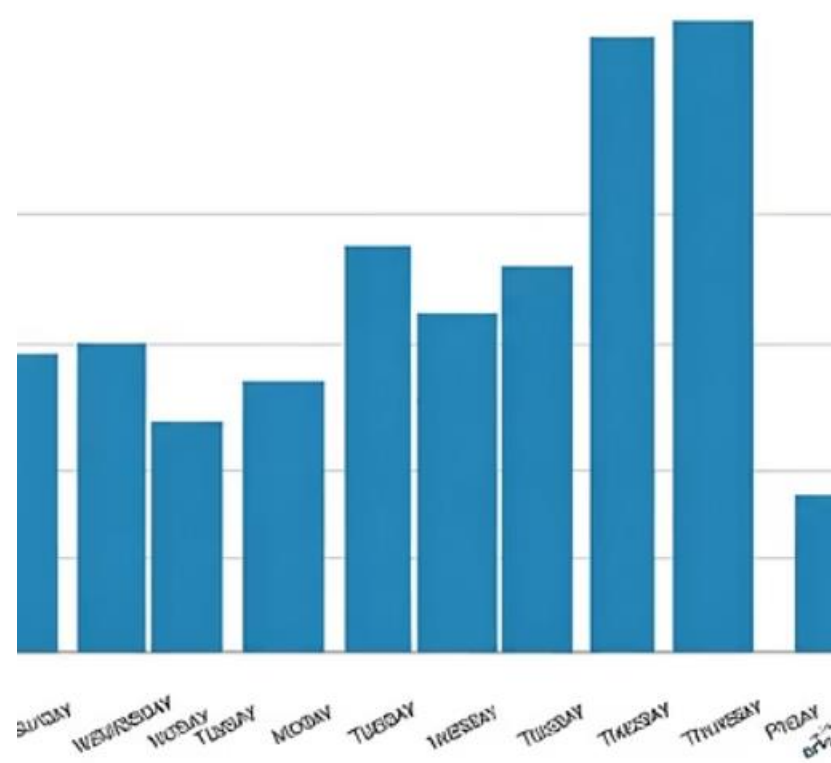
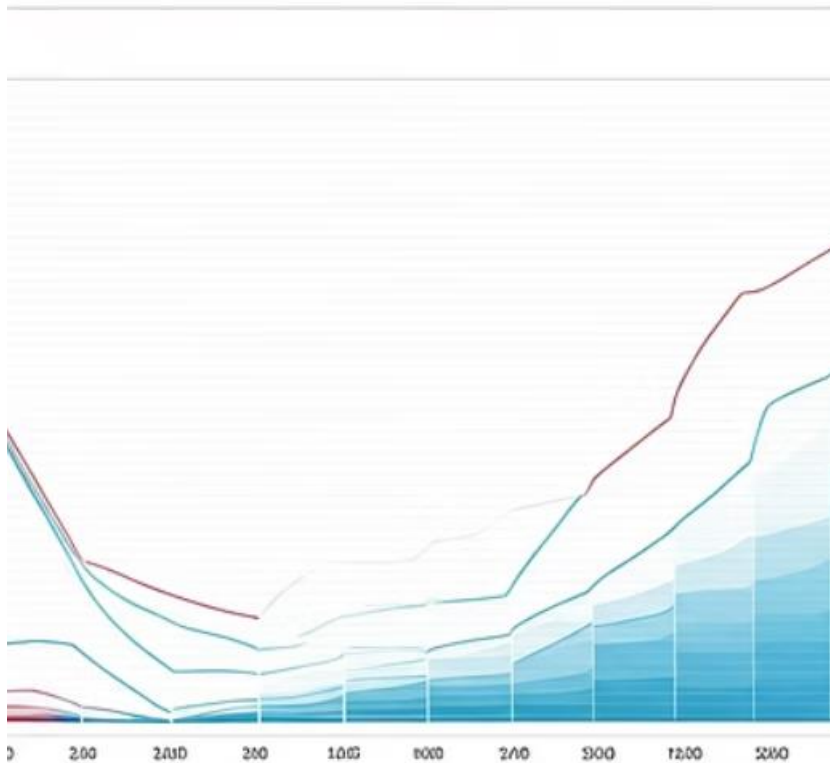
Solución

```
temp = np.random.randint(10, 45, (12, 30))
hum = np.random.randint(20, 80, (12, 30))
dias = np.where((temp > 35) & (hum < 40))
print("Meses:", dias[0])
print("Días:", dias[1])
```

Aplicación Práctica

Este tipo de selección condicional es fundamental en análisis climático, detección de condiciones extremas y planificación de recursos basada en patrones meteorológicos.

Ejercicio 4: Transformación y Agregación



Enunciado: Simula las ventas de una tienda (matriz 7x24, días x horas) y encuentra las horas pico (top 3 valores por fila).

```
ventas = np.random.randint(1, 100, (7, 24))
top_horas = np.argsort(ventas, axis=1)[: , -3:]
print("Índices de horas pico por día:\n", top_horas)
```

Preguntas de Reflexión y Puntos Clave

Preguntas Abiertas

1. ¿Cómo influye la correcta manipulación de arreglos en la calidad y rapidez de los análisis de datos?
2. ¿Qué problemas podrían surgir si se confunde referencia y copia en NumPy?
3. ¿Qué otras aplicaciones industriales puedes imaginar para el manejo vectorial y matricial eficiente?

5 Puntos Clave

1. NumPy es la herramienta estándar para manipulación eficiente de datos numéricos en Python.
2. Vectores y matrices permiten modelar desde sensores hasta sistemas complejos de machine learning.
3. Las funciones de creación, redimensionado, indexación y filtrado son esenciales en cualquier pipeline de datos.
4. Comprender las diferencias entre referencia y copia es crucial para evitar errores sutiles.
5. Las operaciones vectorizadas aceleran cálculos y permiten escalabilidad.