Recibe una cálida:

¡Bienvenida!

Te estábamos esperando 😁

alkemy

Clase grabada

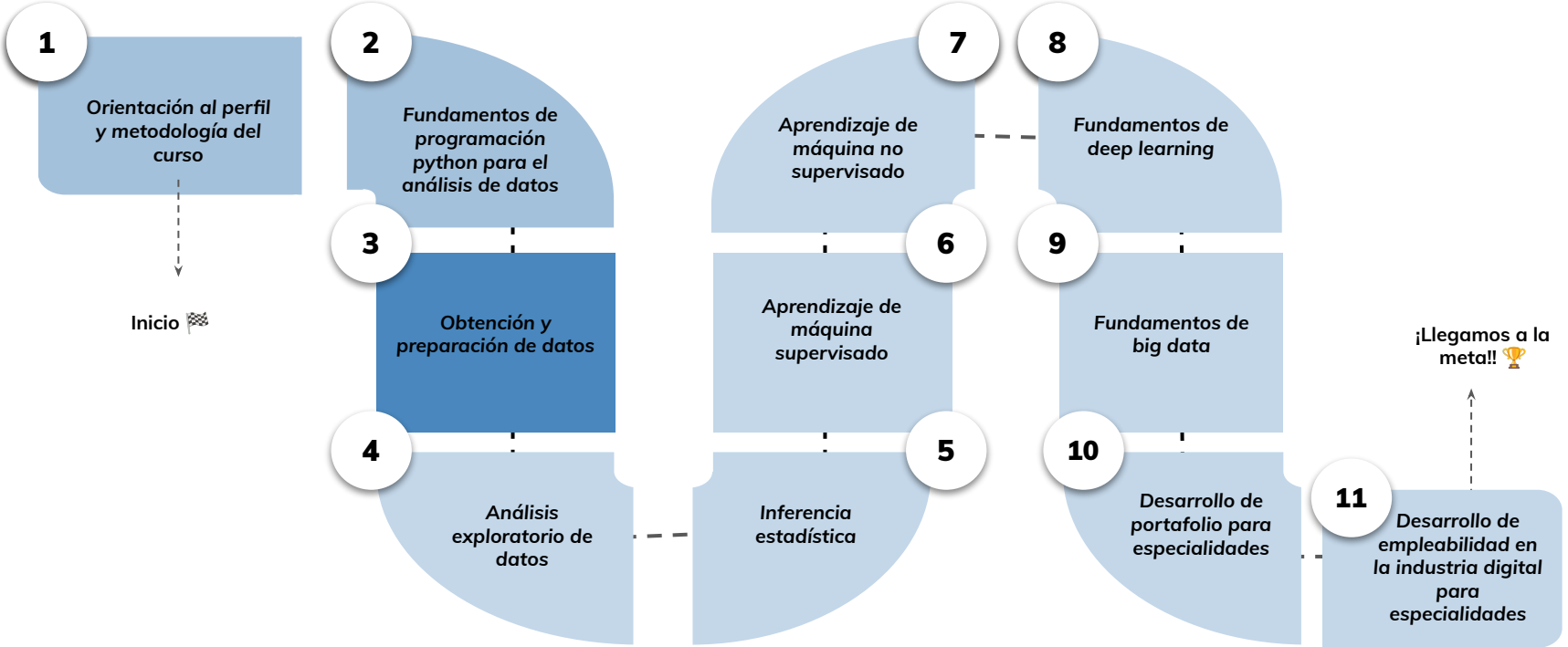
La librería numpy - Parte II

Aprendizaje Esperado: Manipular estructuras de datos vectoriales y matriciales utilizando biblioteca numpy para resolver un problema.

alkemy

Hoja de ruta

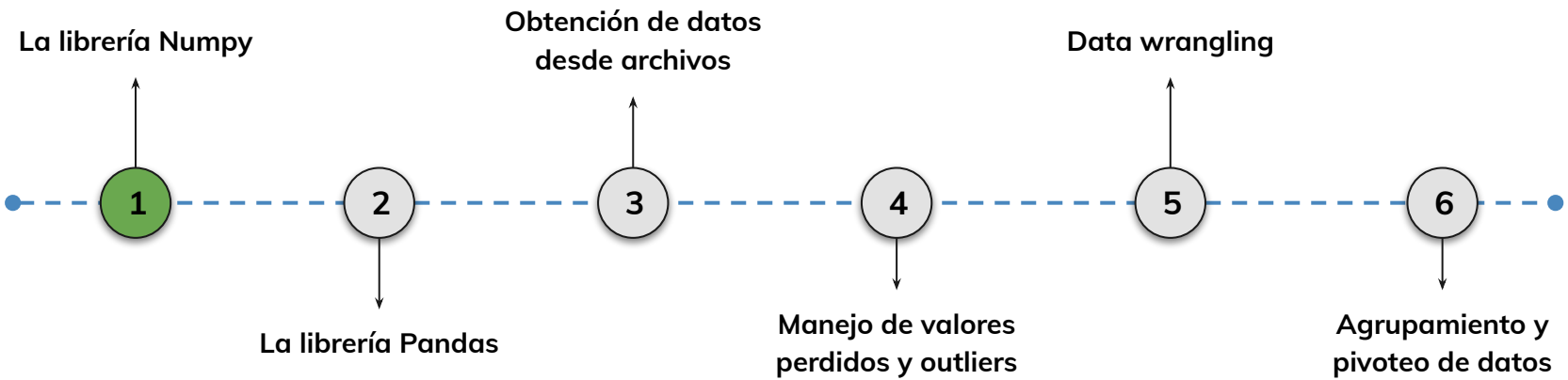
¿Cuáles skills conforman el programa? **Fundamentos de Ciencia de Datos**



alkemy

Roadmap de lecciones

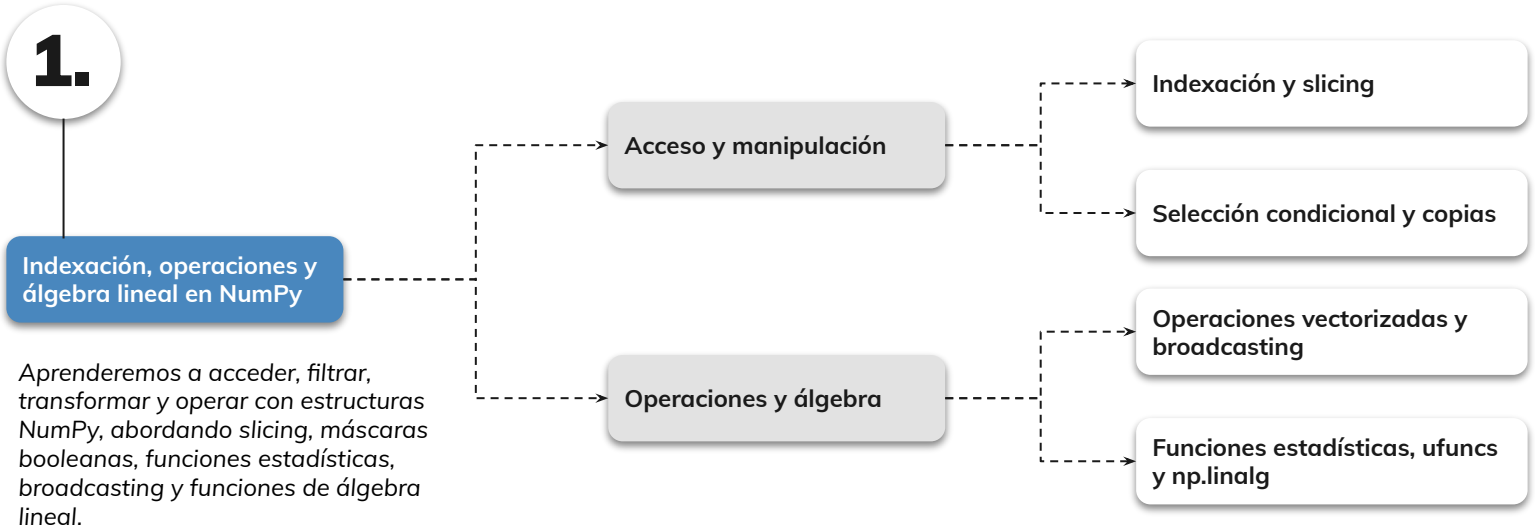
*¿Cuáles **lecciones** estaremos estudiando en este módulo?*



alkemy

Learning Path

¿Cuáles temas trabajaremos hoy?



alkemy

Objetivos de aprendizaje

¿Qué aprenderás?

- Usar slicing e indexación para extraer y transformar estructuras
- Aplicar condiciones lógicas para filtrar datos en arreglos
- Distinguir entre referencias y copias
- Utilizar funciones matemáticas, estadísticas y de álgebra lineal
- Ejecutar operaciones avanzadas sin bucles, maximizando rendimiento

Repaso clase anterior

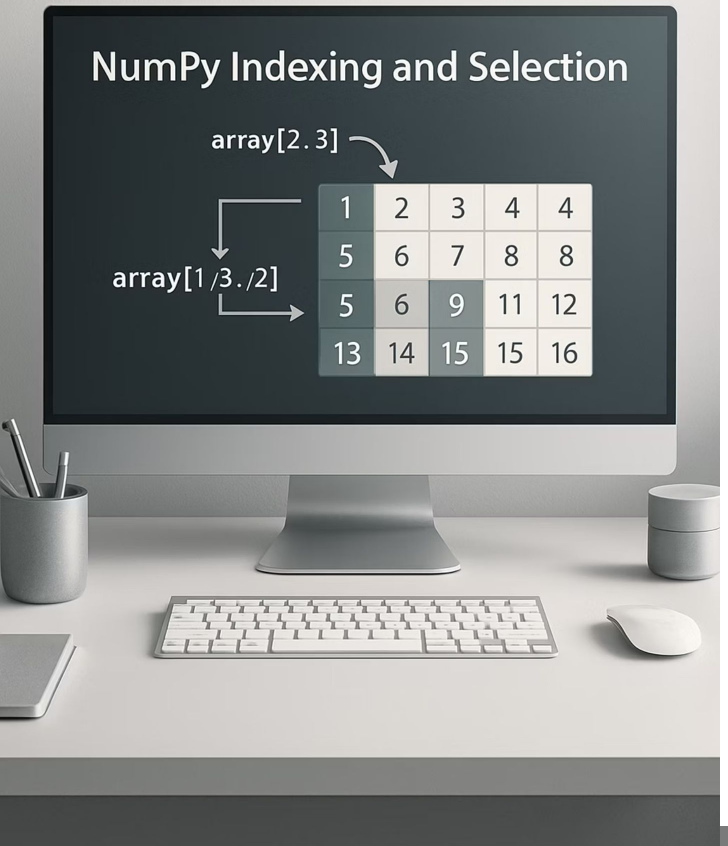
¿Quedó alguna duda?

En la clase anterior trabajamos 📚:

- → *Comprensión de qué es la librería NumPy y su importancia en Python*
- → *Creación de vectores y matrices utilizando np.array()*
- → *Aplicación de operaciones matemáticas sin bucles mediante vectorización*
- → *Uso de funciones como arange, linspace, zeros, ones, reshape y eye para generar y transformar arreglos*

La librería numpy

alkemy



Indexación y Selección

NumPy ofrece métodos potentes para acceder y seleccionar elementos específicos dentro de arreglos, facilitando la manipulación de datos.

alkemy

Selección de Elementos de un Arreglo

NumPy permite acceder a elementos individuales de un arreglo utilizando índices, similar a las listas en Python. En un vector unidimensional, el acceso se realiza indicando la posición del elemento.

```
import numpy as np
vector = np.array([10, 20, 30, 40, 50])
print(vector[2])  # Output: 30
```

Indexación en Matrices

En arreglos bidimensionales (matrices), se deben especificar dos índices: el de la fila y el de la columna.

```
# Acceder al elemento en la fila 1, columna 2
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
element = matrix[1, 2]  # Valor: 6
```

Slicing en NumPy

Definición

También se pueden seleccionar porciones del arreglo utilizando el operador : (slicing).

Sintaxis

La sintaxis es similar a la de las listas de Python: array[inicio:fin:paso].

Dimensiones

En matrices, se puede hacer slicing en ambas dimensiones: matrix[1:3, 0:2].

Ejemplos de Slicing

```
# Vector
arr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
slice1 = arr[2:7]  # [2, 3, 4, 5, 6]
slice2 = arr[::2]  # [0, 2, 4, 6, 8]

# Matriz
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
rows_1_2 = matrix[1:3]  # Filas 1 y 2
cols_0_1 = matrix[:, 0:2]  # Columnas 0 y 1
```

Selección Condicional de Elementos

NumPy permite seleccionar elementos de un arreglo que cumplan ciertas condiciones, generando arreglos booleanos.

```
numeros = np.array([3, 7, 2, 8, 5])
mayores_a_5 = numeros[numeros > 5]
print(mayores_a_5)  # Output: [7 8]
```

Combinación de Condiciones

También se pueden combinar múltiples condiciones usando operadores lógicos como & (AND) y | (OR).

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

# Elementos mayores que 2 Y menores que 7
condition = (arr > 2) & (arr < 7)
result = arr[condition]  # [3, 4, 5, 6]
```

Aplicaciones de la Selección Condicional

Filtrado de Datos

La selección condicional es fundamental para filtrar datos según criterios específicos.

Limpieza de Datos

Permite identificar y tratar valores atípicos o faltantes en conjuntos de datos.

Segmentación

Facilita la segmentación de datos para análisis específicos o visualizaciones.

Referencia y Copia de Arreglos

Cuando se asigna un arreglo a una nueva variable en NumPy, no se crea una copia, sino una referencia al mismo objeto en memoria.

```
original = np.array([1, 2, 3, 4])
copia_ref = original  # No es una copia real
copia_ref[0] = 99
print(original)  # Output: [99  2  3  4]
```

Creación de Copias

Para crear una copia real, se debe usar copy().

```
# Referencia (no copia)
arr = np.array([1, 2, 3, 4])
ref = arr  # Referencia al mismo arreglo

# Copia real
copy_arr = arr.copy()  # Nueva copia independiente
```

Implicaciones de Referencias vs. Copias

Modificaciones

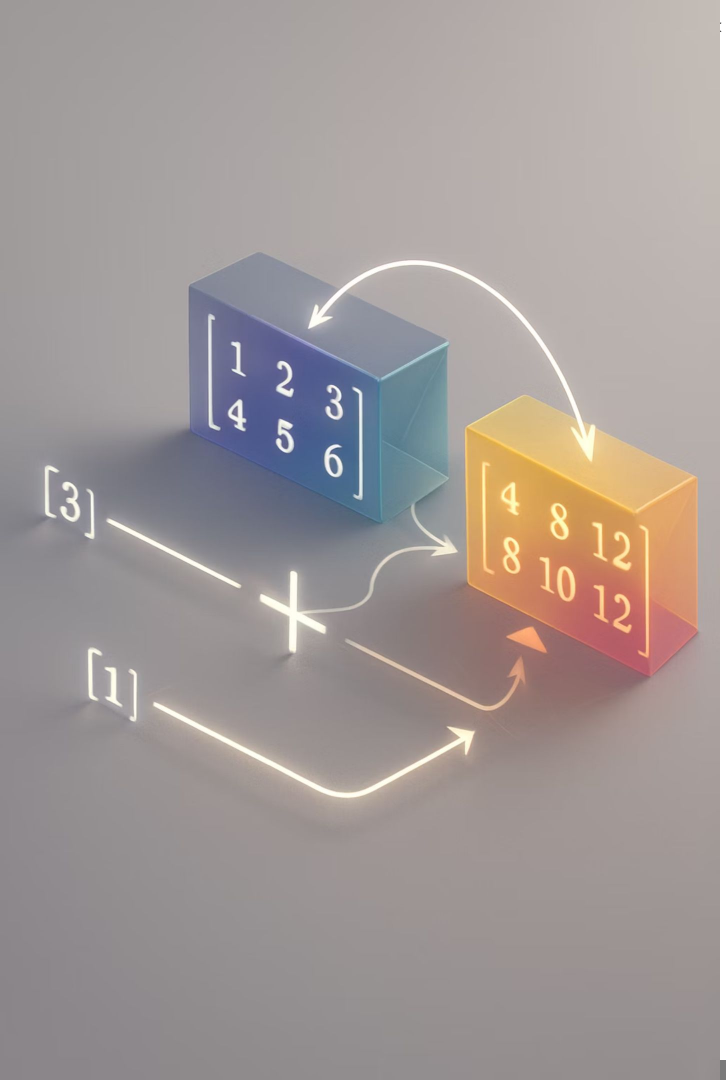
Los cambios en una referencia afectan al arreglo original, mientras que los cambios en una copia no.

Memoria

Las referencias son más eficientes en términos de memoria, pero pueden causar efectos secundarios no deseados.

Rendimiento

Crear copias consume más recursos pero garantiza la integridad de los datos originales.



Operaciones

NumPy proporciona una amplia gama de operaciones matemáticas optimizadas para trabajar con arreglos, permitiendo cálculos eficientes sin necesidad de bucles.

Operaciones entre Arreglos

NumPy permite realizar operaciones aritméticas directamente entre arreglos del mismo tamaño.

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

# Operaciones elemento a elemento
suma = a + b  # [5, 7, 9]
resta = a - b  # [-3, -3, -3]
producto = a * b  # [4, 10, 18]
division = a / b  # [0.25, 0.4, 0.5]
```

Broadcasting

Definición

Si los arreglos tienen dimensiones compatibles, NumPy aplica broadcasting, replicando los valores de menor dimensión para ajustarlos.

Reglas

Las dimensiones deben ser iguales o una de ellas debe ser 1 para que el broadcasting funcione.

Eficiencia

El broadcasting permite operaciones eficientes sin necesidad de crear copias físicas de los datos.

Ejemplo de Broadcasting

```
# Vector y escalar
arr = np.array([1, 2, 3, 4])
result = arr + 10  # [11, 12, 13, 14]

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

Operaciones con Escalares

Definición

Cuando se realizan operaciones con un escalar, esta se aplica a todos los elementos del arreglo.

Ejemplos

Multiplicación por escalar, suma de constante, elevación a potencia.

Eficiencia

Estas operaciones son altamente optimizadas en NumPy, mucho más rápidas que los bucles en Python.

Ejemplos de Operaciones con Escalares

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

# Operaciones con escalares
suma = arr + 5  # [6, 7, 8, 9]
producto = arr * 2  # [2, 4, 6, 8]
potencia = arr ** 2  # [1, 4, 9, 16]
division = arr / 2  # [0.5, 1, 1.5, 2]
```

Aplicando Funciones a un Arreglo

NumPy proporciona funciones matemáticas para aplicar a cada elemento de un arreglo sin necesidad de bucles.

```
array = np.array([1, 4, 9, 16])
raiz_cuadrada = np.sqrt(array)
logaritmo = np.log(array)
seno = np.sin(array)

print(raiz_cuadrada)  # Output: [1. 2. 3. 4.]
print(logaritmo)  # [0. 1.38629436 2.19722458 2.77258872]
print(seno)  # [0.84147098  -0.7568025  0.41211849 -0.28790332]
```

Funciones Matemáticas en NumPy

Trigonométricas

np.sin(), np.cos(), np.tan(), np.arcsin(), np.arccos(), np.arctan()

Exponenciales y Logarítmicas

np.exp(), np.log(), np.log10(), np.sqrt()

Redondeo

np.round(), np.floor(), np.ceil()

Ejemplos de Funciones Matemáticas

```
arr = np.array([0, np.pi/4, np.pi/2, np.pi])

# Aplicar funciones
senos = np.sin(arr)  # [0, 0.7071, 1, 0]
logaritmos = np.log(np.array([1, 2, 3, 4]))
raices = np.sqrt(np.array([1, 4, 9, 16]))  # [1, 2, 3, 4]
```

Funciones Estadísticas

Estas funciones optimizan el procesamiento de datos en cálculos matemáticos avanzados.

Ejemplos de Funciones Estadísticas

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

# Funciones estadísticas
media = np.mean(arr)  # 3.0
mediana = np.median(arr)  # 3.0
desviacion = np.std(arr)  # ~1.41
minimo = np.min(arr)  # 1
maximo = np.max(arr)  # 5
suma = np.sum(arr)  # 15
```

Funciones de Agregación

Por Eje

Las funciones de agregación pueden aplicarse a lo largo de un eje específico en matrices multidimensionales.

Ejemplos

np.sum(matrix, axis=0) suma columnas, np.sum(matrix, axis=1) suma filas.

Aplicaciones

Útiles para calcular estadísticas por grupos o categorías en análisis de datos.

Ejemplos de Agregación por Eje

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

# Suma por filas (axis=1)
row_sums = np.sum(matrix, axis=1)  # [6, 15, 24]

# Suma por columnas (axis=0)
col_sums = np.sum(matrix, axis=0)  # [12, 15, 18]

# Media por filas
row_means = np.mean(matrix, axis=1)  # [2, 5, 8]
```

Funciones Universales (ufuncs)

Definición

Las ufuncs son funciones que operan elemento a elemento en arreglos NumPy de manera optimizada.

Ventajas

Son mucho más rápidas que las funciones equivalentes en Python puro, ya que están implementadas en C.

Tipos

Existen ufuncs unarias (un solo input) y binarias (dos inputs).

Ejemplos de ufuncs

```
# Unarias
arr = np.array([-1, 0, 1, 2])
abs_values = np.abs(arr)  # [1, 0, 1, 2]
squared = np.square(arr)  # [1, 0, 1, 4]

# Binarias
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
maximum = np.maximum(a, b)  # [4, 5, 6]
power = np.power(a, b)  # [1, 32, 729]
```

Álgebra Lineal con NumPy

Operaciones Matriciales

NumPy proporciona funciones para operaciones avanzadas de álgebra lineal como determinantes, inversas y descomposiciones.

Módulo linalg

El submódulo np.linalg contiene funciones especializadas para álgebra lineal.

Aplicaciones

Estas operaciones son fundamentales en machine learning, procesamiento de señales y optimización.

Ejemplos de Álgebra Lineal

```
from numpy import linalg as LA

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

# Determinante
det_A = LA.det(A)  # -2.0

# Inversa
inv_A = LA.inv(A)

# Autovalores
eigenvalues = LA.eigvals(A)

# Norma
norm = LA.norm(A)
```

Operaciones Avanzadas

Transformada de Fourier

NumPy incluye funciones para calcular transformadas de Fourier, útiles en procesamiento de señales e imágenes.

Operaciones con Máscaras

Permite aplicar operaciones solo a elementos que cumplen ciertas condiciones mediante np.where() y np.select().

Manipulación de Ejes

Funciones como np.transpose(), np.swapaxes() y np.moveaxis() permiten reorganizar las dimensiones de un arreglo.

Integración con Otras Bibliotecas

Pandas

NumPy se integra perfectamente con Pandas para análisis de datos estructurados y series temporales.

Matplotlib

Los arreglos de NumPy son la base para la visualización de datos con Matplotlib.

Scikit-learn

Los algoritmos de machine learning en Scikit-learn operan sobre arreglos NumPy.

Optimización de Rendimiento

Consejos para Mejorar el Rendimiento

- Evitar bucles en Python, usar operaciones vectorizadas
- Utilizar vistas en lugar de copias cuando sea posible
- Aprovechar las funciones de agregación con el parámetro axis
- Usar dtypes apropiados para optimizar memoria
- Considerar bibliotecas como Numba para código crítico

Live Coding

¿En qué consistirá la Demo?

Simularemos un pequeño modelo de machine learning desde el punto de vista algebraico, usando np.linalg para resolver un sistema de ecuaciones lineales representando un modelo de regresión lineal simple.

1. *Crear una matriz X de características y un vector y de etiquetas*
2. *Calcular la transpuesta de X*
3. *Resolver los coeficientes con la fórmula de mínimos cuadrados*
4. *Predecir los valores de salida y_hat*
5. *Calcular el error cuadrático medio (MSE)*

Tiempo: 20 minutos

#Momentode Preguntas...

¿?

¿Qué diferencia hay entre .copy() y una simple asignación?

¿?

¿Qué hace np.where() y en qué se diferencia de np.select()?

¿?

¿Cómo mejora el rendimiento el uso de ufuncs respecto a bucles?

¿?

¿Qué ocurre si intento multiplicar dos matrices incompatibles en dimensiones?

Momento:

Time-out!

⏰5 -10 min.

Ejercicio N° 1

Análisis y transformación de una matriz con selección condicional

Análisis y transformación de una matriz con selección condicional

Contexto: 🙌

Trabajas con datos de temperaturas diarias de 5 ciudades durante una semana. Estos datos deben limpiarse, analizarse y procesarse para generar un reporte con estadísticas útiles para el equipo de analítica.

Consigna: ✍️

Utilizando NumPy, deberás:

- Simular una matriz 5x7 con temperaturas aleatorias entre 10°C y 40°C
- Identificar las temperaturas que superan los 30°C
- Reemplazar los valores inferiores a 15°C por el valor 15
- Calcular la media de temperaturas por ciudad (por fila) y por día (por columna)
- Determinar cuál es la ciudad con la mayor temperatura promedio

Tiempo 🕐**:** *45 min*

Análisis y transformación de una matriz con selección condicional

Paso a paso: ⚙️

1. Crea una matriz de 5 filas y 7 columnas con valores enteros aleatorios entre 10 y 40
2. Selecciona las temperaturas mayores a 30°C utilizando operadores lógicos
3. Establece un umbral mínimo: los valores menores a 15 deben ser reemplazados por 15.
4. Calcula promedios por ciudad y por día
5. Compara los promedios por ciudad y encuentra cuál es la ciudad con el promedio más alto.

¿Alguna consulta?

alkemy

Resumen

¿Qué logramos en esta clase?

- ✓ *Aprendimos a usar slicing e indexación avanzada*
- ✓ *Aplicamos selección condicional sobre arreglos*
- ✓ *Diferenciamos referencias y copias*
- ✓ *Usamos np.mean, np.std, np.where, np.linalg*
- ✓ *Realizamos álgebra lineal sin bucles y de forma optimizada*

¡Ponte a prueba!

Momento de ejercitación

Te invitamos a aprovechar esta última sección del espacio sincrónico para realizar de manera individual las ***actividades disponibles en la plataforma.*** *Estas propuestas son clave para afianzar lo trabajado y* ***forman parte obligatoria del recorrido de aprendizaje.***

💪 Análisis de caso ——— 💪 Selección Múltiple

💪 Comprensión lectora

Si al resolverlas surge alguna duda, compartela o tráela al próximo encuentro sincrónico.

alkemy

¡Muchas gracias!

alkemy