Obtención y Preparación

de Datos

Sesión 1

¿Qué es NumPy?

- NumPy (Numerical Python) es una librería fundamental para la computación numérica en Python.

- Su estructura de datos principal es el array (arreglo), que permite trabajar con vectores, matrices y arreglos multidimensionales.

- Es la base de muchas librerías como Pandas, SciPy, Matplotlib y TensorFlow.

Ventajas de Usar NumPy

✅ Más rápido que listas de Python (implementado en C).

✅ Soporta operaciones vectorizadas sin bucles explícitos.

✅ Funciones matemáticas optimizadas (trigonométricas, logarítmicas, estadísticas, álgebra lineal).

✅ Ideal para grandes volúmenes de datos (machine learning, procesamiento de imágenes).

✅ Interoperabilidad con otras librerías como Pandas y SciPy.

✅ Indexación y slicing avanzado para manipulación flexible de datos.

Creación de Arreglos en NumPy

NumPy permite crear diferentes tipos de arreglos, desde vectores simples hasta matrices y arreglos multidimensionales. Estos arreglos permiten almacenar y manipular datos numéricos de manera eficiente, con una sintaxis clara y optimizada para cálculos matemáticos y científicos.

1

Vectores

Los vectores son arreglos de una sola dimensión, similares a una lista en Python, pero con la ventaja de ser más eficientes en términos de almacenamiento y velocidad de procesamiento.

Ejemplo de creación de un vector en NumPy:

```
import numpy as np

# Crear un vector con valores específicos
vector = np.array([1, 2, 3, 4, 5])
print("Vector:\n", vector)
```

Creación de Arreglos en NumPy

1

Vectores

Características de los vectores en NumPy:

- Se crean con np.array(), pasándole una lista de valores.
- Son la base de operaciones matemáticas optimizadas en NumPy.
- Soportan operaciones matemáticas directas, como suma y multiplicación por escalares.

Ejemplo de operaciones con vectores:

```
vector2 = vector * 2  # Multiplicar cada elemento por 2
print("Vector multiplicado por 2:\n", vector2)
```

Creación de Arreglos en NumPy

2

Matrices (2D)

Las matrices son arreglos de dos dimensiones (filas y columnas), similares a una tabla o una hoja de cálculo. Se utilizan en múltiples áreas, como álgebra lineal, procesamiento de imágenes y análisis de datos.

Ejemplo de operaciones con vectores:

```
# Crear una matriz 3x3
matriz = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("Matriz:\n", matriz)
```

Creación de Arreglos en NumPy

2

Matrices (2D)

Características de las matrices en NumPy:

- Se crean con np.array() pasando una lista de listas.
- Cada lista interna representa una fila de la matriz.
- Se pueden manipular con operaciones como suma, multiplicación y transposición.

Ejemplo de operaciones con matrices:

```
# Transpuesta de la matriz
print("Matriz Transpuesta:\n", matriz.T)

# Suma de matrices
matriz2 = np.array([[1, 1, 1], [1, 1, 1], [1, 1, 1]])
print("Suma de Matrices:\n", matriz + matriz2)
```

Creación de Arreglos en NumPy

3

Arreglos Multidimensionales (Más de 2 Dimensiones)

NumPy permite crear arreglos con más de dos dimensiones, ideales para aplicaciones avanzadas como procesamiento de imágenes, redes neuronales y datos científicos complejos.

Ejemplo de creación de un arreglo 3D (tensor) en NumPy:

```
# Crear un arreglo tridimensional (3D)
tensor = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
print("Arreglo 3D:\n", tensor)
```

Funciones Preconstruidas de Creación

NumPy proporciona diversas funciones predefinidas que facilitan la creación de arreglos sin necesidad de ingresar manualmente los valores. Estas funciones permiten generar arreglos con patrones específicos, lo que resulta útil en cálculos científicos, simulaciones y procesamiento de datos.

1

Arange(): Arreglos con valores definidos
Sintaxis:

```
numpy.arange(inicio, fin, paso)
```

- inicio: Valor inicial del rango (opcional, por defecto es 0).
- fin: Valor hasta donde se generarán los números (sin incluirlo).
- paso: Incremento entre los valores generados (opcional, por defecto es 1).

Funciones Preconstruidas de Creación

1

Arange(): Arreglos con valores definidos

Ejemplo

```
import numpy as np
arr = np.arange(1, 10, 2)  # Genera [1, 3, 5, 7, 9]
print(arr)
```

Funciones Preconstruidas de Creación

2 **Matrices de ceros y unos**

Ejemplo

```
ceros = np.zeros((3,3))  # Matriz 3x3 de ceros
unos = np.ones((2,2))  # Matriz 2x2 de unos
```

3 **Vector con Distribución de Puntos (linspace())**

Sintaxis

```
numpy.linspace(inicio, fin, num_puntos)
```

Funciones Preconstruidas de Creación

3 **Vector con Distribución de Puntos (linspace())**

Ejemplo

```
puntos = np.linspace(0, 10, 5)  # 5 valores entre 0 y 10
print(puntos)
```

4 **Matriz de Identidad**

Ejemplo

```
identidad = np.eye(4)  # Matriz identidad 4x4
print(identidad)
```

Funciones Preconstruidas de Creación

5 **Matriz Aleatoria**

Ejemplo

```
aleatoria = np.random.rand(3,3)  # Matriz 3x3 de valores aleatorios
print(aleatoria)
```

Para valores enteros en un rango especifico, usamos randint()

Ejemplo

```
np.random.randint(1, 100, (2, 4))  # Matriz 2x4 con enteros entre 1 y 100
```

Funciones Preconstruidas de Creación

6 **Redimensionado de un Arreglo**

Ejemplo

```
arreglo = np.arange(9)
arreglo_reshaped = arreglo.reshape(3,3)
print(arreglo_reshaped)
```

7 **Indexación y Selección en NumPy**

- Selección de elementos

Ejemplo

```
arreglo = np.array([10, 20, 30, 40, 50])
print(arreglo[2])  # Índice 2 (tercer elemento)
```

Funciones Preconstruidas de Creación

7

Indexación y Selección en NumPy

- Selección de elementos para matrices, se usa la notación [fila, columna]

Ejemplo

```
matriz = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(matriz[1, 2])  # Accede al valor 6
```

- Selección condicional

Ejemplo

```
arr = np.array([10, 20, 30, 40, 50])
print(arr[arr > 25])  # Devuelve [30, 40, 50]
```

Funciones Preconstruidas de Creación

8

Referencia y Copia de Arreglos

- Referencia: Modifica el arreglo original

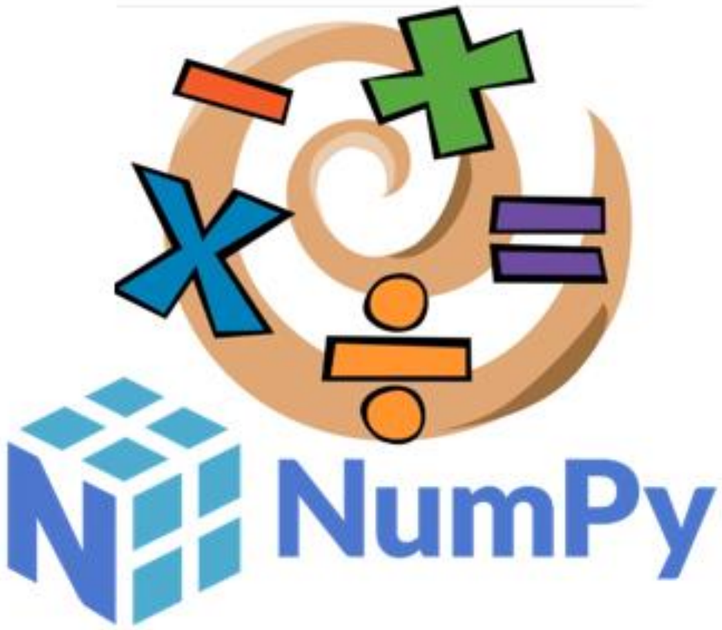
Ejemplo

```
original = np.array([1,2,3])
modifica = original # modifica apunta a los mismos datos de original
modifica[0] = 99
print(original) # modifica también se modifica
```

- Copia: Crea un nuevo arreglo independiente

Ejemplo

```
copia = original.copy()
copia[0] = 99
print(original) #[1,2,3]
print(original) #[99,2,3]
```



Operaciones con NumPy

Operaciones Entre Arreglos

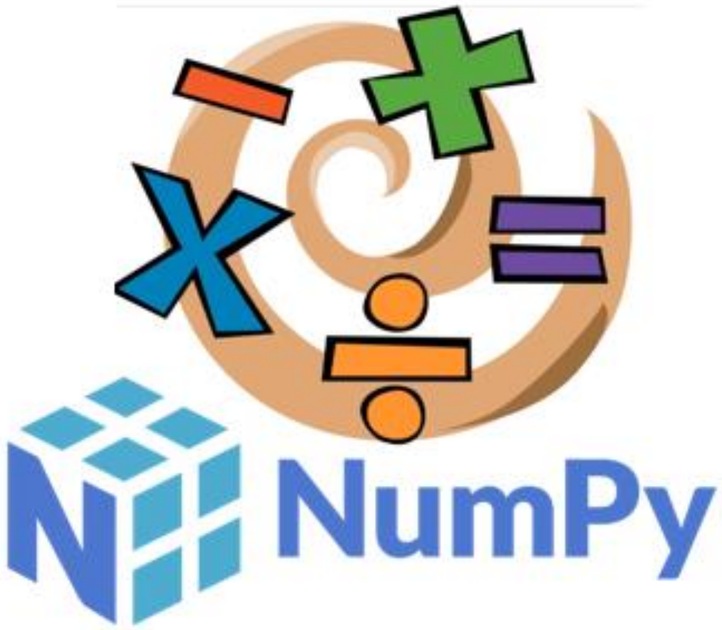
Ejemplo

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
print(arr1 + arr2)  # [5 7 9]
```

Operaciones con Escalares

Ejemplo

```
arreglo = np.array([2, 4, 6])
print(arreglo * 2)  # [4 8 12]
```



Operaciones con NumPy

Otras Operaciones

Ejemplo

```
resta = arr1 - arr2
producto = arr1 * arr2
division = arr1 / arr2
```

Aplicando Funciones a un Arreglo

NumPy incluye funciones matemáticas optimizadas para operaciones con arreglos.

Ejemplo

```
arr = np.array([1, 4, 9, 16])
raiz = np.sqrt(arr)  # Calcula la raíz cuadrada de cada elemento
seno = np.sin(arr)   # Calcula el seno de cada elemento
logaritmo = np.log(arr)  # Calcula el logaritmo natural
```

Conclusión y Recursos

Resumen

1

- NumPy es esencial para eficiencia y escalabilidad en ciencia de datos.
- Dominar sus funciones acelera el análisis y reduce código redundante.

Recursos Recomendados

2

Documentación oficial: numpy.org/doc
Libro: "Python for Data Analysis" (Wes McKinney).

Preguntas

Sección de preguntas

Obtención y Preparación

de Datos

Continúe con las actividades