TIC

Actividad 1

Introducción a NumPy

Universidad Tecnológica de Pereira

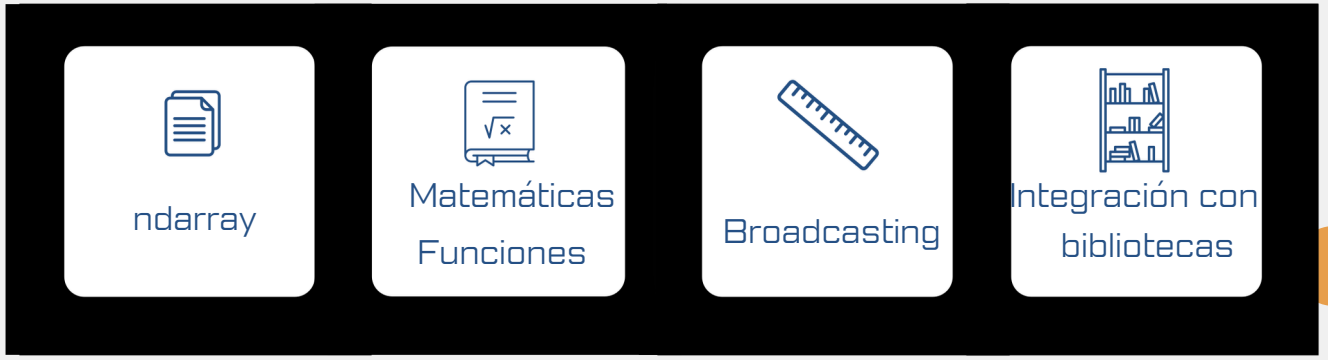
faceIT

La documentacion o cial se puede ver en:

https://numpy.org/doc/stable/user/

¿Qué3 es NumPy?

NumPy, que signi ca "Numerical Python", es una biblioteca de código abierto en Python que proporciona soporte para matrices y matrices multidimensionales, así como funciones matemáticas para trabajar con estas estructuras de datos. Es una de las bibliotecas más fundamentales y ampliamente utilizadas en el ámbito de la ciencia de datos, la ingeniería y la computación cientí ca debido a su e ciencia en el manejo de datos numéricos y su facilidad de uso. Entre las características principales de NumPy se encuentran:



Integración con otras bibliotecas:

NumPy se integra de manera uida con otras bibliotecas populares en el ecosistema de Python, como SciPy (biblioteca de funciones matemáticas avanzadas), Matplotlib (biblioteca de visualización de datos) y Pandas (biblioteca de análisis de datos), lo que lo convierte en una parte central de muchas aplicaciones cientí cas y de análisis de datos en Python.

Broadcasting:

NumPy ofrece una funcionalidad llamada broadcasting que permite realizar operaciones entre ndarrays de diferentes formas y tamaños de manera intuitiva y e ciente. Esta característica es útil para realizar cálculos en matrices de diferentes dimensiones sin la necesidad de hacer explícitamente la expansión de las dimensiones.

Funciones matemáticas:

NumPy proporciona un amplio conjunto de funciones matemáticas y operaciones aritméticas para trabajar con ndarrays. Estas funciones incluyen operaciones básicas como suma, resta, multiplicación y división, así como funciones más avanzadas como cálculos estadísticos, álgebra lineal, transformadas de Fourier, entre otras.

ndarray

NumPy introduce el tipo de datos ndarray (N-dimensional array), que es una estructura de datos e ciente y exible para representar matrices y matrices multidimensionales en Python. Los ndarrays son homogéneos, lo que signi ca que todos los elementos deben ser del mismo tipo de datos, lo que los hace más e cientes en términos de memoria y rendimiento en comparación con las listas de Python estándar.

NumPy es una herramienta fundamental para el procesamiento y análisis de datos numéricos en Python, proporcionando una base sólida para el desarrollo de aplicaciones en una amplia gama de campos, desde la investigación cientí ca hasta el aprendizaje automático y la ingeniería.

Importancia en IA: Discutir por qué NumPy es crucial en el contexto de la Inteligencia Artificial

E ciencia en Operaciones Matriciales:

NumPy proporciona una implementación e ciente de estructuras de datos tipo matriz y operaciones matriciales. Las operaciones vectorizadas en NumPy permiten realizar cálculos sobre matrices de manera rápida y optimizada, lo cual es esencial para algoritmos de aprendizaje automático que involucran manipulación intensiva de datos.

Importancia en IA: Discutir por qué NumPy es crucial en el contexto de la Inteligencia Artificial

Manejo E ciente de Grandes Conjuntos de Datos:

Las capacidades de NumPy para manejar grandes conjuntos de datos son cruciales en la Inteligencia Arti cial, donde a menudo se trabaja con datos complejos y extensos. NumPy optimiza el uso de la memoria y ofrece funciones para realizar operaciones en conjuntos de datos masivos de manera e ciente.

Importancia en IA: Discutir por qué NumPy es crucial en el contexto de la Inteligencia Artificial

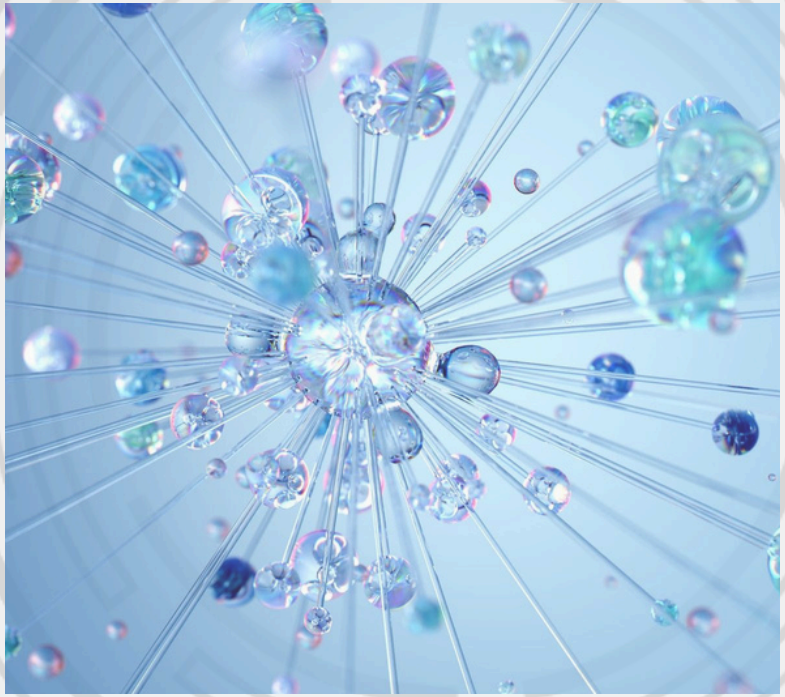
Integración con Otras Bibliotecas de IA:

NumPy sirve como base para numerosas bibliotecas y frameworks de IA. Bibliotecas como Pandas, SciPy y Scikit-learn dependen de NumPy para sus operaciones internas, creando así un ecosistema cohesivo que facilita la interoperabilidad entre diferentes herramientas de Inteligencia Arti cial.

Importancia en IA: Discutir por qué NumPy es crucial en el contexto de la Inteligencia Artificial

Compatibilidad con Modelos y Algoritmos de Aprendizaje Automático:

Dado que muchos algoritmos de aprendizaje automático están basados en operaciones matriciales y manipulación de datos, NumPy se convierte en una herramienta esencial para implementar y ejecutar estos modelos de manera e ciente.



Importancia en IA: Discutir por qué NumPy es crucial en el contexto de la Inteligencia Artificial

Facilita la Experimentación y Desarrollo Rápido:

NumPy proporciona una interfaz simple y consistente para realizar operaciones matriciales y manipulación de datos. Esto facilita la experimentación y el desarrollo rápido de prototipos, permitiendo a los profesionales de la IA probar ideas y ajustar modelos de manera e caz.

Instalación de NumPy

Para instalar NumPy en tu sistema, puedes utilizar pip, el

Abrir la Terminal o el Símbolo del Sistema:

En Windows, puedes abrir el Símbolo del Sistema buscándolo en el menú de inicio. En macOS o Linux, puedes abrir la Terminal desde el lanzador de aplicaciones o utilizando el atajo de teclado Ctrl + Alt + T.

Ejecutar el Comando de Instalación:

Escribe el siguiente comando y presiona Enter:

```
pip install numpy
```

Esperar a que la Instalación se Complete:

Pip descargará e instalará NumPy junto con cualquier otra dependencia necesaria. Dependiendo de tu conexión a internet y la velocidad de tu sistema, la instalación puede tardar unos minutos.

Verifficar la Instalación:

Después de que la instalación haya nalizado, puedes veri car si NumPy se instaló correctamente ejecutando un script de Python que importe NumPy:

```
import numpy as np
print(np.__version__)
```

Esto imprimirá la versión de NumPy instalada en tu sistema.

Con estos pasos, deberías poder instalar NumPy en tu sistema sin problemas. Si tienes algún problema durante la instalación, asegúrate de tener una conexión a internet estable y los

permisos necesarios para instalar paquetes en tu sistema.

Qué son y cómo se crean los arrays y arrays de NumPy

Documetacion o cial: **https://numpy.org/doc/stable/user/basics.creation.html**

Arrays:
Estructuras de datos que almacenan una colección de valores del mismo tipo.

Ndarrays:
Implementación de arrays en la biblioteca NumPy. Ofrecen mayor e ciencia y funcionalidades matemáticas avanzadas.

Los ndarrays (arreglos n-dimensionales) son la estructura fundamental de datos en NumPy juegan un papel crucial en la manipulación e ciente de datos numéricos en Python De nición y Creación: Un ndarray es un conjunto homogéneo y multidimensional de elementos del mismo tipo. Se puede crear fácilmente utilizando la función numpy.array().

Creación de ndarrays

1. A partir de listas:

```
import numpy as np

# Lista de números
numeros = [1, 2, 3, 4, 5]

# Creación de un ndarray a partir de la lista
ndarray_numeros = np.array(numeros)

print(ndarray_numeros)
# Salida: [1 2 3 4 5]
```

Creación de ndarrays

2. Especificando dimensiones y valores:

```
# Creación de un ndarray de 2x3 con ceros
matriz_ceros = np.zeros((2, 3))

# Creación de un ndarray de 3x3 con unos
matriz_unos = np.ones((3, 3))

# Creación de un ndarray con valores especificos
matriz_personalizada = np.array([[1, 2, 3], [4, 5, 6]])

print(matriz_ceros)
# Salida: [[0. 0. 0.]
#  [0. 0. 0.]]

print(matriz_unos)
# Salida: [[1. 1. 1.]
#  [1. 1. 1.]
#  [1. 1. 1.]]

print(matriz_personalizada)
# Salida: [[1 2 3]
#  [4 5 6]]
```

Creación de ndarrays

3. Funciones de NumPy: **np.arange(inicio, n, paso): Crea un ndarray con un rango de valores.**

Parámetros: **inicio: Valor inicial (incluido). n: Valor nal (excluido). paso: Diferencia entre valores consecutivos.**

```python
import numpy as np

# Creación de un ndarray con valores del 0 al 9 (sin incluir el 10)
numeros_1 = np.arange(0, 10)
print(numeros_1)
# Salida: [0 1 2 3 4 5 6 7 8 9]

# Creación de un ndarray con valores del 1 al 5 de 2 en 2
numeros_2 = np.arange(1, 6, 2)
print(numeros_2)
# Salida: [1 3 5]

# Creación de un ndarray con valores del 0 al 4 (sin incluir el 4) de 0.5 en 0.5
numeros_3 = np.arange(0, 4, 0.5)
print(numeros_3)
# Salida: [0.  0.5 1.  1.5 2.  2.5 3.  3.5]
```

Creación de ndarrays

np.linspace(inicio, n, num_puntos): Crea un ndarray con valores espaciados uniformemente. Parámetros: inicio: Valor inicial (incluido). n: Valor nal (incluido). num_puntos: Número de puntos en el ndarray.

Python

```python
# Creación de un ndarray con 10 valores espaciados uniformemente entre 0 y 1
puntos_1 = np.linspace(0, 1, 10)
print(puntos_1)
# Salida: [0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]

# Creación de un ndarray con 5 valores espaciados uniformemente entre 2 y 10
puntos_2 = np.linspace(2, 10, 5)
print(puntos_2)
# Salida: [ 2.  4.  6.  8. 10.]
```

Creación de ndarrays

np.random.rand(dimensiones):Cr ea un ndarray con números aleatorios entre 0 y 1.

Parámetros:

dimensiones:Tupla que indica la forma del ndarray.

Python

```python
# Creación de un ndarray de 3x3 con números aleatorios entre 0 y 1
matriz_aleatoria_1 = np.random.rand(3, 3)
print(matriz_aleatoria_1)
# Salida:
# [[0.42345321 0.74320123 0.32098743]
#  [0.12930129 0.98734232 0.43209843]
#  [0.54320987 0.23453201 0.73290874]]

# Creación de un ndarray de 5 elementos con números aleatorios entre 0 y 1
vector_aleatorio_1 = np.random.rand(5)
print(vector_aleatorio_1)
# Salida: [0.73290874 0.29874321 0.43209874 0.12930129 0.98734232]
```

Características básicas de un ndarray en NumPy:

Dimensiones: **Número de subíndices necesarios para acceder a un elemento.**

Un ndarray 1D tiene una dimensión.

Ejemplo: array([1, 2, 3]).

Un ndarray 2D tiene dos dimensiones.
Ejemplo: array([[1, 2, 3], [4, 5, 6]]).

Un ndarray 3D tiene tres dimensiones.
Ejemplo: array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]]).

Características básicas de un ndarray en NumPy:

Forma (shape): **Tupla que indica la longitud de cada dimensión. Se puede acceder a la forma de un ndarray mediante la propiedad .shape.** Ejemplos: **Un ndarray 1D con 5 elementos tiene forma (5,).**

Un ndarray 2D con 3 las y 4 columnas tiene forma (3, 4).

Un ndarray 3D con 2 matrices de 3x4 tiene forma (2, 3, 4).

Características básicas de un ndarray en NumPy:

Tamaño (size): **Número total de elementos en el ndarray. Se puede acceder al tamaño de un ndarray mediante la propiedad .size.** Ejemplos: **Un ndarray 1D con 5 elementos tiene un tamaño de 5.**

Un ndarray 2D con 3 las y 4 columnas tiene un tamaño de 12.
Un ndarray 3D con 2 matrices de 3x4 tiene un tamaño de 24.

Características básicas de un ndarray en NumPy:

Tipo de dato: **Tipo de dato de los elementos del ndarray. Se puede acceder al tipo de dato de un ndarray mediante la propiedad .dtype.** Ejemplos: **Un ndarray con números enteros tiene un tipo de dato int32.**

Un ndarray con números de punto otante tiene un tipo de dato oat64.

Un ndarray con cadenas de texto tiene un tipo de dato str.

Características básicas de un ndarray en NumPy:

Indexación y selección de elementos: Se puede acceder a elementos especí cos de un ndarray utilizando una secuencia de índices entre corchetes. Selección de elementos: Se pueden seleccionar subconjuntos de un ndarray utilizando diferentes técnicas como slicing, máscaras y boolean indexing.

```python
import numpy as np

# Creación de un ndarray 2D
matriz = np.array([[1, 2, 3], [4, 5, 6]])

# Acceso a un elemento específico (fila 1, columna 2)
elemento = matriz[1, 2]
print(elemento)
# Salida: 6

# Selección de una fila (fila 0)
fila = matriz[0, :]
print(fila)
# Salida: [1 2 3]

# Selección de una columna (columna 1)
columna = matriz[:, 1]
print(columna)
# Salida: [2 5]
```

Operaciones Matriciales:

Los ndarrays facilitan las operaciones matriciales, lo que es esencial para la programación numérica e ciente. NumPy proporciona funciones para realizar operaciones de álgebra lineal directamente en los ndarrays.

Funciones y Métodos: **NumPy ofrece una amplia gama de funciones y métodos especí cos de los ndarrays para realizar operaciones estadísticas, algebraicas y de manipulación de datos.** Por ejemplo:

```
arr = np.array([1, 2, 3])
print(np.sum(arr))  # Salida: 6 - suma de todos los elemento
```

Funciones y Métodos en NumPy

Las funciones y métodos juegan un papel fundamental en la manipulación y análisis de arrays.

Métodos para la Manipulación de ndarrays:

NumPy proporciona una variedad de métodos para manipular ndarrays de diferentes maneras. Algunos de estos métodos incluyen: reshape(): Permite cambiar la forma de un array sin cambiar sus datos. atten(): Convierte un array multidimensional en un array unidimensional. transpose(): Permite cambiar el orden de las dimensiones de un array.

```python
import numpy as np

array = np.array([[1, 2, 3], [4, 5, 6]])
reshaped_array = array.reshape(3, 2)
flattened_array = array.flatten()
transposed_array = array.transpose()

print("Array Original:")
print(array)
print("Array Reshape:")
print(reshaped_array)
print("Array Flatten:")
print(flattened_array)
print("Array Transpose:")
print(transposed_array)
```

Funciones de Generación de Números Aleatorios:

NumPy ofrece funciones para generar números aleatorios que son útiles en simulaciones, análisis de datos y otras aplicaciones. Algunas de estas funciones incluyen: np.random.rand(): Genera números aleatorios entre 0 y 1 con una distribución uniforme. np.random.randint(): Genera números enteros aleatorios dentro de un rango especi cado. np.random.randn(): Genera números aleatorios con una distribución normal (gaussiana) con media 0 y desviación estándar 1.

```python
import numpy as np

numeros_aleatorios_uniformes = np.random.rand(3)
numeros_aleatorios_enteros = np.random.randint(1, 10, size=5)
numeros_aleatorios_normales = np.random.randn(2, 2)

print("Números Aleatorios Uniformes:")
print(numeros_aleatorios_uniformes)
print("Números Aleatorios Enteros:")
print(numeros_aleatorios_enteros)
print("Números Aleatorios Normales:")
print(numeros_aleatorios_normales)
```

Funciones de Generación de Números Aleatorios:

Operaciones Aritméticas:

En NumPy, realizar operaciones aritméticas e cientes en ndarrays es fundamental para la programación e ciente en el ámbito de la inteligencia arti cial.

Elemento por Elemento: **Puedes realizar operaciones aritméticas elemento por elemento en ndarrays. Esto signi ca que cada elemento del primer ndarray se combina con el correspondiente del segundo.**

Por ejemplo:

```
import numpy as np

arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

# Suma elemento por elemento
result = arr1 + arr2  # Salida: [5, 7, 9]
```

Funciones de Generación de Números Aleatorios:

Operaciones con Constantes: **Las operaciones con constantes se realizan automáticamente para cada elemento del ndarray.** Por ejemplo:

```
import numpy as np

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

# Multiplicación por una constante
result = 2 * arr  # Salida: [2, 4, 6]
```

Operaciones entre ndarrays:

Las operaciones entre ndarrays se realizan elemento por elemento. Es importante que los ndarrays involucrados tengan la misma forma o formas compatibles para el broadcasting. Por ejemplo:

```
import numpy as np

arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])

# Suma de matrices
result = arr1 + arr2  # Salida: [[6, 8], [10, 12]]
```

Funciones de Generación de Números Aleatorios:

Funciones Universales (ufuncs): **NumPy proporciona funciones universales (ufuncs) que operan e cientemente en ndarrays. Estas funciones realizan operaciones elemento por elemento y están optimizadas para el rendimiento. Ejemplos incluyen np.sin(), np.cos(), np.exp(), etc.**

Operaciones Matriciales: **NumPy proporciona funciones especí cas para realizar operaciones matriciales, como la multiplicación de matrices (np.dot() o @ en Python 3.5+). Estas operaciones son esenciales en algoritmos de aprendizaje automático y otras aplicaciones.**

Funciones de Generación de Números Aleatorios:

Realizar operaciones aritméticas e cientes en ndarrays es esencial para aprovechar al máximo la velocidad y e ciencia de NumPy en operaciones numéricas. Al entender estas técnicas, puedes escribir código más e ciente y conciso para tus aplicaciones de inteligencia arti cial.

NumPy ofrece una variedad de métodos útiles en ndarrays que son esenciales para el procesamiento de datos en el contexto de la inteligencia arti cial. Aquí hay algunos métodos clave:

Funciones de Generación de Números Aleatorios:

reshape(): **Este método permite cambiar la forma (dimensiones) de un ndarray sin cambiar sus datos. Es útil para transformar datos y prepararlos para su entrada en modelos de aprendizaje automático que pueden requerir formatos de entrada especí cos.**

```python
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6])
reshaped_arr = arr.reshape((2, 3))
# Salida: array([[1, 2, 3],
#                [4, 5, 6]])
```

sum(): **Este método calcula la suma de los elementos a lo largo de un eje especí co o de todo el ndarray.**

```python
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])
total_sum = arr.sum()
# Salida: 21
```

Funciones de Generación de Números Aleatorios:

mean(): Calcula la media de los elementos a lo largo de un eje especí co o de todo el ndarray.

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])
average = arr.mean()
# Salida: 3.5
```

min() y max(): Estos métodos devuelven el valor mínimo y máximo del ndarray o a lo largo de un eje especí co.

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])
min_val = arr.min()
max_val = arr.max()
# Salida: min_val = 1, max_val = 6
```

Funciones de Generación de Números Aleatorios:

argmin() y argmax(): **Estos métodos devuelven los índices del valor mínimo y máximo, respectivamente.**

std():**Calcula la desviación estándar de los elementos a lo largo de un eje especí co o de todo el ndarray.**

+ info

```python
import numpy as np

arr = np.array([5, 2, 8, 1, 7])
idx_min = arr.argmin()
idx_max = arr.argmax()
# Salida: idx_min = 3, idx_max = 2
```

```python
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
std_dev = arr.std()
# Salida: 1.414...
```

Estos métodos proporcionan herramientas poderosas para realizar operaciones estadísticas y de procesamiento de datos en conjuntos de datos representados como ndarrays. Al comprender y aplicar estos métodos, puedes realizar análisis e cientes y preparar tus datos para tareas de aprendizaje automático.

Ejercicios Numpy

Creación de Arrays:

a. Crea un array unidimensional de longitud 5 con números enteros aleatorios.
b. Crea un array bidimensional de tamaño 3x3 con valores booleanos True.
c Crea un array tridimensional de tamaño 2x3x4 con valores otantes aleatorios.

Acceso y Modificación de Elementos:

a. Accede al tercer elemento del array creado en el ejercicio 1a.
b. Modifica el segundo elemento de la segunda la del array creado en el ejercicio 1b para que sea False.
c. Calcula la media de los elementos en la tercera dimensión del array creado en el ejercicio 1c.

Ejercicios Numpy

Operaciones Aritméticas:

a. Crea dos arrays unidimensionales de la misma longitud con números enteros aleatorios y realiza una suma elemento por elemento.

b. Crea dos arrays bidimensionales de la misma forma y realiza una multiplicación matriz vectores

c Calcula el producto punto entre dos arrays tridimensionales de la misma forma.

Funciones y Métodos:

a. maximo minimo y Crea un array unidimensional con números enteros aleatorios y encuentra el valor

b. Crea un array bidimensional de tamaño 3x3 con valores otantes y calcula la suma de cada columna.

c. Utiliza la función np.linspace() para crear un array unidimensional con 10 elementos equidistantes entre 0 y 1.

Ejercicios Numpy

Indexación y Slicing:

a. Crea un array unidimensional con números enteros del 0 al 9 y obtén los elementos pares.

b. Crea un array bidimensional de tamaño 4x4 y obtén la submatriz formada por las primeras 2 las y las últimas 2 columnas.

c Crea un array tridimensional de tamaño 3x3x3 y obtén el subarray formado por las primeras dos dimensiones.

¡Estos ejercicios deberían ayudarte a familiarizarte con las funcionalidades básicas de NumPy y a practicar con la manipulación de arrays en Python!