8. Las mejores bibliotecas de Python

En esta sección, vamos a explorar dos de las bibliotecas más importantes de Python: NumPy y Pandas. Estas bibliotecas se utilizan en la ciencia de datos para realizar operaciones matemáticas complejas y para el análisis de datos. NumPy es una biblioteca que proporciona objetos para matrices multidimensionales y operaciones matemáticas de alto nivel. Por otro lado, Pandas es una biblioteca que se utiliza para el análisis y manipulación de datos. Con Pandas, puedes trabajar con datos en una variedad de formatos, incluyendo CSV, Excel y SQL.

A) NumPy

NumPy es una biblioteca de Python que se utiliza para realizar operaciones matemáticas de alta velocidad en matrices multidimensionales. Con NumPy, puedes crear matrices de n dimensiones y realizar operaciones matemáticas en ellas. La biblioteca incluye funciones para la transformación de matrices, operaciones matemáticas y la realización de álgebra lineal. También se puede utilizar para generar números aleatorios y para integrarse con otras bibliotecas de Python.

Algunas de las funciones que se pueden realizar con NumPy son:

i. Matrices

NumPy proporciona varias funciones para crear matrices, como la función numpy.array(). También es posible modificar los elementos de una matriz existente mediante la indexación y la asignación de valores.

```
import numpy as np
```

- Crear una matriz de 2x2

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

```
> [[1 2]
  [3 4]]
```

- Modificar un elemento de la matriz

```
matriz[1][1] = 5
print(matriz)
```

```
> [[1 2]
   [3 5]]
```

- Crear una matriz de ceros de 3x3

```
matriz_ceros = np.zeros((3,3))
print(matriz_ceros)
```

```
> [[0. 0. 0.]
   [0. 0. 0.]
   [0. 0. 0.]]
```

- Crear una matriz de unos de 2x2

```
matriz_unos = np.ones((2,2))
print(matriz_unos)
```

```
> [[1. 1.]
   [1. 1.]]
```

- Crear una matriz de identidad de 3x3

```
matriz_identidad = np.eye(3)
print(matriz_identidad)
```

```
> [[1. 0. 0.]
   [0. 1. 0.]
   [0. 0. 1.]]
```

ii. Métodos integrados

NumPy tiene varios métodos integrados que permiten realizar operaciones matemáticas en matrices. Algunos ejemplos son np.mean(), np.std(), np.max(), np.min().

- Calcular la media de una matriz

```
matriz = np.array([1, 2, 3])
media = np.mean(matriz)
print(media)
```

```
> 2.0
```

- Calcular la desviación estándar de una matriz

```
desviacion = np.std(matriz)
print(desviacion)
```

```
> 0.816496580927726
```

- Calcular la suma de los elementos de una matriz

```
suma = np.sum(matriz)
print(suma)
```

```
> 6
```

iii. Operaciones con matrices

NumPy permite realizar operaciones matemáticas comunes en matrices, como la suma, la resta, la multiplicación, la división y la transposición. También se pueden realizar operaciones más avanzadas, como la multiplicación de matrices y la resolución de sistemas de ecuaciones lineales.

- Sumar dos matrices

```
matriz1 = np.array([[1, 2], [3, 4]])
matriz2 = np.array([[5, 6], [7, 8]])
suma = matriz1 + matriz2
print(suma)
```

```
> [[ 6  8]
   [10 12]]
```

- Multiplicar dos matrices

```
matriz1 = np.array([[1, 2], [3, 4]])
matriz2 = np.array([[5, 6], [7, 8]])
multiplicacion = np.dot(matriz1, matriz2)
print(multiplicacion)
```

```
> [[19 22]
   [43 50]]
```

iv. Operaciones de álgebra lineal

NumPy es muy útil para realizar operaciones de álgebra lineal, como la inversión de matrices, la descomposición de matrices y la resolución de sistemas de ecuaciones lineales. Estas operaciones son fundamentales para muchos problemas de ciencia de datos y aprendizaje automático. Algunos ejemplos son :

- Calcular la inversa de una matriz

```
matriz = np.array([[1, 2], [3, 4]])
inversa = np.linalg.inv(matriz)
print(inversa)
```

```
> [[-2.   1. ]
   [ 1.5 -0.5]]
```

- Calcular el determinante de una matriz

```
determinante = np.linalg.det(matriz)
print(determinante)
```

```
> -2.0000000000000004
```

- Calcular los valores propios y vectores propios de una matriz

```
valores_propios,vectores_propios=np.linalg.eig(matriz)
print(valores_propios)
```

```
> [-0.37228132  5.37228132]
```

```
print(vectores_propios)
```

```
> [[-0.82456484 -0.41597356]
   [ 0.56576746 -0.90937671]]
```

B) Pandas

Pandas es una biblioteca de Python que se utiliza para el análisis y manipulación de datos. Con Pandas, puedes trabajar con datos en una variedad de formatos, incluyendo CSV, Excel y SQL. La biblioteca incluye funciones para la limpieza de datos, la exploración de datos y la manipulación de datos. También se puede utilizar para la unión de datos, la concatenación y la agrupación.

i. DataFrames

Un DataFrame es una estructura de datos de dos dimensiones similar a una tabla de base de datos o una hoja de cálculo de Excel. Los DataFrames se pueden crear a partir de archivos CSV, Excel o bases de datos, y también se pueden crear manualmente desde cero. Los DataFrames se pueden indexar, filtrar y manipular de diferentes maneras.

```
import pandas as pd
```

Creación de un DataFrame con tres columnas: "Nombre", "Edad" y "Ciudad"

```
datos = {'Nombre': ['Juan', 'Ana', 'Pedro', 'Luis'],
        'Edad': [25, 30, 20, 28],
        'Ciudad': ['Madrid', 'Barcelona', 'Valencia', 'Sevilla']}
df = pd.DataFrame(datos)
print(df)
```

```
>   Nombre  Edad     Ciudad
  0   Juan    25     Madrid
  1    Ana    30  Barcelona
  2  Pedro    20   Valencia
  3   Luis    28    Sevilla
```

ii. Series

Las Series son estructuras de datos de una dimensión que se utilizan para almacenar datos homogéneos, como una columna de un DataFrame. Las Series se pueden crear a partir de listas, diccionarios o arreglos de NumPy, y también se pueden crear manualmente desde cero. Las Series se pueden indexar, filtrar y manipular de diferentes maneras.

- Creación de un DataFrame con tres columnas: "Nombre", "Edad" y "Ciudad"

```
datos = {'Nombre': ['Juan', 'Ana', 'Pedro', 'Luis'],
        'Edad': [25, 30, 20, 28],
        'Ciudad':['Madrid', 'Barcelona', 'Valencia', 'Sevilla']}
df = pd.DataFrame(datos)
```

- Indexación por nombre de columna

```
print(df['Nombre'])
```

```
> 0     Juan
  1      Ana
  2    Pedro
  3     Luis
  Name: Nombre, dtype: object
```

- Indexación por posición de columna

```
print(df.iloc[:, 1])
```

```
> 0    25
  1    30
  2    20
  3    28
  Name: Edad, dtype: int64
```

- Indexación por posición de fila y columna

```
print(df.iloc[1, 2])
```

```
> Barcelona
```

iii. Funciones y métodos

Pandas proporciona herramientas útiles para limpiar y preparar datos antes del análisis. Es común que los datos contengan valores duplicados, valores incorrectos o faltantes. Pandas permite identificar y eliminar valores duplicados, valores nulos, y también permite reemplazar los valores faltantes con valores medios o medianos. Además, se pueden eliminar columnas que no son necesarias para el análisis.

- **read_csv() :** lee un archivo CSV y lo convierte en un objeto DataFrame.
- **head(n) :** devuelve las primeras n filas del DataFrame.
- **tail(n) :** devuelve las últimas n filas del DataFrame.
- **info() :** proporciona información sobre el DataFrame, incluyendo el número de filas y columnas, los tipos de datos de las columnas y si hay valores nulos.
- **describe() :** proporciona estadísticas descriptivas para cada columna del DataFrame.
- **groupby() :** agrupa el DataFrame por una o más columnas y permite realizar operaciones en el grupo.
- **drop() :** elimina filas o columnas del DataFrame.
- **fillna() :** rellena los valores nulos en el DataFrame con un valor especificado.
- **sort_values() :** ordena el DataFrame por una o más columnas.
- **apply() :** aplica una función a una columna o fila del DataFrame.

Estas son solo algunas de las funciones y métodos disponibles en Pandas. El conjunto completo de funciones y métodos es extenso y puede ser consultado en la documentación oficial de Pandas.

NumPy y Pandas son dos bibliotecas muy importantes para la ciencia de datos en Python. NumPy se utiliza para realizar operaciones matemáticas en matrices multidimensionales, mientras que Pandas se utiliza para el análisis y manipulación de datos. Con estas bibliotecas, puedes trabajar con datos en una variedad de formatos, realizar operaciones matemáticas complejas y manipular datos de manera eficiente