
Universidad
Tecmilenio®

Universidad Tecmilenio

Fundamentos y aplicaciones de la inteligencia artificial

Librerías para el manejo de datos en Python

Librerías para la visualización de datos en Python

Introducción

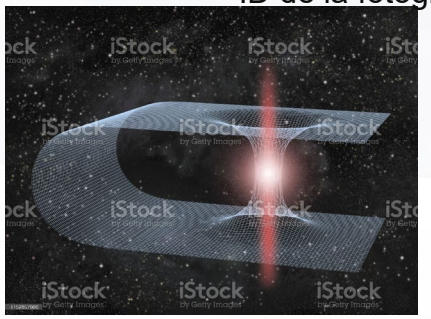
El 10 de abril de 2019, toda la comunidad científica festejó uno de los logros más grandes de la historia de la humanidad, por primera vez se logró fotografiar un agujero negro. Para obtener la imagen se consiguió combinar el poder de varios telescopios ubicados en diferentes partes del planeta a través de un equipo de ingenieros de software, quienes realizaron el procesamiento de las enormes cantidades de datos requeridas para generar la imagen final, utilizando Python y la librería estrella para manejar los datos: NumPy (Godfroid, 2020).

De acuerdo con Matthes (2019), la visualización de datos es muy importante porque permite su análisis y de ese modo explorar patrones y significado en el conjunto de datos, el cual puede estar formado por una pequeña lista de números que cabe en una línea de código o por muchos gigabytes de información.

Una ventaja de utilizar Python es que conjuntos de datos formados por millones de puntos de datos individuales se pueden explorar rápidamente en tan solo una computadora portátil sin necesidad de un equipo sofisticado.

En el desarrollo de este tema aprenderás sobre las características de las principales herramientas que tiene Python para operar con datos, ya que actualmente se aplica para llevar a cabo trabajos intensivos en datos, por ejemplo, de genética, investigación climática, análisis político y económico, entre otros.

ID de la fotografía:1152857986



Explicación

Uso de librerías para arreglos y matrices en NumPy

NumPy sirve de interfaz entre antiguos programas escritos en C, C++ o Fortran y las nuevas formas de programación orientadas a arreglos. Esta facilidad incrementa la vida útil de muchas aplicaciones científicas que de otra forma ya estarían descontinuadas.

Una de las razones por las que NumPy es tan importante para la computación numérica en Python es que está diseñado especialmente para trabajar con grandes arreglos de datos.. Almacena los datos de forma interna en bloques contiguos de memoria. Esto es posible por la forma en que opera el algoritmo de bajo nivel escrito en C y, por supuesto, agregando la ventaja de que se consume una menor cantidad de recursos de memoria.

Todas las funcionalidades de esta librería se encuentran dentro del paquete NumPy. La forma recomendada de incorporarlo dentro de nuestros proyectos se muestra a continuación:

```
In [1]:  import numpy as np
```

ID de la fotografía:1326558181

Explicación

Arreglos en NumPy

Una de las características principales de NumPy es su arreglo n-dimensional, más conocido como “ndarray”, el cual es un contenedor rápido y flexible para cualquier conjunto de datos. Este nos permite realizar operaciones matemáticas en grandes bloques de datos con la misma sencillez que se efectuaría con cada elemento de forma independiente. Todos sus elementos deben ser del mismo tipo.

Un “ndarray” se crea de la siguiente forma:

```
In [2]: datos = [6, 4, 3.5, 8, 6, 9]
        mi_primer_ndarray = np.array(datos)

        array([6. , 4. , 3.5, 8. , 6. , 9. ])
```

mi_primer_ndarray.shape → muestra la forma del arreglo (6,)

mi_primer_ndarray.dtype → muestra el tipo de datos dtype('float64')

De la información anterior podemos concluir que nuestro arreglo es unidimensional, formado por una fila con seis elementos, los cuales son del tipo de punto flotante de doble precisión (64 bits).

Explicación

En la siguiente tabla se muestran algunas funciones que incluye la librería y que nos permiten operar con los arreglos de diferentes formas:

Función	Descripción
Array	Convierte los datos de entrada (lista, tupla, arreglo o cualquier otra estructura de datos) en un *ndarray*. El tipo de datos puede ser inferido o indicado específicamente. Esta función realiza una copia de los datos de entrada.
Asarray	Convierte los datos de entrada en *ndarray*, pero no realiza la copia si detecta que la entrada ya es un *ndarray*.
Arange	Devuelve valores espaciados uniformemente dentro de un intervalo dado con el formato de un *ndarray* (Tabla 6).
Ones	Genera un arreglo de unos, con la forma y el tipo de datos indicados (Tabla 7).
Empty	Crea un arreglo con el tamaño y forma indicados, pero solamente reserva su espacio en memoria sin inicializar los valores (Tabla 8).
Full	Crea un arreglo con el tamaño y forma dados, además de inicializar los elementos con el valor indicado (Tabla 9).

Explicación

Manejo de Entradas/Salidas con NumPy

NumPy tiene la capacidad de importar y guardar elementos directamente en archivos externos. Esta utilidad es muy importante cuando realizamos proyectos de ciencia de datos o de inteligencia artificial y necesitamos importar los conjuntos de datos con los que posteriormente vamos a desarrollar nuestros modelos o entrenar nuestros algoritmos.

En el siguiente segmento de código se muestra un ejemplo básico de la operación de salvar en un archivo a través de la función *save()*. Es importante destacar que, en caso de no especificarse, NumPy le asignara la extensión “npy” al fichero exportado.

```
In [11]: arr5 = np.arange(20)
         arr5

         array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10,
         11, 12, 13, 14, 15, 16, 17, 18, 19])

In [12]: np.save("valores_0_al_20", arr5)
```

ID de la fotografía:1300745916

Savetxt() los exporta como archivos de texto.

Uso de librerías para la manipulación de datos y análisis en Pandas

Otra de las librerías indispensables cuando desarrollamos proyectos de ciencia de datos e inteligencia artificial es **Pandas**. La mayor diferencia es que Pandas se creó para trabajar con datos heterogéneos y tabulares, mientras que NumPy opera de forma preferencial con arreglos de datos homogéneos (McKinney, 2017).

La forma que se recomienda para incorporar el paquete Pandas dentro de nuestros proyectos se muestra a continuación:

```
In [1]:  import pandas as pd
```

O importar solamente algunas funciones de la librería.

```
In [1]:  from pandas import Series, Dataframe
```

ID de la fotografía:913582626

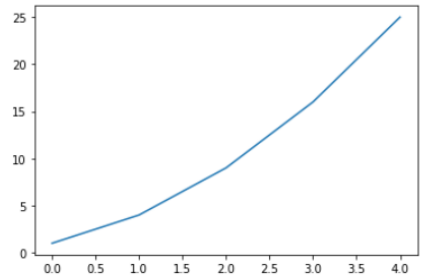
Explicación

Librerías para graficado en Matplotlib

Con la librería Matplotlib se pueden generar gráficos como líneas y diagramas de dispersión de una forma muy simple. Como primer ejemplo, trazaremos un gráfico de una línea simple empleando Matplotlib

1. Importar el módulo *pyplot* usando el alias *plt* para no repetir toda la palabra (esa suele ser una práctica muy común al utilizarlo). El módulo contiene una serie de funciones que generan gráficos y diagramas.
2. Después se crea una lista llamada *cuadrados* para contener los datos que se trazarán.
3. En la línea **3** se llama a la función *subplots()* de Matplotlib. Esta función puede generar uno o más gráficos en la misma figura.
4. La variable *fig* representa la figura completa o la colección de gráficos que se generen.
5. La variable *ax*, por su parte, representa un solo gráfico en la figura y se usará la mayor parte del tiempo.
6. Como últimos pasos para este ejemplo se usa el método *plot()*, cuya función es graficar los datos proporcionados, mientras que la función plt.show () abre el visor de Matplotlib y muestra el gráfico, como pudo apreciarse en el resultado de la tabla 1. Sin embargo, al analizar el gráfico es que el valor de 4 en el eje horizontal no coincide con el valor de 25 en el eje vertical.

```
In [1]: import matplotlib.pyplot as plt
        cuadrados = [1, 4, 9, 16, 25]
        fig, ax = plt.subplots()
        ax.plot(cuadrados)
        plt.show()
```

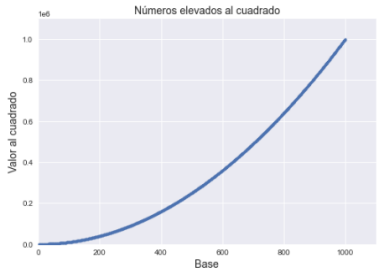


ID de la fotografía:1361507082

Explicación

Sin embargo, al analizar el gráfico es que el valor de 4 en el eje horizontal no coincide con el valor de 25 en el eje vertical, por lo que, en el siguiente ejemplo, se muestra cómo graficar de manera más eficaz cuando se tienen muchos puntos y con ello se logra un gráfico de mejor calidad, lo cual permite optimizar también la interpretación:

```
In [3]:  import matplotlib.pyplot as plt                                    ...
         x_valores = range(1, 1001)                                         2
         y_valores = [x**2 for x in x_valores]                              ...
         plt.style.use('seaborn')                                           ...
         fig, ax = plt.subplots()                                           ...
         ax.scatter(x_valores, y_valores, s=10)                             6
         #Establecer el titulo del gráfico y los ejes de la etiqueta.      ...
         ax.set_title("Números elevados al cuadrado", fontsize=14)          ...
         ax.set_xlabel("Base", fontsize=14)                                 ...
         ax.set_ylabel("Valor al cuadrado", fontsize=14)                    ...
         #Establezca el rango para cada eje.                                ...
         ax.axis([0, 1100, 0, 1100000])                                     12
         plt.show()
```



En la línea **2** se establece un rango de valores de *x* que contienen los números del 1 al 1,000. Después se declara una lista que genera los valores de *y* recorriendo los valores de *x* (para *x* en *valores_x*), elevando al cuadrado cada número (*x**2*) y que almacenará los resultados en *y_values*. Lo siguiente es pasar las listas de entrada y salida a la función *scatter()* en la línea **6** del código. Debido a que esto representa un conjunto de datos grande, se utiliza un tamaño de punto más pequeño.

En la línea **12** se usa el método *axis()* para especificar el rango de cada eje. Este método requiere cuatro valores: los valores mínimo y máximo para el eje *x* y *y*. Es así como se genera el resultado mostrado en la tabla 3, donde el eje *x* va de 0 a 1,100 y el eje *y* de 0 a 1,100,000.

Explicación

Si se desea guardar el gráfico en un archivo, se puede hacer por medio de la función *plt.savefig()*:
El parámetro *linewidth* controla el grosor de la línea.

Librerías estadísticas en Seaborn y Plotly

Seaborn es una biblioteca para hacer gráficos estadísticos en Python basada en Matplotlib y se integra muy de cerca con las estructuras de datos de Pandas. Su API declarativa orientada a conjuntos de datos le permite centrarse en lo que significan los diferentes elementos de los gráficos en vez de los detalles sobre cómo dibujarlos.

Por otra parte, Matthes (2019) explica que el paquete de **Plotly** es una herramienta útil para producir visualizaciones interactivas, en especial cuando se crean visualizaciones que deben mostrarse en un navegador y es necesario escalarlas de manera automática para que se adapten a la pantalla del espectador.

Las visualizaciones que genera Plotly también son interactivas cuando el usuario pasa el cursor sobre ciertos elementos en la pantalla, la información sobre ese elemento se resalta.

Para hacer uso de Plotly en Python es necesario instalarlo.

ID de la fotografía:1329183101

Explicación

El siguiente ejemplo te permitirá explorar algunas características y funciones de Seaborn:

```
In [1]:
#Se importa seaborn
import seaborn as sns
#Se aplica el tema predeterminado
sns.set_theme()
#Se carga una base de datos de ejemplo
propinas = sns.load_dataset("tips")
#Se realiza la visualización de los datos
sns.relplot(

data=propinas,
x="total_bill", y="tip", col="time",
hue="smoker", style="smoker", size="size",

)
```

En la línea **2** se importa la librería Seaborn como la abreviatura *sns*.

Al declarar *sns.set_theme()* en la línea **4** se emplea el sistema de *matplotlib rcParam*, entonces, el aspecto de cada gráfico es modificado.

La función *load_dataset()* usada en la línea **6** permite el acceso rápido a un conjunto de datos de ejemplo, hecho en una estructura de Pandas de código abierto, también construida en Python.

El código a partir de la línea **7** lleva a cabo la gráfica, la cual es una relación entre cinco variables en el conjunto de datos tips (propinas) haciendo una sola llamada a la función *seaborn relplot()*.

Para esto, se proporcionan únicamente los nombres de las variables y se especifica el rol de cada una de ellas.

A diferencia de Matplotlib, no es necesario especificar atributos de los elementos en términos de valores de color o códigos de marcador. Detrás del código del programa, Seaborn traduce los argumentos que Matplotlib entiende.

Este enfoque permite concentrarse en el análisis de los datos, en lugar de los detalles de cómo controlar Matplotlib.

Actividad

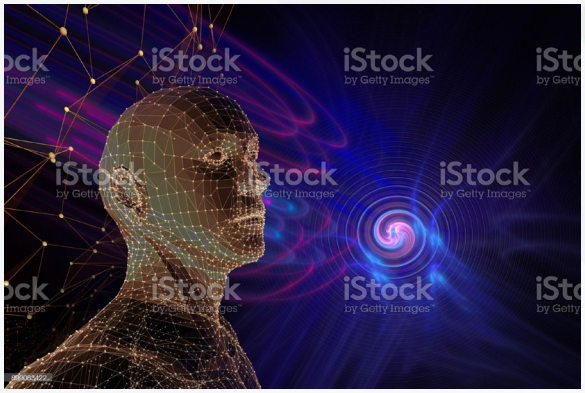
Universidad Tecmilenio

Objetivo: programar ejercicios que implementen manejo de archivos, objetos y clases.

Según lo aprendido en el presente tema:

A. Analiza y explica qué hace cada línea/segmento de código a continuación:

```
1. import numpy as np
2. import pandas as pd
3. In [3]:
lista = [1, 2, 3, 4, 5]
In [4]:
type(lista)
4. In [8]:
a.dtype
Out[8]:
dtype('int64')
```



ID de la fotografía:899083422

B. Explica en qué aspecto de la vida real utilizarás las librerías que acabas de conocer.

¡Éxito!

Cierre

En este tema aprendiste a generar conjuntos de datos y crear las visualizaciones de estos.

Se crearon gráficos simples con NumPy, Panda y Matplotlib, y se especificaron los diferentes enfoques y usos de cada librería para considerar integrarlas en tus proyectos futuros al desarrollar aplicaciones que deben ofrecer una interfaz clara para los usuarios.

Finalmente, desarrollar la habilidad de generar tus conjuntos de datos propios programando en Python permite modelar y explorar una amplia variedad de situaciones del mundo real de forma poderosa. A medida que continúes aplicando cada una de estas herramientas para la visualización de datos en tus proyectos, estos serán comprensibles para cualquier tipo de público y eso se traducirá en un mayor alcance.

ID de la fotografía:1282184282

Bibliografía

Godfroid, M. (2020). *Why NumPy Is So Fundamental*. Recuperado de https://towardsdatascience.com/why-numpy-is-so-fundamental-78ae2807300

Matthes, E. (2019). *Python Crash Course: A Hands-On, Project-Based Introduction to Programming* (2ª ed.). Estados Unidos: No Starch Press.

McKinney, W. (2017). *Python for Data Analysis* (2ª ed.). Estados Unidos: O'Reilly Media.