BI4SME

boosting business intelligence skills for SME growth

Co-funded by the European Union

Este documento se crea en el marco del proyecto BI4SME "Boosting Business Intelligence Skills for SME Growth" (ACUERDO DE SUBVENCIÓN 2021-1-ES01-KA220-VET-000033132). Este proyecto ha sido financiado con el apoyo de la Comisión Europea. Esta publicación refleja únicamente las opiniones del autor, y la Comisión no se hace responsable del uso que pueda hacerse de la información contenida en él.

Co-funded by the European Union

Index

Co-funded by the European Union

UNIDAD 5: Python

5.1. Conceptos generales e instalación del entorno

Python es uno de los lenguajes de programación más utilizados hoy en día y puede ser empleado para una amplia variedad de propósitos. Por ejemplo, puede ser adoptado para desarrollo web y de software, análisis de datos, análisis estadístico, aprendizaje automático, automatización de tareas, computación científica, entre otros. En el contexto de este curso, utilizaremos **Python para Inteligencia de Negocios (BI).**

Existen varias características que hacen de Python un buen punto de entrada a la programación para principiantes y una herramienta útil en el ámbito empresarial. Algunas de estas características incluyen la facilidad para aprender su sintaxis, su facilidad de uso, la intuición al leer el código, una gran cantidad de bibliotecas completas, herramientas y documentación, y la existencia de una amplia comunidad de desarrolladores, entre otros.

En esta unidad, vamos a utilizar Anaconda para desarrollar nuestro código. Anaconda es una distribución de código abierto de los lenguajes de programación Python y R para ciencia de datos, que tiene como objetivo simplificar la gestión y despliegue de paquetes.

La distribución de **Anaconda** incluye automáticamente más de 250

paquetes instalados. Además, se pueden instalar más de 7500 paquetes adicionales de código abierto desde PyPI, así como a través de conda, el gestor de paquetes y entornos virtuales de Anaconda.

Ahora, vamos a instalar el entorno de desarrollo integrado (IDE) siguiendo estos pasos:

Co-funded by the European Union

1. Primero, abre tu navegador web preferido y accede a https://www.anaconda.com/

   Luego, haz clic en el botón "Download"

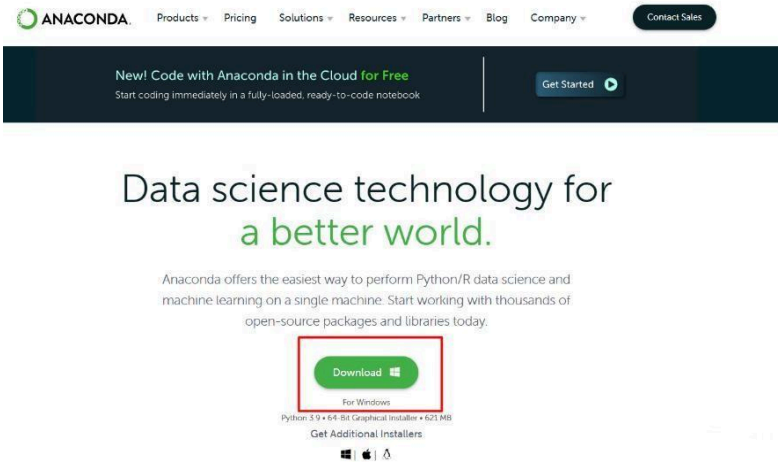


Figura 5.1 – Instrucciones de descarga de Anaconda

2. Después de descargar el programa, ábrelo y aparecerá una ventana de instalación. Sigue exactamente los mismos pasos que se muestran a continuación:

Figura 5.2 – Paso 1 – Instrucciones de instalación de Anaconda

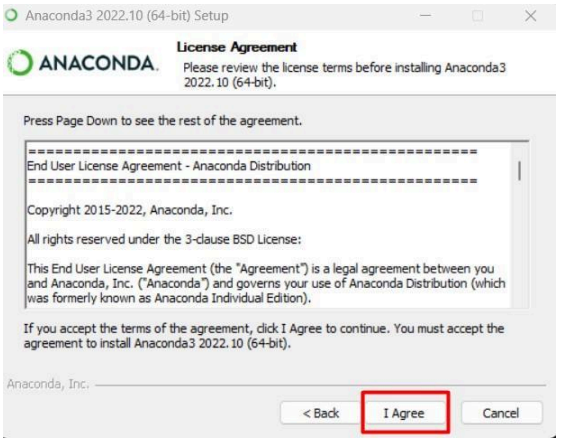


Figura 5.3 – Paso 2 – Instrucciones de instalación de Anaconda

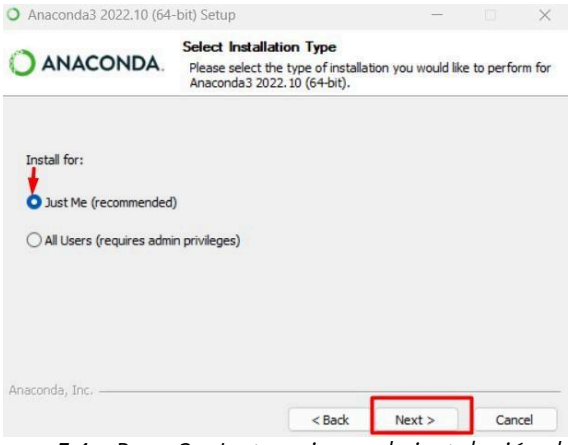


Figura 5.4 – Paso 3 – Instrucciones de instalación de Anaconda

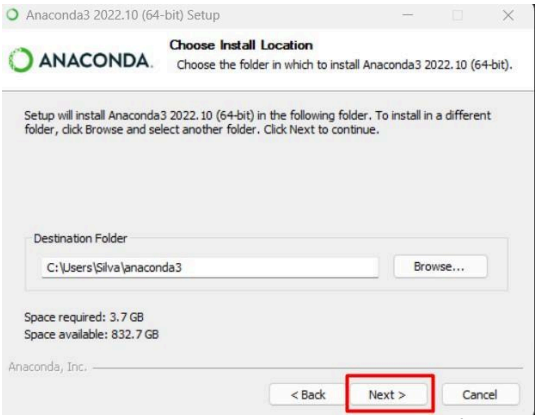


Figura 5.5 – Paso 4 – Instrucciones de instalación de Anaconda

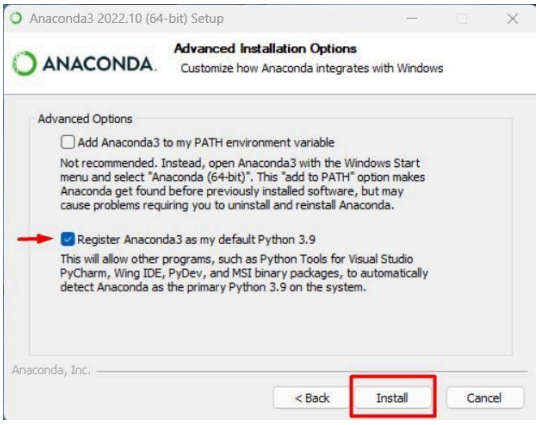


Figura 5.6 – Paso 5 – Instrucciones de instalación de Anaconda

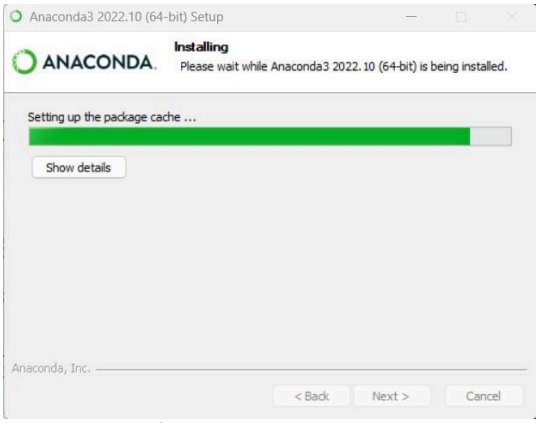


Figura 5.7 – Instalación de Anaconda, barra de progreso

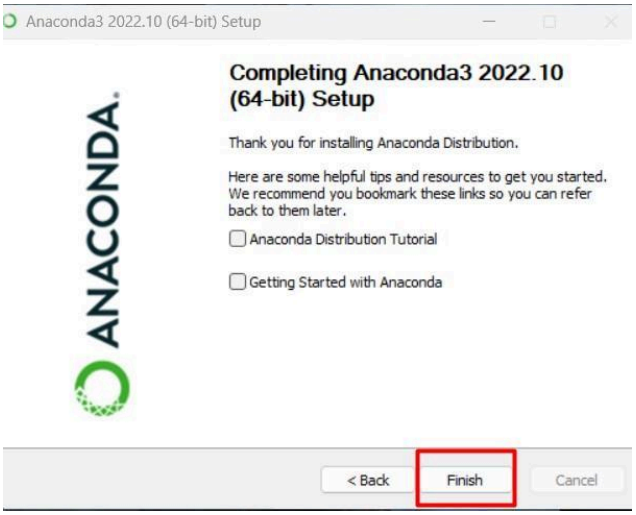


Figura 5.8 – Paso 6 – Conclusión de la instalación de Anaconda

Ahora has instalado correctamente el programa Anaconda.

Ahora, vamos a iniciarlo buscando en la barra de búsqueda de Windows "Anaconda Navigator":

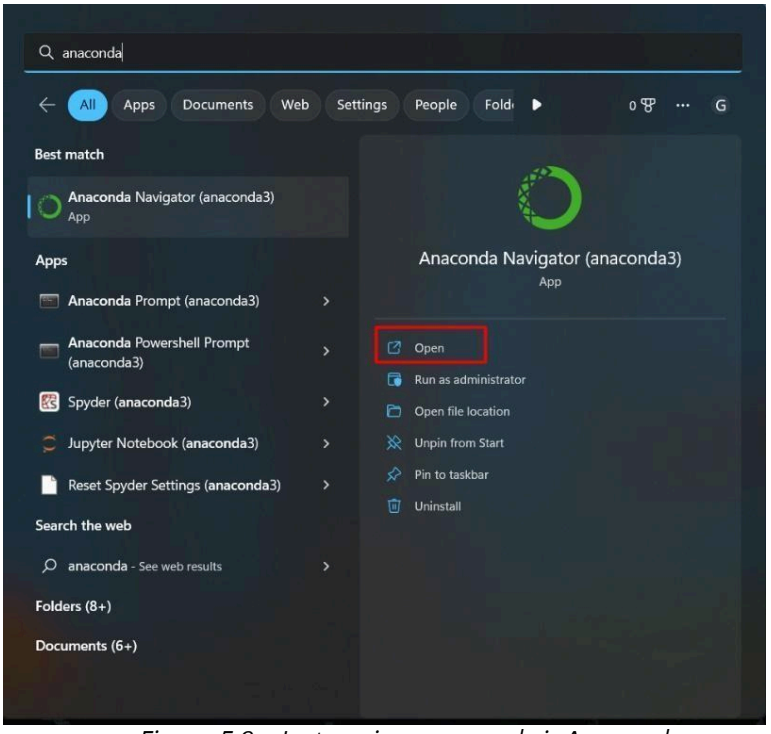


Figura 5.9 – Instrucciones para abrir Anaconda

Después de abrir el programa, aparecerá este panel de control:

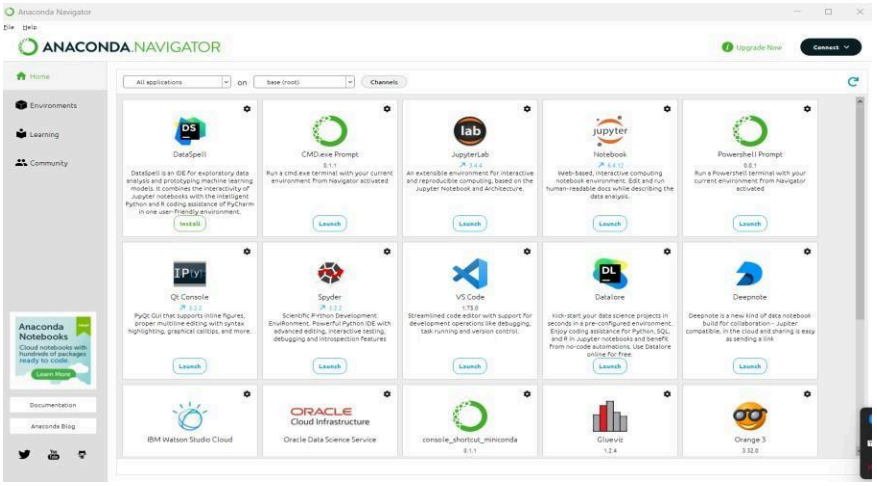


Figura 5.10 – Vista de Anaconda Navigator

Next, press the **Spyder** module:

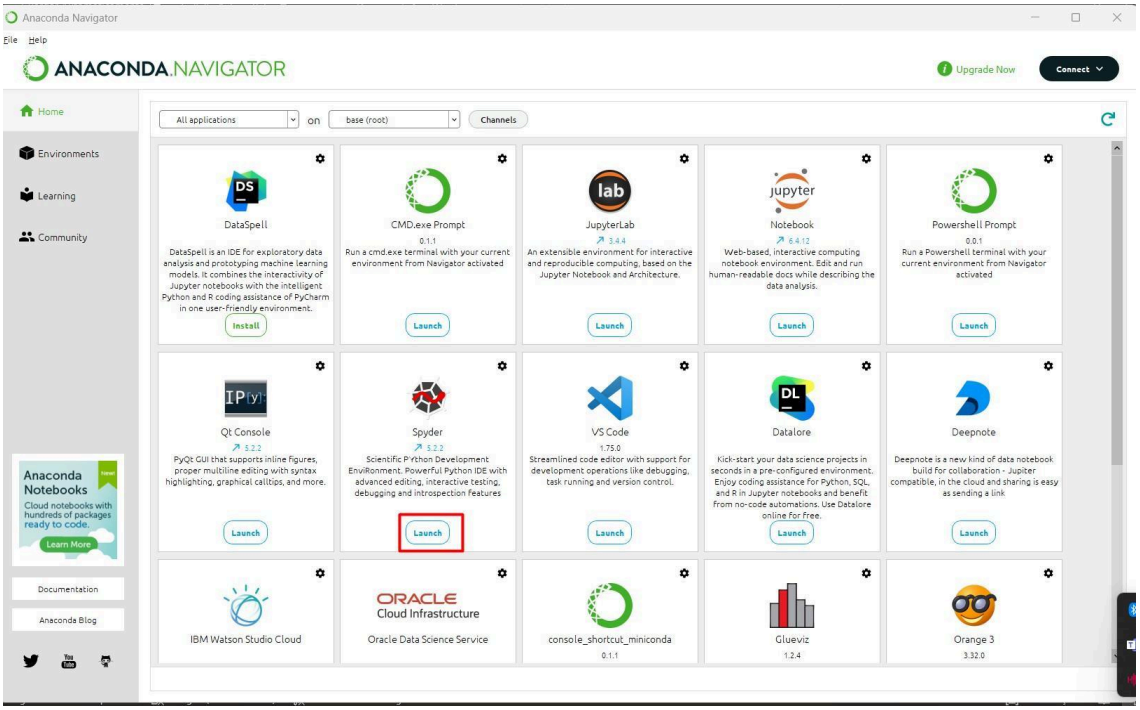


Figura 5.11 – Instrucciones sobre cómo seleccionar el IDE

Cuando hagas clic en *"Launch"*, el IDE **Spyder** se iniciará y podrás usarlo para desarrollar código.

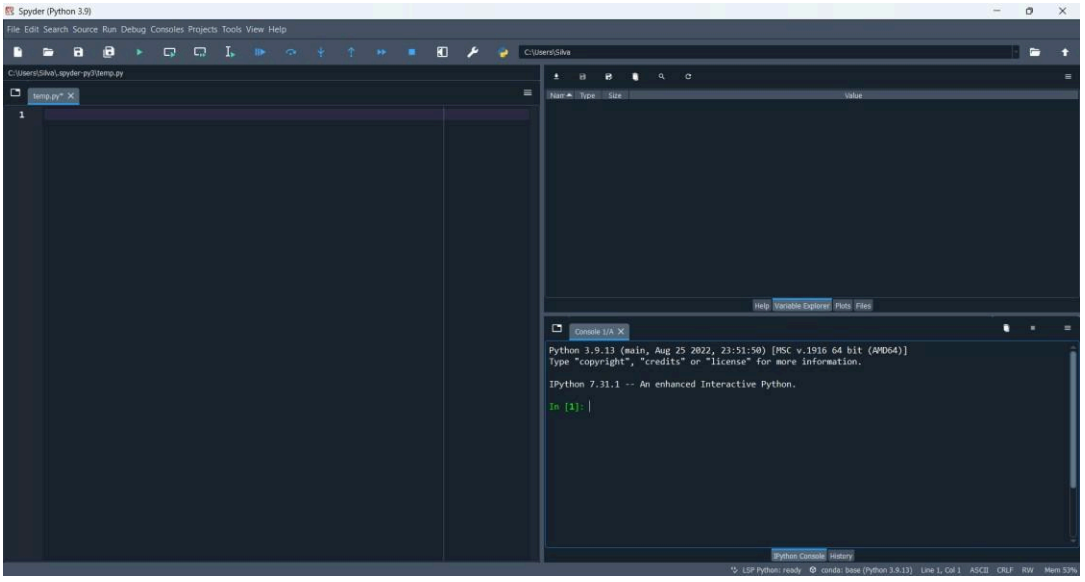


Figura 5.12 – Vista del IDE Spyder

Felicidades! Ahora puedes comenzar a trabajar con **Python!**

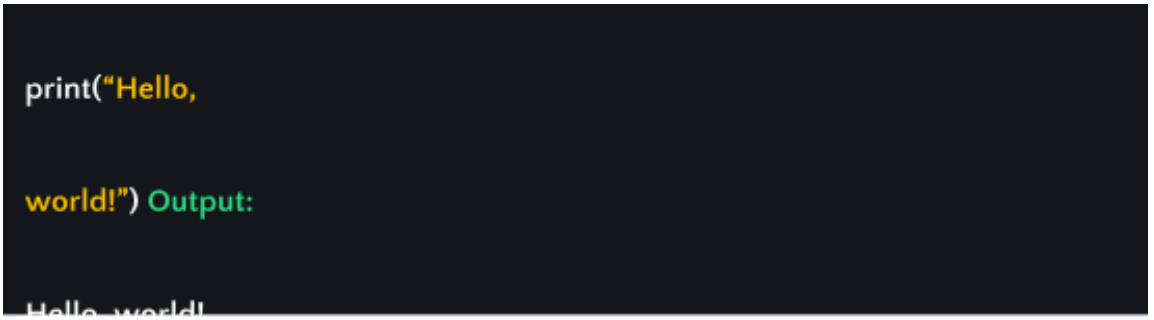
Co-funded by the European Union

5.1.1. Primeros pasos en Python

Ahora que hemos instalado **Python,** ¡vamos a probarlo! Vamos a crear nuestro primer programa en Python:

Primero, crea un nuevo proyecto haciendo clic en **Archivo > Nuevo archivo... > Archivo Python.**

Después de que se haya creado el archivo, ¡vamos a programar! Comienza haciendo el programa más icónico y simple utilizando una línea de código:

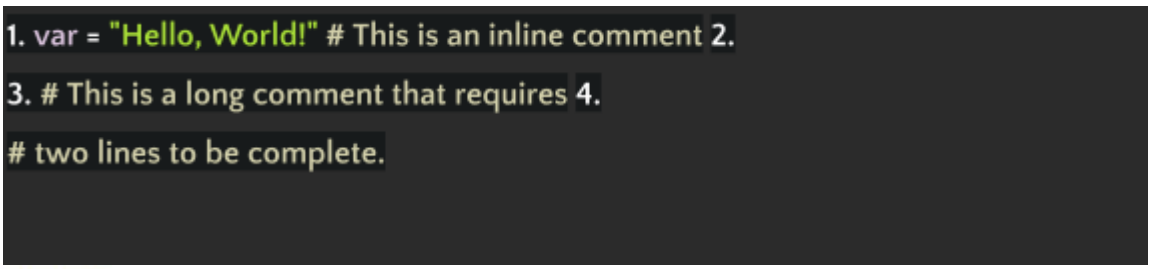


Luego, al ejecutarlo, en la terminal se imprimirá la icónica frase "Hello, world!". ¡Felicidades, acabas de crear tu primer programa!

5.1.1.1. Comentarios en Python

Puedes añadir un comentario en Python utilizando el signo de numeral (#) al inicio de cada línea que desees comentar, o utilizando tres comillas dobles (""") al inicio y al final del bloque de código al que se refiere.

Ejemplo:



Co-funded by the European Union

5.1.1.2. Variables en Python

Ahora vamos a empezar a utilizar algunas variables, ya que son una parte esencial del código. Si deseas guardar o definir un valor, necesitas utilizar una variable:

```
numbers = [1, 2, 3, 4, 5]
print(numbers)

Output: [1, 2, 3, 4, 5]
```

```
variable_name = "variable_value"
print(variable_name)

Output: Variable_value
```

Python ofrece diferentes tipos de variables: enteros (integers), números de punto flotante (floating-point numbers) y números complejos (complex numbers). Los enteros y los números de punto flotante son los tipos numéricos más comúnmente utilizados en la programación diaria, mientras que los números complejos tienen casos de uso específicos en matemáticas y ciencias. Aquí tienes un resumen de sus características:

Number	Description	Examples	Python Data Type

Integer	Whole numbers	1; 2; 42; 476; -9999	int

Floating – Point	Numbers with decimal points	1.0; 2.2; 42.09; -8.2432;	float
complex	Numbers with a real part and an imaginary part	Complex(1, 2) Complex(-1, 7), Complex("1+2j")	complex

Tabla 5.1 - Resumen de los diferentes tipos de números utilizados en la programación en Python

Aquí tienes algunos ejemplos:

```
n = 139.3567
r = int(n)
print(r)
Output: 139
```

```
n = 10
r = float(n)
print(r)
Output: 10.0
```

```
n = complex(1,10)
print(n)
Output: (1+10j)
```

Los operadores representan operaciones como la suma, resta, multiplicación, división, entre otras. Cuando los combinas con números, forman expresiones que Python puede evaluar:

```
# Addition
```

```
n = 5 + 3
print(n)
Output: 8
# Subtraction
n = 5 - 3
print(n)
Output: 2
# Multiplication
n = 5 * 3 print(n)
Output: 15
# Division n
= 5 / 3
print(n)
Output: 1.6666666666666667
# Floor division
n = 5 // 3 print(n)
Output: 1
# Modulus (returns the remainder from division) n = 5
% 3
print(n)
Output: 2
# Power n
= 5 ** 3
print(n)
Output: 125
```


Co-funded by
the European Union

Aquí tienes un ejemplo del mundo real:

```
money = 2000
investment = 1500

profit = money - investment print("You got
a profit of:", profit, "€")

Output: You got a profit of: 500 €
```

Pero este programa no es eficiente en tiempo. Imaginemos que necesitas editar el código cada vez que realizas una venta o un servicio con valores diferentes; esto se puede programar. Así que usemos la función input() que te permite insertar diferentes valores cada vez que lo ejecutes:

```
money = float(input("Please enter the money you got in return for the service:
"))
investment = float(input("Please enter the initial investment: "))

profit = investment - money print("You
got a profit of:", profit, "€") Output:
Please enter the money you got in return of the service: (Enter Value Ex:1034.45)
Please enter the initial investment: (Enter Value Ex: 2549.40)
You got a profit of: 1514.95 €
```

Co-funded by the European Union

Vamos a realizar unos ejercicios para desarrollar nuestra nueva habilidad:

- El lenguaje de programación Python se puede utilizar como una calculadora simple. Verifica el resultado de los siguientes cálculos:

```
2 + 4

40 * 300

1 / 2

1.0 / 2

1.0 // 2

2e30 * 4

20e50 * 20e50

7 % 5

(5 + 2j) + (3 + 4j)

(5 + 2j) * (3 + 4j)

(5 + 2j) / (3 + 4j)
```

- Calcula el número de segundos en un año normal (365 días).
- Supón que tienes una habitación rectangular de medidas 8 x 6 metros. Suponiendo que deseas cubrir el suelo con baldosas de 2 x 2 metros, calcula la cantidad de unidades que necesitarás para cubrir el suelo.

Co-funded by
the European Union

5.1.1.3. Declaraciones condicionales en Python

La toma de decisiones es el aspecto más importante de casi todos los lenguajes de programación. Como su nombre lo indica, la toma de decisiones nos permite ejecutar un bloque particular de código para una decisión específica. Aquí, las decisiones se toman en función de la validez de condiciones particulares. La verificación de condiciones es la base de la toma de decisiones.

En Python, la toma de decisiones se realiza mediante las siguientes declaraciones:

Declaración	Descripción
IF	La palabra clave "IF" es la manera en Python de decir "si una condición especificada es verdadera".
ELIF	La palabra clave "ELIF" es la manera en Python de decir "si las condiciones anteriores no fueron verdaderas, entonces prueba esta condición".
ELSE	La palabra clave "ELSE" captura cualquier cosa que no sea capturada por las condiciones anteriores.

Tabla 5.2 - Resumen de las declaraciones en Python

Python also supports the usual logical conditions from Mathematics:

Condición lógica	Descripción

a == b	a Equals b
a != b	a Not Equal b
a < b	a Less than b
a <= b	a Less than or equal to b
a > b	a Greater than b
a >= b	a Greater than or equal to b

Tabla 5.3 – Lista de condiciones lógicas matemáticas compatibles con Python

Aquí tienes algunos ejemplos prácticos:

```
a = 33
b = 200
if b > a:
  print("b is greater than a"):
Output: b is greater than a
```

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
Output: a and b are equal
```

```
a = 200
b = 33
```

```
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
Output: a is greater than b
```

En este ejemplo, 'a' es mayor que 'b', por lo que la primera condición no es verdadera. La condición elif tampoco es verdadera, por lo que pasamos a la condición else y mostramos en pantalla que "a es mayor que b".

También puedes tener un else sin el elif:

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
Output: b is not greater than a
```

Aquí tienes un ejemplo del mundo real:

```
# Variables

service = float(input("How much you earned from the service: ")) investment =
float(input("insert here initial investment (Materials used): "))
```

```
serviceHours = float(input("How many hours did take to provide the service: "))
employees = int(input("how many employees do you have: ")) moneyEmployees =
float(input("how much you pay an hour: "))

# Calculate the employees earned from this service

employeesEarned = moneyEmployees * serviceHours

# calculate the profit

profit = service - (investment + (employeesEarned * employees))

if profit == 0:
  print("you didn't make any money from this service") elif
profit > 0:
  print("you made a profit of:", profit, "€") else:
  print("you lost:", profit, "€" )
```

Output:

How much you earned from the service: Insert Here value (Example: 1500) insert here initial investment (Materials used): Insert Here value (Example: 300)
How many hours did take to provide the service: Insert Here value (Example: 2)
how many employees do you have: Insert Here value (Example: 5)

Co-funded by the European Union

how much you pay an hour: Insert Here value (Example: 5)

you made a profit of: Value calculated (Example: 1150.0 €)

Este programa está diseñado para calcular cuánto dinero has ganado por un servicio, teniendo en cuenta los gastos calculados.

Vamos a hacer algunos ejercicios para desarrollar nuestra nueva habilidad:

- Escribe un programa para verificar si una persona es elegible para votar o no. (Aceptar entrada del usuario)
- Escribe un programa para verificar si un número es divisible por 7 o no. (Aceptar entrada del usuario)
- Escribe un programa para aceptar un número del 1 al 7 y mostrar el nombre del día de la semana correspondiente. (Ejemplo: 1 = Domingo) (Aceptar entrada del usuario)
- Escribe un programa para aceptar un número del 1 al 12 y mostrar el nombre del mes y los días en ese mes. (Ejemplo: 1 = Enero) (Aceptar entrada del usuario)

5.1.1.4. Básicos de los bucles en Python

El lenguaje de programación **Python** proporciona diferentes tipos de bucles para manejar requerimientos de iteración. Proporciona tres formas de ejecutar los bucles. Aunque todas las formas proporcionan funcionalidades básicas similares, difieren en su sintaxis y en el momento de comprobación de condiciones.

Un **bucle while** se utiliza para ejecutar un bloque de declaraciones repetidamente hasta que se cumpla una condición dada. Cuando la condición se vuelve falsa, se ejecuta la línea inmediatamente después del bucle en el programa.

Ejemplo:

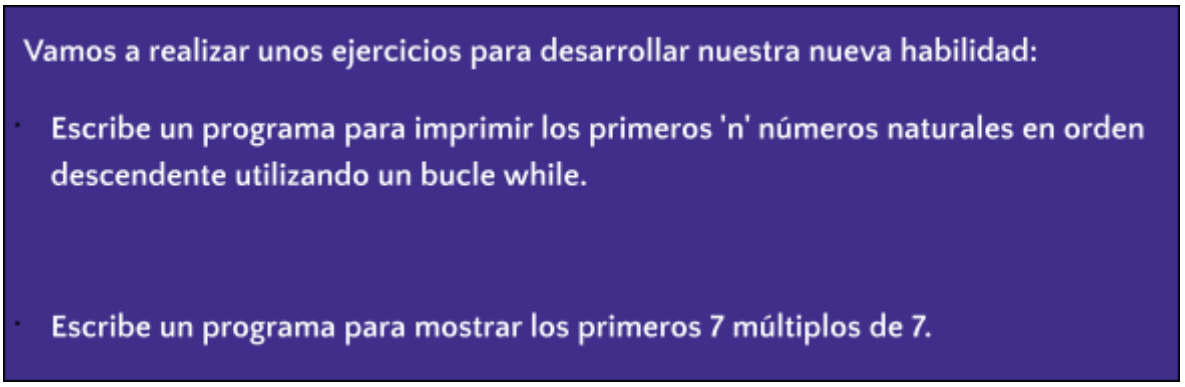
```
# while loop
count = 0
while (count < 3):
    count = count + 1
    print("BI4SME")

Output: BI4SME
        BI4SME
        BI4SME
```

Cuando la condición se vuelve falsa, se ejecuta la declaración inmediatamente después del bucle.

La cláusula *else* solo se ejecuta cuando la condición del bucle *while* se vuelve falsa. Si sales del bucle con break, o si se genera una excepción, la cláusula else no se ejecutará.

Ejemplo:

```
count = 0
while (count < 3):
    count = count + 1
    print("BI4SME")
else:
    print("In Else
Block") Output:
BI4SME
SME
```



5.1.2. Análisis de datos en programación

La forma más simple de describir los datos es como "información traducida a forma binaria digital" (AMC College, 2022, p. 1). Hoy en día, hay más datos que nunca. Según un estudio realizado por Statista, el volumen global de datos creados, capturados, copiados y consumidos alcanzará los 181 zettabytes para 2025 (Statista, 2022). Todos estos datos necesitarán ser ordenados, limpiados, analizados y visualizados.

Afortunadamente, existe una amplia variedad de herramientas y software disponibles para el modelado y la visualización de datos, entre las cuales se encuentran algunos lenguajes de programación. Antes de abordar cómo se puede utilizar la programación para el análisis de datos, veamos en qué consiste este proceso.

El análisis de datos es el proceso de obtener información útil a partir de datos para tomar decisiones empresariales más informadas. El proceso de

análisis de datos se desarrolla a través de 5 etapas diferentes:

1. **Identificar** la pregunta que necesitas responder y los datos que deseas analizar;
2. **Recopilar** los datos;
3. **Limpiar** los datos para prepararlos para el análisis, lo que implica eliminar duplicados, errores y datos anómalos para mantener la calidad de los datos
4. **Analizar** los datos utilizando diferentes técnicas y herramientas para encontrar tendencias, correlaciones entre diferentes categorías y datos, outliers y variaciones. Durante esta etapa, puedes minar datos para encontrar patrones o utilizar visualización de datos para transformar los datos en un formato gráfico que facilite la interpretación de los datos;
5. **Interpretar** los resultados del análisis.

Mira el siguiente video[1] sobre **Data Analysis**:

Aquí hay algunas preguntas para consolidar el tema anterior:

1. Cuál es la forma más simple de describir los datos?

2. Según Statista, ¿cuál será el volumen global estimado de datos para 2025?

[1] https://www.youtube.com/watch?v=OQUfnEJvMXk

- Cuáles son las diferentes etapas involucradas en el proceso de análisis de datos?
- Por qué es importante limpiar los datos antes de analizarlos?
- Cómo puede la visualización de datos ayudar a interpretar los resultados del análisis de datos?

5.1.2.1.Tipos de Análisis de Datos

Data analytics can be very important for businesses, as data can be used to answer questions and aid decision-making processes, and there are different types of data analytics:

El **análisis de datos** puede ser muy importante para los negocios, ya que los datos pueden usarse para responder preguntas y ayudar en los procesos de toma de decisiones. Existen diferentes tipos de **análisis de datos:**

Tipos de Análisis de Datos	Descripción
Análisis descriptivo	Este tipo de análisis puede usarse para describir y resumir datos cuantitativos mediante estadísticas; responde a la pregunta: "Qué sucedió?"

Análisis diagnóstico	Este tipo de análisis responde a la pregunta: "Por qué sucedió?". Justifica los resultados mostrados por el tipo de análisis anterior.
Análisis predictivo	Interpreta datos para hacer proyecciones sobre el futuro y responder a la pregunta: "Qué podría suceder?"
Análisis prescriptivo	Finalmente, este tipo reúne las ideas obtenidas de los 3 tipos mencionados anteriormente y utiliza estos datos para crear recomendaciones; responde a la pregunta: "¿Qué deberíamos hacer al respecto?" y es el tipo que se aplica directamente en la toma de decisiones basada en datos.

Tabla 5.4 – Lista y descripción de los tipos de análisis de datos:

- Cuál es un ejemplo de análisis descriptivo en los negocios?
- Cómo se puede utilizar un análisis diagnóstico en Marketing?

Co-funded by the European Union

5.1.2.1. Métodos de Análisis de Datos

Para lograr los objetivos de los tipos mencionados anteriormente, hay diferentes métodos que puedes usar:

Métodos de Análisis de Datos	Descripción
Análisis factorial	Este método consiste en condensar varias variables en solo unas pocas para facilitar el análisis de datos.
Análisis de regresión	Este método utiliza procesos estadísticos para examinar la relación entre dos o más variables

Co-funded by the European Union

Data mining	Este método consiste en buscar tendencias, patrones y correlaciones en grandes conjuntos de datos.
Análisis de clúster	Este método organiza los datos en grupos y/o clústeres que comparten características comunes.
Análisis de texto	Finalmente, este método extrae información legible por máquina a partir de texto no estructurado (por ejemplo, PDFs, correos electrónicos, documentos de procesamiento de texto).

Tabla 5.5 – Lista y descripción de los Métodos de Análisis de Datos

Fuente: *What is Data Analysis? And How Can You Start Learning It Today?*, by Learn to Code with Me

Como se mencionó anteriormente, algunos lenguajes de programación se utilizan comúnmente para el análisis de datos, uno de los más utilizados es Python. A continuación, hablaremos sobre Big Data y aprenderemos por qué se usa Python y qué papel juega en el análisis de datos.

5.1.3. Big Data en Python

Big Data es la tecnología que te permite analizar grandes volúmenes de datos que no podrían ser procesados con otras herramientas, como bases de datos relacionales o aplicaciones estadísticas tradicionales.

Comienza con datos sin procesar que no han sido agregados y a menudo son demasiado grandes para caber en la memoria de una sola computadora e implica el uso de análisis, aprendizaje automático, minería y estadísticas, para aumentar y proporcionar a las empresas una ventaja competitiva (...) las organizaciones y los equipos pueden usar Big Data para realizar muchos procesos en una sola plataforma, almacenar terabytes de datos, preprocesarlos, analizar todos los datos independientemente de su tamaño o tipo, y luego visualizarlos" (AMC College, 2022, p. 2).

Además de ser el lenguaje de programación más popular, Python es relativamente fácil de aprender. Python es un lenguaje de programación dinámico, orientado a objetos, de alto nivel y de propósito general, conocido por su sintaxis intuitiva que imita el lenguaje natural, ya que combina la estructura de datos con una sintaxis fácil de aprender.

- La programación orientada a objetos es un modelo de codificación que organiza datos y funciones en piezas reutilizables de código en clases, objetos, métodos o atributos. Este enfoque es adecuado para programas más grandes y complejos.

- La programación de alto nivel es un tipo de lenguaje que presenta una sintaxis fácil de leer y entender para los humanos, y que se convierte en código máquina para que la computadora lo reconozca y ejecute.

Co-funded by the European Union

5.1.4. Data en Python

Un DataFrame es una estructura de datos de dos dimensiones en la que se pueden almacenar datos de diferentes tipos (como caracteres, enteros, valores de punto flotante, factores y más) en columnas. Es como una hoja de cálculo o una tabla SQL de datos

Python tiene los siguientes tipos de datos en las categorías correspondientes:

Text type:	str
Numeric types:	int, float, complex
Sequence types:	list, tuple, range
Mapping type:	dict
Set types:	set, frozenset
Boolean type:	bool
Binary types:	bytes, bytearray, memoryview
None type:	NoneType

Table 5.6 - Lista de tipos de datos en Python:

Para obtener el tipo de dato de cualquier objeto, puedes usar la función `type():` e.g.,

```
x = 5
print type(x))
```

Como se mencionó anteriormente, **Python** es un lenguaje de programación de alto nivel y de tipo dinámico, lo que significa que el tipo de dato de una variable se determina automáticamente en función del valor que se le asigna. A diferencia de los lenguajes de tipo estático como C o Java, donde los tipos de datos deben ser declarados explícitamente, **Python** permite una mayor flexibilidad y facilidad de uso al trabajar con datos.

Co-funded by
the European Union

Los diferentes tipos de datos en **Python** se han presentado anteriormente clasificados por categoría. A continuación se enumeran los tipos de datos más comunes de **Python:**

1. **Integer:** Los enteros son números enteros sin punto decimal, como 42 o -7. Se almacenan usando el tipo de dato 'int' en Python.

e.g.,

```
# Example of integer data type x
= 42
y = -7
print type(x)) # Output: <class 'int'>
print type(y)) # Output: <class 'int'>
```

2. **Float:** Los números de punto flotante tienen un punto decimal, como 3.14 o -0.7. Se almacenan usando el tipo de dato 'float' en Python

e.g.,

```
# Example of float data type x
= 3.14
y = -0.7
print type(x)) # Output: <class 'float'> print type(y)) #
Output: <class 'float'>
```

3. **String:** Las cadenas (strings) son secuencias de caracteres, como "hola" o "adiós". Se almacenan usando el tipo de dato 'str' en Python y pueden estar encerradas entre comillas simples o dobles.

e.g.,

```
# Example of string data type
x = "hello"
y = 'goodbye'
```

```
print type(x)) # Output: <class 'str'>
print type(y)) # Output: <class 'str'>
```

4. **Boolean:** Los valores booleanos representan verdadero o falso y se almacenan usando el tipo de dato 'bool' en Python.

e.g.,

```
# Example of boolean data type x
= True
y = False
print type(x)) # Output: <class 'bool'> print type(y)) #
Output: <class 'bool'>
```

5. **List:** Las listas son colecciones ordenadas de objetos, como [1, 2, 3] o ["manzana", "banana", "cereza"]. Se almacenan usando el tipo de dato 'list' en Python y son mutables, lo que significa que sus elementos pueden cambiarse.

e.g.,

```
# Example of list data type x
= [1, 2, 3]
y = ["apple", "banana", "cherry"]
print type(x)) # Output: <class 'list'>
print type(y)) # Output: <class 'list'>
```

6. **Tuple:** Las tuplas son similares a las listas, pero son inmutables, lo que significa que sus elementos no pueden cambiarse una vez que se crean. Se almacenan usando el tipo de dato 'tuple' en Python.

Co-funded by
the European Union

```
# Example of tuple data type
x = (1, 2, 3)
y = ("apple", "banana", "cherry")
print type(x)) # Output: <class 'tuple'>
print type(y)) # Output: <class 'tuple'>
```

7. **Set:** Los conjuntos (sets) son colecciones desordenadas de objetos únicos, como {1, 2, 3} o {"manzana", "banana", "cereza"}. Se almacenan usando el tipo de dato 'set' en Python.

e.g.,

```
# Example of set data type x
= {1, 2, 3}
y = {"apple", "banana", "cherry"}
print type(x)) # Output: <class 'set'>
print type(y)) # Output: <class 'set'>
```

8. **Dictionary:**. Los diccionarios son colecciones de pares clave-valor, como {"nombre": "Juan", "edad": 30}. Se almacenan usando el tipo de dato 'dict' en Python y permiten la recuperación eficiente de valores basados en sus claves asociadas.

e.g.,

```
# Example of dictionary data type x
= {'name': 'John', 'age': 32}
y = {'city': 'New York', 'state': 'NY'}
print type(x)) # Output: <class 'dict'>
print type(y)) # Output: <class 'dict'>
```

Ten en cuenta que en los diccionarios, cada clave debe ser única y se puede usar para acceder a su valor asociado. Por ejemplo

```
# Accessing values in a dictionary
x = {'name': 'John', 'age': 32}
print(x['name']) # Output: John
print(x['age']) # Output: 32
```

Es importante entender los diferentes tipos de datos en Python, ya que afectan cómo se almacenan y manipulan los datos en un programa. Se debe usar el tipo de dato adecuado según el propósito previsto de los datos, ya que algunos tipos de datos son más adecuados para ciertas tareas que otros.

- Escribe un programa en Python que acepte una cadena del usuario y cuente el número de caracteres en la cadena.
- Escribe un programa en Python que tome dos números como entrada del usuario y realice operaciones de multiplicación y división con los números. Las salidas deben mostrarse con los tipos de datos apropiados.
- Escribe un programa en Python que acepte una lista de enteros del usuario y calcule la suma de todos los enteros en la lista. El programa debe mostrar un mensaje de error si el usuario ingresa algún valor no entero en la lista.

En conclusión, la flexibilidad y facilidad de uso de Python con los tipos de datos permite una amplia gama de aplicaciones y lo convierte en una opción ideal para muchas tareas relacionadas con los datos. Comprender los diferentes tipos de datos en Python es un paso clave para volverse competente en el uso del lenguaje.

5.1.5. Análisis de Datos con Python

El **análisis de datos** es el proceso de explorar, transformar y examinar datos para identificar tendencias y patrones que revelen conocimientos importantes y aumenten la eficiencia para apoyar la toma de decisiones.

Python se ha convertido en una opción popular para realizar tareas de análisis de datos. En esta sección, discutiremos los beneficios de usar Python para el análisis de datos y exploraremos algunas de las bibliotecas comúnmente utilizadas en este campo.

Beneficios de Usar Python para el Análisis de Datos:

- **Legibilidad:** El código de Python es conocido por su legibilidad, lo que facilita su comprensión y mantenimiento. Esto lo convierte en una opción ideal para la colaboración y el intercambio de código entre científicos de datos y otros interesados.
- **Comunidad Grande:** Python tiene una comunidad grande y activa que proporciona soporte, tutoriales y documentación. Esto facilita a los principiantes aprender y comenzar con el análisis de datos en Python.
- **Bibliotecas Comprensivas:** Python tiene un rico ecosistema de bibliotecas y herramientas para el análisis de datos, incluyendo NumPy, Pandas, Matplotlib, Seaborn y más. Estas bibliotecas proporcionan una gama de funciones y características para la manipulación, visualización y análisis de datos.

En esta unidad, aprenderás a trabajar con Python y con bibliotecas como NumPy y Pandas:

- **NumPy:** NumPy es una biblioteca para el lenguaje de programación Python que proporciona soporte para arreglos y matrices. Proporciona funciones para operaciones numéricas, como álgebra lineal, generación de números aleatorios y más. NumPy es una biblioteca clave para el análisis de datos, ya que permite la manipulación y el cálculo eficiente de arreglos y matrices.
- **Pandas:** Pandas es una biblioteca para el análisis de datos en Python que proporciona estructuras de datos para almacenar y manipular datos de manera eficiente en un formato tabular. Proporciona características para la lectura y escritura de datos de varias fuentes, limpieza y preprocesamiento de datos, y manipulación de datos. Con Pandas, es fácil realizar tareas como agregar, filtrar y transformar datos, lo que la convierte en una opción popular para el análisis de datos.

En conclusión, Python proporciona un entorno rico y completo para el análisis de datos. Con su facilidad de uso, gran comunidad y bibliotecas comprensivas, Python facilita la realización de tareas de análisis y visualización de datos, lo que lo convierte en una opción ideal para científicos de datos y otros interesados. Esta unidad te ayudará a adquirir las habilidades necesarias para convertirte en un analista de datos.

Co-funded by
the European Union

5.1.6. Tipo de Objeto en Python

Python es un lenguaje de programación versátil que admite varios estilos de programación, como la programación funcional y la programación orientada a objetos (OOP).

La programación orientada a objetos (OOP) es un paradigma de programación que proporciona un medio para estructurar programas de manera que las propiedades y comportamientos se agrupen en objetos individuales. La OOP modela entidades del mundo real como objetos de software que tienen algunos datos asociados y pueden realizar ciertas funciones. Por ejemplo, un objeto podría representar un loro con **propiedades** como nombre, edad, género, ubicación y **comportamientos** como “hablar”, “comer”, “moverse” o “volar”.

En Python, al igual que en otros lenguajes OOP, podemos hacer uso de la OOP a través del uso de **objetos** y **clases.** Usamos una **clase** para definir la estructura de un componente que contiene algunas propiedades y definimos algunos **métodos** que pueden usar (comportamientos) o modificar (acciones) el valor de estas propiedades.
Por ejemplo:

Co-funded by
the European Union

```
class Parrot:

    # class
    attribute name
    = ""
    age = 0

    def speak(self):
        println( f"My name is {name}!")
        println( f"I'm {age} years old!")
```

Cuando creamos una nueva variable con el operador de clase, decimos que hemos creado un objeto de esta clase. Un objeto es la representación en memoria de un elemento creado a partir de una clase. Podemos crear tantos objetos, también llamados instancias de la clase, como queramos. Por ejemplo:

```
# create parrot1 object
parrot1 = Parrot()
parrot1.name = "Blu"
```

Co-funded by
the European Union

```
parrot1.age = 10

# create another object parrot2
parrot2 = Parrot() parrot2.name
= "Woo" parrot2.age = 15

# access
attributes
parrot1.speak()
parrot2.speak()
```

Este es el resultado de ejecutar el código anterior:

```
My name is Blu
I'm 10 years old
My name is Woo
I'm 15 years old
```

Cuando desarrollamos en Python, todos los elementos que usamos como variables, cadenas, números, listas y funciones, internamente están construidos sobre clases. Por esta razón, podríamos decir que en Python todo es un objeto. El tipo de objeto determina qué tipo de datos puede contener y qué operaciones se pueden realizar sobre él.

Hay diferentes tipos de objetos con diferentes funciones, como hemos visto anteriormente al aprender sobre los tipos de datos de Python. Estos tipos de objetos integrados forman la base de las capacidades de procesamiento de datos de Python, y los usarás extensamente al trabajar con datos en Python.

Co-funded by
the European Union

5.2. NumPy y Pandas

NumPy (Numerical Python) es una biblioteca para el lenguaje de programación Python que proporciona soporte para arreglos y matrices. Proporciona funciones para operaciones numéricas, como álgebra lineal, generación de números aleatorios y más. NumPy es una biblioteca clave para el análisis de datos, ya que permite la manipulación y el cálculo eficientes de arreglos y matrices.

Una de las características clave de NumPy es su capacidad para realizar operaciones elemento por elemento en arreglos y matrices, lo que permite una computación rápida y eficiente. NumPy también proporciona funciones para leer y escribir arreglos en y desde el disco, lo que facilita el trabajo con grandes conjuntos de datos.

Pandas (Python Data Analysis Library) es una biblioteca para el análisis de datos en Python que proporciona estructuras de datos para almacenar y manipular datos de manera eficiente en un formato tabular. Proporciona características para la lectura y escritura de datos de varias fuentes, limpieza y preprocesamiento de datos, y manipulación de datos. Con Pandas, es fácil realizar tareas como agregar, filtrar y transformar datos, lo que la convierte en una opción popular para el análisis de datos.

Una de las características clave de **Pandas** es su estructura de datos DataFrame, que proporciona una forma conveniente y poderosa de representar y manipular datos tabulares. Los DataFrames soportan una variedad de operaciones, incluyendo el filtrado de filas y columnas, operaciones de groupBy, y la fusión y unión con otras estructuras de datos.

Pandas también proporciona funciones para manejar datos faltantes y trabajar con datos de series temporales. Pandas es un paquete de Python que proporciona estructuras de datos similares a DataFrame de R. Pandas depende de NumPy, la biblioteca que agrega un tipo de matriz potente a Python.

Juntas, **NumPy** y **Pandas** forman la columna vertebral del ecosistema de análisis de datos de Python, proporcionando la base para una amplia gama de tareas de análisis de datos, desde la manipulación y visualización de datos simples hasta el aprendizaje automático complejo y el modelado estadístico.

5.2.1. Installing Pandas and NumPy libraries

NumPy y Pandas están preinstalados con **Anaconda**, por lo que no necesitas preocuparte por la instalación. Puedes importar las bibliotecas en tu código Python utilizando las siguientes declaraciones de importación:

```
import pandas as pd
import numpy as np
```

Estas declaraciones de importación crean un alias para cada biblioteca, de modo que puedas referirte a ellas como *np* y *pd* respectivamente, facilitando la llamada a funciones de estas bibliotecas en tu código.

5.2.2. Series de datos en Pandas

Una Serie de Pandas es un objeto unidimensional similar a un array etiquetado en Python. Se asemeja a una columna en una hoja de cálculo o a una tabla en una base de datos, y puede almacenar datos de cualquier tipo, incluidos enteros, números de punto flotante, cadenas e incluso otros objetos.

Puede crearse pasando una lista de valores al constructor de objetos de Pandas *(pd series),* o utilizando un diccionario para crear una serie con índices etiquetados:

Co-funded by
the European Union

```
import pandas as
pd
import numpy as
np
# Creating a series from a list
s = pd.Series([1, 3, 5, np.nan, 6, 8])
print(s)
# Output:
# 0    1.0
# 1    3.0
# 2    5.0
# 3    NaN
# 4    6.0
# 5    8.0
# dtype: float64
# Creating a series from a dictionary
data = {'a': 0., 'b': 1., 'c': 2.}
s =
pd.Series(data)
print(s)
# Output:
# a    0.0
# b    1.0
# c    2.0
# dtype: float64
```

Puedes acceder a los elementos en una Serie de Pandas utilizando indexación, de manera similar a como lo harías en un diccionario o un array:

```
import pandas as
pd import numpy
as np
s = pd.Series([1, 3, 5, np.nan, 6, 8], index=['A', 'B', 'C', 'D', 'E', 'F'])

# Accessing elements by index
print(s['A']) # Output: 1.0

# Accessing multiple elements by index
print(s[['A', 'B', 'C']])
# Output:
# A    1.0
# B    3.0
# C    5.0
# dtype: float64
```

Las **Series de Pandas** son una estructura de datos versátil y flexible que se utiliza ampliamente en el análisis y manipulación de datos. Son fáciles de crear y manipular, y ofrecen una amplia gama de funcionalidades para trabajar con datos.

5.2.3. DataFrame

Un **DataFrame** es una estructura de datos que organiza los datos en una tabla bidimensional de filas y columnas, similar a una hoja de cálculo. Los **DataFrames** son una de las estructuras de datos más comunes utilizadas en el análisis de datos moderno debido a que son una forma flexible e intuitiva de almacenar y trabajar con datos.

Cada **DataFrame** contiene un esquema, conocido como schema, que define el nombre y el tipo de datos de cada columna. Los **DataFrames de Spark** pueden contener tipos de datos universales como StringType e IntegerType, así como tipos de datos específicos de Spark, como StructType. Los valores faltantes o incompletos se almacenan como valores nulos (null) en el **DataFrame.**

En resumen, es una estructura de datos bidimensional en Python que se utiliza para almacenar y manipular datos etiquetados. Es un componente clave en la **biblioteca Pandas** y es similar a una tabla en una base de datos relacional.

Un **DataFrame** puede crearse pasando un diccionario de listas o un array bidimensional de **NumPy al** constructor pd.DataFrame:

```
import pandas as
pd import numpy
as np
# Creating a DataFrame from a dictionary data
= {'Name': ['John', 'Jane', 'Jim', 'Joan'],
      'Age': [32, 28, 41, 35],
      'Country': ['USA', 'UK', 'Canada', 'Australia']} df
= pd.DataFrame(data)
print(df
) #
Output:                Count
# 0     Name Age       ry
John        2          USA
# 1         2          UK
Jane        8
                  Canada
# 2         41 35
Jim             Australia
# 3
Joan
```

Co-funded by
the European Union

Un **DataFrame** tiene etiquetas tanto para filas como para columnas, y puedes acceder a columnas individuales en un **DataFrame** como si fueran atributos:

```
import pandas as
pd import numpy
as np
# Accessing a column in a DataFrame
print(df['Name'])
# Output:
# 0    John
# 1    Jane
# 2    Jim
# 3    Joan
# Name: Name, dtype: object
```

Además del acceso y manipulación básicos de datos, **Pandas** proporciona funcionalidades para realizar operaciones más avanzadas en **DataFrames,** como filtrado, agrupamiento y agregación de datos.

```
# Filtering a DataFrame
print(df[df['Age'] > 35])
# Output:
#       Name Age
Country
# 2 Jim 41 Canada


# Grouping a DataFrame
grouped = df.groupby(by='Country')
print(grouped['Age'].mean())
# Output:
# Country
```

Co-funded by
the European Union

```
# Australia    35.0
# Canada       41.0
# UK   28.0
# USA 32.0
# Name: Age, dtype: float64
```

En conclusión, el DataFrame es una estructura de datos poderosa en **Python** que proporciona una forma flexible y eficiente de almacenar, manipular y analizar datos etiquetados. Es un componente clave de la **biblioteca Pandas** y se utiliza ampliamente en la comunidad de ciencia de datos y análisis para la limpieza, preparación y análisis de datos.

Preguntas:

- Escribe un programa en Python que lea un archivo CSV y lo convierta en un DataFrame de Pandas. El programa debería mostrar las primeras cinco filas del DataFrame.
- Escribe un programa en Python que cree un nuevo DataFrame a partir de un DataFrame existente eliminando una columna especificada. El programa debería mostrar el DataFrame original y el nuevo DataFrame.
- Escribe un programa en Python que fusione dos DataFrames basados en una columna común. El programa debería mostrar el DataFrame fusionado.

Co-funded by
the European Union

5.3. Desarrollo en Python Pandas

Python Pandas es una poderosa y ampliamente utilizada biblioteca de código abierto que revoluciona el análisis y la manipulación de datos al proporcionar estructuras de datos y funciones que simplifican el proceso de trabajar con datos estructurados, como tablas y series temporales.

Se integra perfectamente con otras bibliotecas de Python y ofrece formas intuitivas de limpiar, transformar y explorar datos, convirtiéndose en la opción preferida entre científicos de datos, analistas e investigadores. Con su amplia funcionalidad y rendimiento eficiente, Pandas se ha convertido en una herramienta indispensable para el desarrollo basado en datos, permitiendo a los profesionales extraer ideas valiosas y desbloquear todo el potencial de sus conjuntos de datos.

En esta sección, pasaremos a un enfoque práctico de su uso.

5.3.1. Estadísticas con Pandas

Pandas proporciona varios métodos para calcular estadísticas básicas como la media, la mediana y la desviación estándar. Por ejemplo, puedes utilizar el método mean() para calcular la media de todas las columnas numéricas en un DataFrame, o el método median() para calcular la mediana de una columna específica.

Además de las estadísticas básicas, **Pandas** también proporciona herramientas estadísticas más avanzadas, como cálculos de correlación y covarianza. El método corr() se puede utilizar para calcular la correlación

entre todas las columnas numéricas en un **DataFrame**, mientras que el método cov() se puede utilizar para calcular la covarianza entre dos columnas.

Otra característica útil de **Pandas** es su capacidad para manejar datos faltantes. La biblioteca proporciona varios métodos para manejar datos faltantes, como fillna(), que se puede utilizar para llenar valores faltantes con un valor especificado, o dropna(), que se puede utilizar para eliminar filas o columnas que contengan datos faltantes.

Aquí tienes algunos ejemplos prácticos:

- Supongamos que tienes un conjunto de datos de salarios de empleados, con columnas para nombre, edad, salario y departamento. Quieres calcular algunas estadísticas básicas para la columna de salarios, así como la correlación entre salario y edad. También deseas manejar cualquier dato faltante en el conjunto de datos:

  Primero, importarías la biblioteca Pandas y leerías el archivo CSV que contiene el conjunto de datos:

```
import pandas as pd
df = pd.read_csv('employee_salaries.csv')
```

A continuación, puedes usar el método describe() para obtener algunas estadísticas básicas de las columnas numéricas en el DataFrame, incluyendo la media y la desviación estándar de la columna de salario:

```
1. print(df.describe())
```

Output:

1.

	age	salary
count	100.000000	100.0000000
mean	35.120000	5550.000000
std	7.40410200	2128.005684
min	22.000000	2000.000000

```
6. 25%    29.0000000    4000.000000
7. 50%    35.0000000    5500.000000
8. 75%    40.0000000    7000.000000
9. max    50.0000000    9500.000000
```

Para calcular la correlación entre salario y edad, puedes utilizar el método corr():

```
1. print(df['salary'].corr(df['age'])) 2.
Output: 0.23590320142303756
```

Esto indica una correlación positiva débil entre salario y edad. Si el conjunto de datos contiene datos faltantes, puedes utilizar el método fillna() para completar los valores faltantes con un valor especificado. Por ejemplo, para llenar los valores faltantes de salario con el salario promedio, podrías hacer:

```
1. df['salary'].fillna(df['salary'].mean(), inplace=True)
```

Alternativamente, podrías utilizar el método dropna() para eliminar cualquier fila que contenga datos faltantes:

```
1. df.dropna(inplace=True)
```

Estos son solo algunos ejemplos de las muchas herramientas y métodos estadísticos proporcionados por Pandas para trabajar con datos.

EJERCICIO

1. Escribe un programa en Python que lea un archivo CSV y utilice el método corr() para calcular la correlación entre todas las columnas numéricas en el archivo. El programa debería mostrar la matriz de correlación.

Haz clic AQUÍ para descargar el archivo CSV (consejo: Debería estar en la misma carpeta que el programa Python).

5.3.2. Llenando un DataFrame vacío

Un **DataFrame** puede crearse con listas, diccionarios, arrays de NumPy y series. Para crear un DataFrame vacío, no se necesitan pasar argumentos a la clase DataFrame de Pandas. En este ejemplo, creamos un DataFrame vacío y lo imprimimos en la salida de la consola. Como no hemos proporcionado ningún argumento, el arreglo de columnas está vacío y el arreglo de índices también está vacío.

Aquí tienes un ejemplo:

```
import pandas as pd
df = pd.DataFrame()
print(df)
```

5.3.3. Llenando un DataFrame

Añade una fila al **DataFrame** "DataFrame" y agregaremos una nueva fila a este DataFrame. La nueva fila se inicializa como un **diccionario de Python** y se utiliza la función "append()" para agregar la fila al DataFrame. Al agregar un diccionario de Python a append(), asegúrate de pasar "ignore_index=True".

Ejemplo:

```
1. Import pandas as pd 2.
   # create an empty DataFrame
   df = pd.DataFrame() 5.
   # add a new row to the DataFrame
   new_row = {'Name': 'John', 'Age': 30, 'City': 'New York'}
   df = df.append(new_row, ignore_index=True) 9.
10. print(df)
11.
```

En este ejemplo, creamos un DataFrame vacío utilizando el constructor DataFrame() de Pandas. Luego, creamos una nueva fila como un diccionario de Python y utilizamos la función append() para agregar la fila al DataFrame. Pasamos ignore_index=True para asegurarnos de que el índice no se asigne automáticamente según la posición de la fila en el DataFrame.

Co-funded by
the European Union

El resultado será:

Como puedes ver, la nueva fila ha sido agregada al DataFrame con los nombres de columna y valores especificados.

EJERCICIO
Añade una nueva fila al DataFrame existente (el ejemplo anterior).

5.3.4. Tamaño de un DataFrame

En **Python**, la propiedad size del **DataFrame** de **Pandas** se utiliza para mostrar el tamaño del DataFrame de Pandas. Devuelve el tamaño del DataFrame o una serie que es equivalente al número total de elementos. Si deseas calcular el tamaño de la serie, devolverá el número de filas.

Ejemplo:

```
1. import pandas as
pd 2.
   # create a DataFrame
df = pd.DataFrame({'Name':
['John', 'Alice', 'Bob'], 'Age':
[30, 25, 35], 'City': ['New York', 'Paris', 'London']})
5.
   # display the size of the DataFrame
print("Size of the DataFrame: ", df.size) 8.
```

```
# create a Series
s = pd.Series([10, 20, 30, 40])

# display the size of the Series
print("Size of the Series: ", s.size)
```

En este ejemplo, creamos un **DataFrame de Pandas** con tres columnas: Name, Age y City. Luego utilizamos el atributo size para mostrar el tamaño del DataFrame, que es igual al número total de elementos en el DataFrame (es decir, el número de filas multiplicado por el número de columnas).

Después, creamos una **Serie de Pandas** con cuatro valores. Utilizamos nuevamente el atributo size para mostrar el tamaño de la Serie, que es igual al número de filas en la Serie.

La salida de este código será:

```
Size of the DataFrame: 9
Size of the Series: 4
```

EJERCICIO

- Crea un nuevo DataFrame con 2 columnas: "Product" y "Price".
- Añade 3 filas al DataFrame con los siguientes datos: "Product A", $10, "Product B", $20 y "Product C", $30.
- Muestra el tamaño del DataFrame utilizando el atributo size.
- Crea una nueva Serie con 5 valores

Co-funded by
the European Union

5. Muestra el tamaño de la Serie utilizando el atributo size.

5.3.5. Agregar una nueva columna a un DataFrame

Podemos usar los métodos assign() e insert() de los objetos **DataFrame** para agregar una nueva columna al DataFrame existente con valores predeterminados. También podemos asignar directamente un valor predeterminado a la columna del DataFrame que se va a crear.

Ejemplo:

```
1. import pandas as pd 2.
   # create a DataFrame with three columns
   df = pd.DataFrame({'name': ['Alice', 'Bob', 'Charlie'], 'age': [25, 30,
   35], 'gender': ['female', 'male', 'male']})
5.
   # add a new column 'height' to the DataFrame with default value of 170
   df = df.assign(height=170) 8.
9. print(df)
```

En este ejemplo, creamos un DataFrame df con tres columnas: name, age y gender. Luego utilizamos el método assign() para agregar una nueva columna llamada height con un valor predeterminado de 170. El método assign() devuelve un nuevo DataFrame con la columna añadida, que luego asignamos de nuevo a df.

Cuando ejecutamos este código, obtenemos la siguiente salida:

```
1.      name age gender height
2. 0     Alice   25   female     170
3. 1       Bob   30     male     170
4. 2   Charlie   35     male     170
```

Como puedes ver, se ha añadido la nueva columna "height" al DataFrame con el valor predeterminado de 170.

EJERCICIO
Añade una nueva columna y fila al DataFrame existente (el ejemplo a continuación)

```
1. import pandas as
pd 2.
3.      df      =                                   ['Alice', 'Bob', 'Charlie'], 'age':
pd.DataFrame({'name':
[25, 30, 35], 'gender': ['female', 'male', 'male']})
4.
5. print(df)
```

5.3.6. Creando una tabla vacía

Puedes crear una lista vacía en **Python** utilizando estas dos alternativas:

1. Primero, puedes usar corchetes vacíos [], lo cual tiende a ser más conciso y rápido.
2. Además, puedes usar el constructor de tipo list() sin especificar argumentos, lo cual también crea una lista vacía.

Ejemplo:
Usando corchetes vacíos []

```
empty_list = []
print(empty_list)
Output: []
```

Usando el constructor list():

```
empty_list = list()
print(empty_list)
Output: []
```

5.3.7. Tipo de datos de una columna

Podemos usar el comando dtypes para ver el tipo de datos de cada columna de un DataFrame (todos a la vez).
Ejemplo:

```
1. import pandas as
pd 2.
3. # Create a sample dataframe
4 d = {'Nam ['John', 'Mary', 'Peter', 'David'], 'Age':
. at e':
[30, 25, 40, 35], 'Salary': [50000, 60000, 70000, 45000]}
a
5. df = pd.DataFrame(data)
6.
7. # Use dtypes command to see the datatype of each column
```

```
8.
print(df.dtypes)
```

Output:

```
Name:     object
Age:      int64
Salary:   int64
dtype:    object
```

El comando dtypes muestra el tipo de datos de cada columna en el DataFrame. En este ejemplo, la columna "Name" es de tipo object (una cadena de texto), mientras que las columnas 'Age' y 'Salary' son de tipo int64 (entero).

5.3.8. Listas

Para crear una lista en **Python,** debemos colocar todos los elementos que queremos añadir a esa lista entre corchetes cuadrados [ ] y separados por comas. Las listas pueden contener tantos elementos como queramos y, como mencionamos, pueden ser de diferentes tipos (enteros, cadenas de texto, booleanos, etc.).

Example:

```
1. # Create a list of integers
2. int_list = [1, 2, 3, 4, 5]
3.
   # Create a list of strings
   str_list = ['apple', 'banana', 'cherry']
   6.
```

```
# Create a list of mixed types
mixed_list = [1, 'apple', True, 3.14] 9.

# Create an empty list
empty_list = []
```

En este ejemplo, creamos cuatro listas diferentes: int_list que contiene enteros, str_list que contiene cadenas de texto, mixed_list que contiene una mezcla de diferentes tipos, y empty_list que es una lista vacía. Observa que cada elemento está separado por una coma y todos los elementos están encerrados entre corchetes cuadrados.

5.3.9. Indices

Para encontrar el índice de la primera ocurrencia de un elemento en una lista de Python dada, puedes utilizar el método "index()" de la clase "List" con el elemento pasado como argumento. El método "index()" devuelve un entero que representa el índice de la primera ocurrencia del elemento especificado en la lista.

```
1. my_list = [3, 8, 2, 6, 1, 7, 4]
2.
3. index_of_six = my_list.index(6)
4.
5. print("The index of the first occurrence of 6 is:", index_of_six) 6.
   Output:
   The index of the first occurrence of 6 is: 3
   9.
```

Co-funded by the European Union

En el ejemplo anterior, tenemos una lista llamada 'my_list' con algunos o utilizamos el método index() para encontrar el índice de la primera ocurrencia del elemento 6 en la lista y lo almacenamos en una variable llamada index_of_six. Finalmente, imprimimos el valor de index_of_six en la consola.

5.3.10. Conversiones de tipos de datos

1. int (x) Convierte x a un entero.
2. float (x) Convierte x a un número de punto flotante..
3. str (x) Convierte x a una cadena de texto (Text string).
4. hex (x) Convierte el entero x a una cadena hexadecimal.
5. chr (x) Convierte el entero x a un carácter.

Ejemplo:

int(x):

```
num_string = "123"
num_int = int(num_string)
print(num_int) # Output: 123
```

float(x):

```
num_string = "3.14"
num_float = float(num_string)
print(num_float) # Output: 3.14
```

```
num_int = 123
num_str = str(num_int)
```

```
3. print(num_str) # Output: "123"
```

hex(x):

```
num_int = 255
num_hex = hex(num_int)
print(num_hex) # Output: "0xff"
```

chr(x):

```
ascii_code = 65
char = chr(ascii_code)
print(char) # Output: "A"
```

5.3.11. Joining dos DataFrame

Los dos **DataFrames** que queremos unir se pasan a la función merge utilizando los argumentos left y right. El argumento left_on='species' indica a merge que utilice la columna species_id como la clave de unión de survey_sub (el DataFrame izquierdo).

Supongamos que tenemos dos DataFrames:

- Un DataFrame de clientes con las columnas 'customer_id', 'customer_name', 'customer_address', 'customer_phone'.
- Un DataFrame de ventas con las columnas 'transaction_id', 'transaction_date', 'customer_id', 'product_id', 'quantity'.

Queremos unir estos dos DataFrames para obtener un nuevo DataFrame que contenga todos los detalles de las transacciones junto con la información correspondiente del cliente. Podemos utilizar la función merge de la siguiente manera:

```
# importing pandas library
import pandas as
pd 3.
# reading customer and sales data from CSV files
customer_df = pd.read_csv('customer.csv')
sales_df = pd.read_csv('sales.csv')
7.
# merging customer and sales dataframes on customer_id column
merged_df=      pd.merge(sales_df,   customer_df, on='customer_id',
how='left')

# displaying the merged dataframe
print(merged_df.head())
```

En este ejemplo, se utiliza la función *pd.merge()* para unir los DataFrames *sales_df* y customer_df en la columna *'customer_id'* con un tipo de *unión 'left'.* Esto mantendrá todos los registros del DataFrame sales_df y coincidirá la información del cliente para los valores correspondientes de 'customer_id'. El DataFrame resultante merged_df contendrá todos los datos de ventas con la información correspondiente del cliente.

Co-funded by
the European Union

5.3.12. Group by

GroupBy es un concepto bastante simple. Podemos crear agrupaciones de categorías y aplicar una función a esas categorías. Es un concepto sencillo, pero es una técnica extremadamente valiosa que se utiliza ampliamente en la ciencia de datos.

Digamos que tenemos un **DataFrame** que contiene datos de ventas para una tienda con las columnas 'Date', 'Product', 'Category', 'Price' y 'Quantity':

Podemos usar **GroupBy de Pandas** para agrupar los datos de ventas por Categoría y calcular las ventas totales para cada Categoría:

```
 # group by Category and sum sales
sales_by_category = sales_df.groupby('Category')['Price'].sum() 3.

 # print sales by category
print(sales_by_category

) 6.
```

Esto producirá:

```
Category
Category 1 36.97
Category 2 22.98
Name: Price, dtype: float64
```

.

Aquí, estamos agrupando los datos de ventas por la columna 'Category' y luego sumando la columna 'Price' para cada categoría utilizando la *función sum().* La variable *sales_by_category* resultante contiene las ventas totales para cada categoría.

Co-funded by
the European Union

Referencias

Online referencias

Coursera. (n.d.). Data Analysis Software [Article]. Retrieved from https://www.coursera.org/articles/data-analysis-software

Coursera. (n.d.). Introduction to Python [Project]. Retrieved from https://www.coursera.org/projects/introduction-to-python

Coursera. (n.d.). Popular Programming Languages [Article]. Retrieved from https://www.coursera.org/articles/popular-programming-languages

Coursera. (n.d.). Python for Everybody Specialization [Course]. Retrieved from https://www.coursera.org/specializations/python?trk_ref=articleProductCard

Coursera. (n.d.). Python or R for Data Analysis [Article]. Retrieved from https://www.coursera.org/articles/python-or-r-for-data-analysis

Coursera. (n.d.). What is Data Analysis with Examples [Article]. Retrieved from https://www.coursera.org/articles/what-is-data-analysis-with-examples

Coursera. (n.d.). What is Python Used For? A Beginner's Guide to Using Python [Article]. Retrieved from https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python

Co-funded by
the European Union

Learn to Code with Me. (n.d.). What Does a Data Analyst Do? A Career Guide [Article]. Retrieved from https://www.coursera.org/articles/what-does-a-data-analyst-do-a-career-guide

Learn to Code with Me. (n.d.). What is Data Analysis? And How Can You Start Learning It Today? [Blog post]. Retrieved from https://learntocodewith.me/posts/data-analysis/

Quora. (n.d.). What is data analytics and what are the programming languages used in data analytics? [Question]. Retrieved from https://www.quora.com/What-is-data-analytics-and-what-are-the-programming-languages-used-in-data-analytics

Real Python. (n.d.). Real Python Tutorials [Website]. Retrieved from https://realpython.com/

TechTarget. (n.d.). Data Analytics [Definition]. Retrieved from https://www.techtarget.com/searchdatamanagement/definition/data-analytics

W3Schools. (n.d.). Python Tutorial [Website]. Retrieved from https://www.w3schools.com/python/

YouTube. (n.d.). Data Analytics [Video]. Retrieved from https://www.youtube.com/watch?v=OQUfnEJvMXk

Bibliography

Carvalho, A. (2021). Práticas de Python – Algoritmia e programação. FCA.

Costa, E. (2015). Programação em Python – Fundamentos e Resolução de problemas. FCA.

Co-funded by
the European Union