Data Science y Big Data aplicados a la Economía y a la Administración y Dirección de Empresas

Contents

Tutorial de Introducción a Python para Data Science. Realizado por: Pablo Sánchez Cabrera

Introducción a Python

1. Comenzando con Python...

```
print("Hola Mundo")
```

```
Hola Mundo
```

```
print("Quiero aprender Python")
```

```
Quiero aprender Python
```

2. Tipado de datos

Skip to main content

2.1 Cadenas de caracteres

```
cadena = "Hola, estoy aprendiendo los tipos de datos en Python"
```

```
cadena
```

```
'Hola, estoy aprendiendo los tipos de datos en Python'
```

```
print(cadena)
```

```
Hola, estoy aprendiendo los tipos de datos en Python
```

```
type(cadena)
```

```
str
```

```
len(cadena)
```

```
52
```

2.2 Número de enteros

```
num = 10
```

```
num
```

```
10
```

```
type(num)
```

Skip to main content

```
int
```

```
num2 = 10
```

```
num*num2
```

```
100
```

```
num2 = 100
```

```
num*num2
```

```
1000
```

```
num3 = num*num2
```

```
num3
```

```
1000
```

```
num4 = int(5)
```

```
num4
```

```
5
```

2.3 Números decimales

```
num = 10.4
```

Skip to main content

```
num
```

```
10.4
```

```
type(num)
```

```
float
```

```
num2 = float(num)
```

```
num2
```

```
10.4
```

```
type(num2)
```

```
float
```

2.4 Booleanos

```
a = True
```

```
b = False
```

```
type(a)
```

```
bool
```

```
type(b)
```

Skip to main content

```
bool
```

```
a|b
```

```
True
```

```
a&b
```

```
False
```

2.5 Listas

Las listas son una colección ordenada y mutable de valores separados por comas y entre corchetes. Pueden estar compuestas de diferentes tipos de valores e, incluso, Python permite la creación de listas de listas.

- Definición de una lista

```
lista = [cadena, num, num2, num3, a, b]
```

```
lista
```

```
['Hola, estoy aprendiendo los tipos de datos en Python',
10.4,
10.4,
1000,
True,
False]
```

```
len(lista) # nº elementos de la lista
```

```
6
```

- Elementos de una lista

```
lista[0] # primer elemento de la lista
```

Skip to main content

```
'Hola, estoy aprendiendo los tipos de datos en Python'
```

```
lista[5] # quinto elemento de la lista
```

```
False
```

```
lista[1:3] # segundo y tercer elemento de la lista
```

```
[10.4, 10.4]
```

```
lista[:2] # los dos primeros elementos de la lista
```

```
['Hola, estoy aprendiendo los tipos de datos en Python', 10.4]
```

```
lista[6]
```

```
---------------------------------------------------------------------------
IndexError Traceback (most recent call last)
Cell In[38], line 1
----> 1 lista[6]

IndexError: list index out of range
```

Solo hay cinco elementos en la lista, por eso el error. Python empieza en cero!!

```
lista[::2] # elementos de la lista (1 salto)
```

```
['Hola, estoy aprendiendo los tipos de datos en Python', 10.4, True]
```

- Copiar una lista en otra

```
lista_copia = lista.copy() # muy útil cuando se quiere trabajar con listas que tienen elementos comunes
```

```
lista_copia
```

Skip to main content

```
['Hola, estoy aprendiendo los tipos de datos en Python',
10.4,
10.4,
1000,
True,
False]
```

```
prueba = lista
```

```
prueba
```

```
['Hola, estoy aprendiendo los tipos de datos en Python',
10.4,
10.4,
1000,
True,
False]
```

Crear un objeto como igual a otro no es adecuado, usar siempre *copy*

- Cambiar y agregar elementos a una lista

```
lista_copia[0] = "Ya sé algo de tipos de datos"
```

```
lista_copia
```

```
['Ya sé algo de tipos de datos', 10.4, 10.4, 1000, True, False]
```

```
prueba[4] = "Pablo"
```

```
prueba
```

```
['Hola, estoy aprendiendo los tipos de datos en Python',
10.4,
10.4,
1000,
'Pablo',
False]
```

```
lista
```

Skip to main content

```
['Hola, estoy aprendiendo los tipos de datos en Python',
10.4,
10.4,
1000,
'Pablo',
False]
```

```
lista_copia.append(200)
```

```
lista_copia
```

```
['Ya sé algo de tipos de datos', 10.4, 10.4, 1000, True, False, 200]
```

```
lista_2 = [0, "Pablo", "curso UAH", 25] # creación de una segunda lista
```

```
lista_copia.append(lista_2) # segunda lista añadida al último elemento de la lista inicial
```

```
lista_copia
```

```
['Ya sé algo de tipos de datos',
10.4,
10.4,
1000,
True,
False,
200,
[0, 'Pablo', 'curso UAH', 25]]
```

```
lista_copia[7]
```

```
[0, 'Pablo', 'curso UAH', 25]
```

```
lista_copia.extend(lista_2) # se añade los elementos de la segunda lista a la lista inicial
(copia)
```

```
lista_copia
```

Skip to main content

```
['Ya sé algo de tipos de datos',
10.4,
10.4,
1000,
True,
False,
200,
[0, 'Pablo', 'curso UAH', 25],
0,
'Pablo',
'curso UAH',
25]
```

- Eliminar elemento de la lista

```
lista_copia.pop(7) # elimina el séptimo elemento de la lista
```

```
[0, 'Pablo', 'curso UAH', 25]
```

```
lista_copia
```

```
['Ya sé algo de tipos de datos',
10.4,
10.4,
1000,
True,
False,
200,
0,
'Pablo',
'curso UAH',
25]
```

```
lista.pop() # elimina el último elemento de la lista
```

```
False
```

```
lista
```

```
['Hola, estoy aprendiendo los tipos de datos en Python',
10.4,
10.4,
1000,
'Pablo']
```

Skip to main content

2.6 Tuplas

Las tuplas son un tipo de objeto de Python muy similar a las listas. Se diferencian en su sintaxis (uso de paréntesis en vez de corchetes) y en que su contenido no puede ser modificado.

- Definición de la tupla

```
tupla = (1,2,3,4,5,6,7,8,9)
```

```
tupla
```

```
(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

```
len(tupla)
```

```
9
```

- Elementos de la tupla

```
tupla[0]
```

```
1
```

```
tupla.index(3)
```

```
2
```

- Cambiar elemento de la tupla

```
tupla[0] = "Hola"
```

```
---------------------------------------------------------------------------
TypeError Traceback (most recent call last)
Cell In[65], line 1
----> 1 tupla[0]="Hola"

TypeError: 'tuple' object does not support item assignment
```

Skip to main content

El error se debe a que las tuplas son inmutables

2.7 Diccionarios

Los diccionarios son una colección desordenada, mutable e indexada de un par clave-valor; esto es, representan un *array* asociativo con el que representar con una clavea un determinado valor.

- Definición de un diccionario

```
diccionario = {"cadena": cadena, "entero": num3, "decimal": num, "booleano": a, "lista": lista}
```

```
diccionario
```

```
{'cadena': 'Hola, estoy aprendiendo los tipos de datos en Python',
'entero': 1000,
'decimal': 10.4,
'booleano': True,
'lista': ['Hola, estoy aprendiendo los tipos de datos en Python',
10.4,
10.4,
1000,
'Pablo']}
```

- Elemento de un diccionario

```
diccionario['decimal'] # a partir de la clave se obtine el valor
```

```
10.4
```

```
diccionario.get("decimal") # también se puede usar el método get
```

```
10.4
```

- Clave y valor de un diccionario

```
list(diccionario.keys())
```

```
['cadena', 'entero', 'decimal', 'booleano', 'lista']
```

Skip to main content

```
list(diccionario.values())
```

```
['Hola, estoy aprendiendo los tipos de datos en Python',
1000,
10.4,
True,
['Hola, estoy aprendiendo los tipos de datos en Python',
10.4,
10.4,
1000,
'Pablo']]
```

```
diccionario.items() # también items
```

```
dict_items([('cadena', 'Hola, estoy aprendiendo los tipos de datos en Python'), ('entero',
1000), ('decimal', 10.4), ('booleano', True), ('lista', ['Hola, estoy aprendiendo los tipos de
datos en Python', 10.4, 10.4, 1000, 'Pablo'])])
```

- Añadir y eliminar elementos de un diccionario

```
diccionario["final"] = "Fin del diccionario"
```

```
diccionario
```

```
{'cadena': 'Hola, estoy aprendiendo los tipos de datos en Python',
'entero': 1000,
'decimal': 10.4,
'booleano': True,
'lista': ['Hola, estoy aprendiendo los tipos de datos en Python',
10.4,
10.4,
1000,
'Pablo'],
'final': 'Fin del diccionario'}
```

```
diccionario.pop("final")
```

```
'Fin del diccionario'
```

```
diccionario
```

Skip to main content

```
{'cadena': 'Hola, estoy aprendiendo los tipos de datos en Python',
'entero': 1000,
'decimal': 10.4,
'booleano': True,
'lista': ['Hola, estoy aprendiendo los tipos de datos en Python',
10.4,
10.4,
1000,
'Pablo']}
```

```
diccionario.pop("lista")
```

```
['Hola, estoy aprendiendo los tipos de datos en Python',
10.4,
10.4,
1000,
'Pablo']
```

```
diccionario
```

```
{'cadena': 'Hola, estoy aprendiendo los tipos de datos en Python',
'entero': 1000,
'decimal': 10.4,
'booleano': True}
```

3. Estructuras de flujo

3.1 Sentencias condicionales

Permite trabajar con diferentes condiciones (if, elif o else) de forma que, en base a la evaluación de la sentencia, se llevará a cabo la ejecución correspondiente.

Se presentan varios ejemplos de uso:

```
a = 20
b = 10
```

```
a>b
```

```
True
```

Skip to main content

```
a<b
```

```
False
```

```
a==b
```

```
False
```

```
a!=b
```

```
True
```

```
print("a=" + a)
```

```
---------------------------------------------------------------------------
TypeError Traceback (most recent call last)
Cell In[87], line 1
----> 1 print("a=" + a)

TypeError: can only concatenate str (not "int") to str
```

```
type(a)
```

```
int
```

```
str(a)
```

```
'20'
```

```
print("a=" + '_' + str(a))
```

```
a=_20
```

Skip to main content

```
print("a=" + str(a) + "; b=" + str(b))
if a>b:
print("El mayor es " + str(a))
elif a==b:
print("Son iguales")
else:
print("El mayor es " + str(b))
```

```
a=20; b=10
El mayor es 20
```

- Ejemplo 2: ¿Positivo, negativo o neutro?

```
x = 3
if x > 0: # importante la identación en python
print('El número es positivo')
elif x == 0:
print('Es igual a cero')
else:
print('El número es negativo')
```

```
El número es positivo
```

3.2 Bucles

Los bucles son utilizados cuando un mismo bloque del código tiene que ser ejecutado un número “x” de veces. Los dos bucles más destacados en Pyhton son:

- For: permite iterar sobre una colección de valores (listas, tuplas, diccionarios, etc.)
- While: funciona repetidamente mientras que la condición que lo controla sea cierta

For

- Uso de la función *range*

```
# Pintar números desde 1 a 10
for i in range(1,11):
print(i)
```

```
1
2
3
4
5
6
7
8
9
10
```

```
suma=0
for i in range(1,11):
suma = suma+i
print(suma)
```

```
55
```

```
# Pintar números impares (1 a 10)
for i in range(1,11,2):
print(i)
```

```
1
3
5
7
9
```

```
# Tablas de multiplicar
total = 10
hasta = 5

for i in range(1,hasta+1):
print(f'Tabla del {i}')
for j in range(1, total+1):
print(f'{i}*{j}={i*j}')
```

Skip to main content

```
Tabla del 1
1*1=1
1*2=2
1*3=3
1*4=4
1*5=5
1*6=6
1*7=7
1*8=8
1*9=9
1*10=10
Tabla del 2
2*1=2
2*2=4
2*3=6
2*4=8
2*5=10
2*6=12
2*7=14
2*8=16
2*9=18
2*10=20
Tabla del 3
3*1=3
3*2=6
3*3=9
3*4=12
3*5=15
3*6=18
3*7=21
3*8=24
3*9=27
3*10=30
Tabla del 4
4*1=4
4*2=8
4*3=12
4*4=16
4*5=20
4*6=24
4*7=28
4*8=32
4*9=36
4*10=40
Tabla del 5
5*1=5
5*2=10
5*3=15
5*4=20
5*5=25
5*6=30
5*7=35
5*8=40
5*9=45
5*10=50
```

- Iteracción sobre una lista

```python
# Presentación de elementos de la lista en Mayúsculas
fruta = ['naranja', 'manzana', 'fresa', 'uva', 'cereza', 'plátano']
for i in fruta:
print(i.upper())
```

Skip to main content

```
NARANJA
MANZANA
FRESA
UVA
CEREZA
PLÁTANO
```

- Uso de la función **enumerate**

```
nombres = ['Pablo', 'David', 'María', 'Lucía', 'Roberto', 'Paula']
for i in enumerate(nombres):
print(i)
```

```
(0, 'Pablo')
(1, 'David')
(2, 'María')
(3, 'Lucía')
(4, 'Roberto')
(5, 'Paula')
```

- Uso de la función **zip**

```
for i,j in zip(nombres, fruta):
print(i,j)
```

```
Pablo naranja
David manzana
María fresa
Lucía uva
Roberto cereza
Paula plátano
```

```
ciudad = ['Madrid', 'León', 'Barcelona', 'Toledo']
for i,j in zip(nombres, ciudad):
print(i,j)
```

```
Pablo Madrid
David León
María Barcelona
Lucía Toledo
```

- Uso de las **list Comprehension**

```
[i for i in range(1,11)]
```

```
fruta = ['naranja', 'manzana', 'fresa', 'uva', 'cereza', 'plátano']
[i.upper() for i in fruta]
```

```
['NARANJA', 'MANZANA', 'FRESA', 'UVA', 'CEREZA', 'PLÁTANO']
```

```
ciudad_2 = ['León', 'Ávila', 'Salamanca', 'Zamora', 'Palencia', 'Burgos', 'Valladolid',
'Soria', 'Segovia']

[i for i in ciudad if i in ciudad_2]
```

```
['León']
```

```
[n for n in range(10) if n % 2 == 0]
```

```
[0, 2, 4, 6, 8]
```

- Iteracción sobre un diccionario

```
datos = {'Nombre': 'Pablo', 'Apellido': 'Sánchez', 'Estudios': 'Máster Universitario',
'Profesion':'Data Scientist'}
```

```
# solo las claves del diccionario
for i in datos.keys():
print(i)
```

```
Nombre
Apellido
Estudios
Profesion
```

```
# solo los valores del diccionario
for i in datos.values():
print(i)
```

```
Pablo
Sánchez
Máster Universitario
Data Scientist
```

Skip to main content

```
for i, j in zip(datos.keys(), datos.values()):
print(f'{i}: {j}')
```

```
Nombre: Pablo
Apellido: Sánchez
Estudios: Máster Universitario
Profesion: Data Scientist
```

```
# salida en formato tupla
for i in datos.items():
print(i)
```

```
('Nombre', 'Pablo')
('Apellido', 'Sánchez')
('Estudios', 'Máster Universitario')
('Profesion', 'Data Scientist')
```

```
for i,j in datos.items():
print(f'{i}: {j}')
```

```
Nombre: Pablo
Apellido: Sánchez
Estudios: Máster Universitario
Profesion: Data Scientist
```

While

```
i = 1
while i <= 10:
print(i)
i = i + 1
```

```
1
2
3
4
5
6
7
8
9
10
```

Skip to main content

```
i=1
suma=0
while i <= 10:
suma = suma+i
i = i+1
print(suma)
```

```
55
```

```
total = 10
hasta = 5

i = 1
while( i<= 5):
print(f'Tabla de {i}')
j = 1
while( j<= 10):
print(f'{i}*{j}={i*j}')
j = j+1
i=i+1
```

```
Tabla de 1
1*1=1
1*2=2
1*3=3
1*4=4
1*5=5
1*6=6
1*7=7
1*8=8
1*9=9
1*10=10
Tabla de 2
2*1=2
2*2=4
2*3=6
2*4=8
2*5=10
2*6=12
2*7=14
2*8=16
2*9=18
2*10=20
Tabla de 3
3*1=3
3*2=6
3*3=9
3*4=12
3*5=15
3*6=18
3*7=21
3*8=24
3*9=27
3*10=30
Tabla de 4
4*1=4
4*2=8
4*3=12
4*4=16
4*5=20
4*6=24
4*7=28
4*8=32
4*9=36
4*10=40
Tabla de 5
5*1=5
5*2=10
5*3=15
5*4=20
5*5=25
5*6=30
5*7=35
5*8=40
5*9=45
5*10=50
```

4. Funciones

Las funciones permiten aislar partes específicas del código y reutilizarlas en cualquier lugar. Para definir una función Python tiene la palabra reservada **def**.

4.1 Funciones estándar

Se presentan algunos ejemplos para ilustrar su sintaxis:

- Un número es mayor que otro

```
def Mayor(a, b):
if a>b:
return a
else:
return b
```

```
n = Mayor(3, 7)
print(n)
```

```
7
```

```
n = Mayor(10, 2)
print(n)
```

```
10
```

```
n = Mayor(40,40)
print(n)
```

```
40
```

- Sumar dos números cualesquiera

```
def Sumar(x,y):
return x+y
```

```
Sumar(3, 5)
```

```
8
```

```
Sumar(-2,-7)
```

Skip to main content

```
-9
```

- Sumar todos los números existentes en un intervalo

```
def SumaEntre(a, b):

suma = 0
for i in range(a, b+1): # b+1 para que el intervalo sea cerrado (contenga el valor b)
suma = suma + i
return suma
```

```
valor = SumaEntre(1, 5)
print(valor)
```

```
15
```

- Funciones con más de una salida

```
def Operacion_Basica(a,b):
return a+b, a-b
```

```
Operacion_Basica(5,3)
```

```
(8, 2)
```

```
suma, resta = Operacion_Basica(5,3)
```

```
print(suma)
print(resta)
```

```
8
2
```

```
suma, _ = Operacion_Basica(6,9)

_, resta = Operacion_Basica(6,9)
```

```
suma
```

Skip to main content

```
15
```

```
resta
```

```
-3
```

4.2 Funciones anónimas

- Uso de funciones **lambda**

En algunos casos, pueden definirse funciones a través de la sentencia **lambda**. Esta palabra se reserva para la construcción de funciones “anónimas” en Python.

```
Sumar_2 = lambda x,y: x+y
```

```
Sumar_2(3, 5)
```

```
8
```

```
Sumar_2(-2,-7)
```

```
-9
```

5. Excepciones

Las excepciones son una forma de trabajo que disponen los lenguajes de programación para encapsular posibles errores en el código.

A la hora de definir una excepción se utilizan las sentencias *try* y *except* (aunque también puede haber otras cláusulas como *finally* o *else*).

Las excepciones pueden ser utilizadas en cualquier parte de nuestro código.

5.1 Excepciones básicas

Skip to main content

- Divisible entre cero

```
a=5
b=0

a/b
```

```
---------------------------------------------------------------------------
ZeroDivisionError Traceback (most recent call last)
Cell In[129], line 4
1 a=5
2 b=0
----> 4 a/b

ZeroDivisionError: division by zero
```

```
try:
a/b
except Exception as e:
print(e)
```

```
division by zero
```

- Fuera de rango

```
lista = [1,2,3,4,5]
```

```
a = len(lista)
try:
print(lista[a+1])
except Exception as e:
print(e)
```

```
list index out of range
```

- Conversión a mayúsculas de un caracter o palabra

```
def mayuscula(nombre):

try:
print(nombre.upper())
except Exception as e:
print(e)
finally:
print("Ejecución realizada")
```

Skip to main content

```
mayuscula("hola")
```

```
HOLA
Ejecución realizada
```

```
mayuscula(5)
```

```
'int' object has no attribute 'upper'
Ejecución realizada
```

```
def mayuscula2(nombre):

try:
print(nombre.upper())
except:
print("Por favor, revisa que se ha introducido una cadena de caracteres")
finally:
print("Ejecución realizada")
```

```
mayuscula2("3")
```

```
3
Ejecución realizada
```

- Cálculo de raíz cuadra y cuadrado de un número

```
import math

def operacion_matematica(numero):

try:
print(f'La raiz cuadra de {numero} es {math.sqrt(numero)}')
except Exception as e:
print(e)
else:
print(f'El cuadrado de {numero} es {numero*numero}')
```

```
operacion_matematica(5)
```

```
La raiz cuadra de 5 es 2.23606797749979
El cuadrado de 5 es 25
```

Skip to main content

```
operacion_matematica(-1)
```

```
math domain error
```

5.2 Excepciones personalizadas

Este tipo de excepciones son específicas de nuestro código. Se usa la cláusula *raise*.

- Ejemplo 1

```
def profesiones_data(profesion):
lista_profesiones = ["data-scientist", "data-engineer", "machine-learning engineer", "data-analyst"]
profesion = profesion.lower()
if profesion not in lista_profesiones:
raise ValueError(f"{profesion} no es compatible. Solo disponible {lista_profesiones}")
return profesion
```

```
profesiones_data("DATA-scientist")
```

```
'data-scientist'
```

```
profesiones_data("data-SCIENTIST")
```

```
'data-scientist'
```

```
profesiones_data("data SCIENTIST")
```

```
---------------------------------------------------------------------------
ValueError Traceback (most recent call last)
Cell In[158], line 1
----> 1 profesiones_data("data SCIENTIST")

Cell In[154], line 5, in profesiones_data(profesion)
3 profesion = profesion.lower()
4 if profesion not in lista_profesiones:
----> 5 raise ValueError(f"{profesion} no es compatible. Solo disponible {lista_profesiones}")
6 return profesion

ValueError: data scientist no es compatible. Solo disponible ['data-scientist', 'data-engineer',
```

- Ejemplo 2

```
def interes_simple(cantidad, anio, tasa):

tasa_porcentaje = tasa/100

try:
if tasa_porcentaje >1:
raise ValueError(tasa_porcentaje)
interes = (cantidad * anio * tasa_porcentaje)
print(f'La cantidad percibida es de {interes}')
return interes
except ValueError:
print('Tasa de interés fuera de rango')
```

```
interes_simple(1000, 2, 8)
```

```
La cantidad percibida es de 160.0
```

```
160.0
```

```
interes_simple(1000, 2, 12)
```

```
La cantidad percibida es de 240.0
```

```
240.0
```

```
interes_simple(1000, 2, 150)
```

```
Tasa de interés fuera de rango
```

6. Clases

Las clases son constructoras de objetos y constituyen una parte esencial en la programación orientada a objetos. Las clases se componenten de métodos que definen la clase y sus operaciones.

Una clase se define con la instrucción **class** y se inicia con la sentencia *init* que es el constructor de la misma. Por su parte, los métodos se inician con la instrucción *self* (en general).

Skip to main content

- Operaciones matemáticas

```
class Calculadora_basica:

def __init__(self, a, b):
self.a = a
self.b = b

def suma(self):
return self.a + self.b

def resta(self):
return self.a - self.b

def producto(self):
return self.a*self.b

def division(self):
return self.a/self.b
```

Se llama a la clase para definir el objeto y, posteriormente, a los respectivos métodos

```
operaciones = Calculadora_basica(5, 8) # argumentos de entrada (números 5, 8)
```

El objeto *operaciones* dispondrá de la funcionalidad necesaria para realizar las operaciones básicas (5,8)

```
# se llama a los respectivos métodos
print(f'Operaciones básicas - datos de entrada: {operaciones.a} y {operaciones.b}')
print(f'\n Suma: {operaciones.suma()}')
print(f'\n Resta: {operaciones.resta()}')
print(f'\n Producto: {operaciones.producto()}')
print(f'\n Cociente: {operaciones.division()}')
```

```
Operaciones básicas - datos de entrada: 5 y 8

Suma: 13

Resta: -3

Producto: 40

Cociente: 0.625
```

- Cálculo Área (circunferencia, cuadrado, rectángulo y triángulo equilátero)

La siguiente clase define una estructura para calcular el área de diferentes figuras geométricas

```
class Area:

def __init__(self, r, l, b, a):
self.r = r
self.l = l
self.b = b
self.a = a

def circunferencia(self, pi):
return pi*self.r**2

def cuadrado(self):
return self.l**2

def rectangulo(self):
return self.b*self.a

def triangulo_equilatero(self):
return self.b*self.a/2
```

Se llama a la clase y se define el objeto en cuestión

```
calculo_area = Area(r=4, l=5, b=6, a=3)
```

```
from math import pi

print("Cálculo del área de diferentes figuras geométricas")
print(f'\n Circunferencia de radio {calculo_area.r}: {calculo_area.circunferencia(pi)}')

print(f'\n Cuadrado de lado {calculo_area.l}: {calculo_area.cuadrado()}')

print(f'\n Rectángulo de base {calculo_area.b} y lado {calculo_area.a}:
{calculo_area.rectangulo()}')

print(f'\n Triángulo equilátero de base {calculo_area.b} y lado {calculo_area.a}:
{calculo_area.triangulo_equilatero()}')
```

```
Cálculo del área de diferentes figuras geométricas

Circunferencia de radio 4: 50.26548245743669

Cuadrado de lado 5: 25

Rectángulo de base 6 y lado 3: 18

Triángulo equilátero de base 6 y lado 3: 9.0
```

Las Clases también pueden contener *métodos privados*. Un método privado es aquel método al que no se puede acceder fuera de la clase en la que están declarados ni de ninguna otra clase base. Este tipo de métodos se declaran mediante guiones bajos dobles al principio del nombre.

- Información Personal

Skip to main content

```
class Info_persona:

def __init__(self, nombre, edad, provincia, trabajo):
self.nombre = nombre
self.edad = edad
self.provincia = provincia
self.trabajo = trabajo

def __tiempo_jubilacion(self):
return 65-self.edad

def info(self):
print(f'Nombre: {self.nombre}, Edad: {self.edad}, Provincia: {self.provincia}, Trabajo:
{self.trabajo}, Años hasta la jubilacion: {self.__tiempo_jubilacion()}')
```

```
persona_1 = Info_persona(nombre="Pablo", edad=28, provincia="Avila", trabajo="DS")
persona_1.info()
```

```
Nombre: Pablo, Edad: 28, Provincia: Avila, Trabajo: DS, Años hasta la jubilacion: 37
```

```
persona_2 = Info_persona(nombre="Luis", edad=33, provincia="Madrid", trabajo="BA")
persona_2.info()
```

```
Nombre: Luis, Edad: 33, Provincia: Madrid, Trabajo: BA, Años hasta la jubilacion: 32
```

```
persona_2.__tiempo_jubilacion()
```

```
---------------------------------------------------------------------------
AttributeError Traceback (most recent call last)
Cell In[175], line 1
----> 1 persona_2.__tiempo_jubilacion()

AttributeError: 'Info_persona' object has no attribute '__tiempo_jubilacion'
```

Estos métodos no pueden invocarse en la clase instanciada. Son métodos ocultos

6.1 Herencia

Se puede definir una clase a partir de una o más clases ya existentes. Este hecho se conoce como **Herencia**.

Es importante destacar que las clases hijas no pueden acceder a los métodos privados de la clase padre.

- Más funcionalidades a la calculadora...

Skip to main content

```
class Calculadora_compleja(Calculadora_basica):

def mayor_menor(self):
if self.a > self.b:
return self.a
elif self.a == self.b:
return self.a
else:
return self.b

def resto(self):
return self.a % self.b
```

Se define un nuevo objeto que permite realizar, además de las operaciones básicas a partir de dos números, otros funcionalidades

```
operaciones2 = Calculadora_compleja(2, 6)

print(f'Operaciones básicas - datos de entrada: {operaciones2.a} y {operaciones2.b}')
print(f'\n Suma: {operaciones2.suma()}')
print(f'\n Resta: {operaciones2.resta()}')
print(f'\n Producto: {operaciones2.producto()}')
print(f'\n Cociente: {operaciones2.division()}')
print(f'\n Resto: {operaciones2.resto()}')
print(f'\n Número más alto: {operaciones2.mayor_menor()}')
```

```
Operaciones básicas - datos de entrada: 2 y 6

Suma: 8

Resta: -4

Producto: 12

Cociente: 0.3333333333333333

Resto: 2

Número más alto: 6
```

Puede ser que, en la clase hija se necesiten nuevos parámetros. En este caso, es necesario utilizar el constructor *init*.

```
class Sumar_numeros_divisibles(Calculadora_basica):

def __init__(self, a, b, c):
Calculadora_basica.__init__(self, a, b)
self.c = c

def suma_divisibles(self):
suma= 0
for i in range(self.a, self.b):
if not i%self.c==0:
suma = suma + i
return suma

def mayor_menor(self):
if self.a>self.b:
return self.a

elif self.a==self.b:
return self.a
else:
return self.b

def resto(self):
return self.a%self.b
```

```
operaciones3 = Sumar_numeros_divisibles(1, 11, 2)

print(f'Operaciones básicas - datos de entrada: {operaciones3.a} y {operaciones3.b}')
print(f'\n Suma: {operaciones3.suma()}')
print(f'\n Resta: {operaciones3.resta()}')
print(f'\n Producto: {operaciones3.producto()}')
print(f'\n Cociente: {operaciones3.division()}')
print(f'\n Resto: {operaciones3.resto()}')
print(f'\n Número más alto: {operaciones3.mayor_menor()}')
print(f'\n Suma intervalo no divisibles por {operaciones3.c}:
{operaciones3.suma_divisibles()}')
```

```
Operaciones básicas - datos de entrada: 1 y 11

Suma: 12

Resta: -10

Producto: 11

Cociente: 0.09090909090909091

Resto: 1

Número más alto: 11

Suma intervalo no divisibles por 2: 25
```

6.2 Tipos de Métodos (instancia, clase y estático)

Skip to main content

Ejemplo de método de clase

- Área rectángulo y cuadrado:

```
class Rectangulo_area:

def __init__(self, ancho, largo):
self.ancho = ancho
self.largo = largo

def area(self):
return self.ancho*self.largo

@classmethod
def cuadrado(cls, lado):
return cls(lado, lado)
```

Se define un objeto “rectángulo” y se calcula su área

```
rectangulo = Rectangulo_area(ancho=2, largo=3)
print(f'Área del rectángulo de base {rectangulo.ancho} y altura {rectangulo.largo}:
{rectangulo.area()}')
```

```
Área del rectángulo de base 2 y altura 3: 6
```

Uso del método de clase para calcular un objeto que obtenga el área de un cuadrado (ancho=largo)

```
cuadrado = Rectangulo_area.cuadrado(2)
print(f'Área del cuadrado de lado {cuadrado.ancho}: {cuadrado.area()}')
```

```
Área del cuadrado de lado 2: 4
```

Ejemplo de método de clase y estático

- Menú del día

Skip to main content

```
class Alimento:

def __init__(self, tipo):
self.tipo = tipo

@staticmethod
def caliente_frio(temperatura):

if temperatura > 30:
print("\n Se sirve: caliente")
elif temperatura > 10:
print("\n Se sirve: normal")
elif temperatura > 0:
print("\n Se sirve: frío")
else:
print("\n Se sirve: helado")

@classmethod
def add_alimento(cls, tipo):
return cls(tipo)

def nombre(self):
return self.tipo
```

Creamos un objeto para cada pantalla y presentamos el menú para hoy

```
sopa = Alimento("Sopa de marisco")
filete = Alimento.add_alimento("Filete de ternera")
helado = Alimento.add_alimento("helado de chocolate")
agua = Alimento.add_alimento("agua")
cafe = Alimento.add_alimento("café con leche")
chupito = Alimento.add_alimento("chupito de hierbas")


print("### MENÚ DEL DÍA ###")

print(f'\n Primeros: {sopa.nombre()}') # indicar nombre
sopa.caliente_frio(36) # indicar bucket temperatura

print(f'\n Segundos: {filete.nombre()}') # indicar nombre
filete.caliente_frio(21) # indicar bucket temperatura

print(f'\n Postre: {helado.nombre()}') # indicar nombre
helado.caliente_frio(-5) # checkear temperatura

print(f'\n Bebida: {agua.nombre()}') # indicar nombre
agua.caliente_frio(32) # checkear temperatura

print(f'\n Sobremesa (I): {cafe.nombre()}') # indicar nombre
cafe.caliente_frio(16) # checkear temperatura

print(f'\n Sobremesa (II): {chupito.nombre()}') # indicar nombre
chupito.caliente_frio(7) # checkear temperatura
```

```
### MENÚ DEL DÍA ###

Primeros: Sopa de marisco

Se sirve: caliente

Segundos: Filete de ternera

Se sirve: normal

Postre: helado de chocolate

Se sirve: helado

Bebida: agua

Se sirve: caliente

Sobremesa (I): café con leche

Se sirve: normal

Sobremesa (II): chupito de hierbas

Se sirve: frío
```

Análisis de Datos

Antes de comenzar, se importa el paquete *os* y se define el directorio donde se encuentra el notebook.

```
import os
path = 'C:/Users/p_san/OneDrive/Escritorio/master_uned_modulo_1/Introducción a Python' #
cambiar la ruta donde se encuentran los ficheros en el PC
```

1. Lectura y escritura de ficheros

```
import csv
```

1.1 Ficheros csv

Lectura: sentencia *reader*

- Lectura de cabeceras

Skip to main content

```
with open(os.path.join(path, 'info_personas.csv')) as csv_file:
csv_reader = csv.reader(csv_file, delimiter=',')

contar_lineas = 0
for row in csv_reader:
if contar_lineas == 0:
print(f'Los campos del fichero son {row}')
contar_lineas = contar_lineas + 1
```

```
Los campos del fichero son ['Nombre;Provincia;Nivel_Estudios;Edad']
```

- Lectura de los registros

```
with open(os.path.join(path, 'info_personas.csv')) as csv_file:
csv_reader = csv.reader(csv_file, delimiter=',')

contar_lineas= 0
next(csv_reader, None) # omitir encabezado
for row in csv_reader:
print(row)
contar_lineas = contar_lineas +1
print(f'Total de líneas {contar_lineas} lines.')
```

```
['Pablo;Avila;Grado;28']
['Luis;Madrid;Bachiller;45']
['Fran;Toledo;Licenciatura;34']
['Maria;Barcelona;Doctor;56']
['Bea;Malaga;Basico;22']
['Emilio;Salamanca;Bachiller;37']
['Juan;Caceres;Grado;21']
['Laura;Vitoria;Licenciatura;40']
Total de líneas 8 lines.
```

Lectura: sentencia *DictReader*

Importación del fichero en formato diccionario

```
results= []

with open(os.path.join(path, 'info_personas.csv')) as csv_file:
csv_reader = csv.DictReader(csv_file, delimiter=',')

for row in csv_reader:
results.append(row)

print(results)
```

```
[{'Nombre;Provincia;Nivel_Estudios;Edad': 'Pablo;Avila;Grado;28'},
{'Nombre;Provincia;Nivel_Estudios;Edad': 'Luis;Madrid;Bachiller;45'},
{'Nombre;Provincia;Nivel_Estudios;Edad': 'Fran;Toledo;Licenciatura;34'},
{'Nombre;Provincia;Nivel_Estudios;Edad': 'Maria;Barcelona;Doctor;56'},
{'Nombre;Provincia;Nivel_Estudios;Edad': 'Bea;Malaga;Basico;22'},
{'Nombre;Provincia;Nivel_Estudios;Edad': 'Emilio;Salamanca;Bachiller;37'},
{'Nombre;Provincia;Nivel_Estudios;Edad': 'Juan;Caceres;Grado;21'},
{'Nombre;Provincia;Nivel_Estudios;Edad': 'Laura;Vitoria;Licenciatura;40'}]
```

```
for i in results:
for j in i.values():
print(j)
```

```
Pablo;Avila;Grado;28
Luis;Madrid;Bachiller;45
Fran;Toledo;Licenciatura;34
Maria;Barcelona;Doctor;56
Bea;Malaga;Basico;22
Emilio;Salamanca;Bachiller;37
Juan;Caceres;Grado;21
Laura;Vitoria;Licenciatura;40
```

Escritura: sentencia *writer*

```
datos1 = [["Nombre", "Provincia", "Nivel_Estudios", "Edad"],
["Maca", "Valencia", "Licenciatura", 37],
["Roberto", "Pontevedra", "Basico", 19]]
```

```
with open(os.path.join(path, 'info_personas2a.csv'), 'w', newline='') as file:
writer = csv.writer(file, delimiter=';')
writer.writerows(datos1)
```

Escritura: sentencia *Dictwriter*

```
datos2 = [{'Nombre': 'Maca', 'Provincia': 'Valencia', 'Nivel_Estudios': 'Licenciatura',
'Edad':37},
{'Nombre': 'Roberto', 'Provincia': 'Pontevedra', 'Nivel_Estudios': 'Basico', 'Edad':19}]

cabecera = ['Nombre', 'Provincia', 'Nivel_Estudios', 'Edad']
```

```
with open(os.path.join(path, 'info_personas2b.csv'), 'w', newline='') as file:
writer = csv.DictWriter(file, delimiter=';', fieldnames= cabecera)
writer.writeheader()
```

Skip to main content

1.2 Ficheros json

```
import json
```

Lectura del fichero

```
with open(os.path.join(path, 'config.json')) as file:
data = json.load(file)
```

- Información del fichero

```
data
```

```
{'umbrales': {'riesgo_bajo': 100,
'riesgo_medio_bajo': 150,
'riesgo_medio_alto': 190},
'fecha_model': {'time_init': '2015-01-01',
'time_end': '2021-09-30',
'time_use': 'today'},
'parametros_model': {'semilla': 123,
'split': 0.3,
'n_estimators': 50,
'max_depth': 6,
'min_samples_split': 4}}
```

El fichero *json* es leído como un *diccionario* de forma que se puede iterar sobre él.

- Manipulación del fichero

```
data.keys()
```

```
dict_keys(['umbrales', 'fecha_model', 'parametros_model'])
```

```
data.values()
```

```
dict_values([{'riesgo_bajo': 100, 'riesgo_medio_bajo': 150, 'riesgo_medio_alto': 190},
{'time_init': '2015-01-01', 'time_end': '2021-09-30', 'time_use': 'today'}, {'semilla': 123,
'split': 0.3, 'n_estimators': 50, 'max_depth': 6, 'min_samples_split': 4}])
```

Skip to main content

```
for i in data['umbrales'].items():
print(i)
```

```
('riesgo_bajo', 100)
('riesgo_medio_bajo', 150)
('riesgo_medio_alto', 190)
```

Escritura del fichero

De igual forma, un *diccionario* se puede guardar como un fichero *json*

```
datos_personales = {'Nombre':'Pablo', 'Apellido':'Sanchez', 'Residencia':'Madrid'}
```

```
with open(os.path.join(path,'datos_personales.json'), 'w') as file:
json.dump(datos_personales, file)
```

1.3 Ficheros yaml

```
import yaml
```

Lectura del fichero

```
with open(os.path.join(path, 'config_data_model.yaml')) as file:
data = yaml.load(file, Loader=yaml.FullLoader)
```

- Información del fichero

```
data
```

Skip to main content

```
{'data': {'target': 'class',
'features': ['checking_status',
'credit_history',
'personal_status',
'existing_credits',
'age',
'credit_amount']},
'model': {'type': 'glm', 'family': 'binomial'},
'decision': {'threshold': 0.35,
'new_data': {'features': ['checking_status',
'credit_history',
'personal_status',
'existing_credits',
'age',
'credit_amount'],
'values': ['<0', "'existing paid'", "'male mar/wid'", 1, 45, 3893]}}}
```

El fichero *yaml* es leído como un *diccionario* de forma que se puede iterar sobre él.

- Manipulación del fichero

```
data.keys()
```

```
dict_keys(['data', 'model', 'decision'])
```

```
data.values()
```

```
dict_values([{'target': 'class', 'features': ['checking_status', 'credit_history',
'personal_status', 'existing_credits', 'age', 'credit_amount']}, {'type': 'glm', 'family':
'binomial'}, {'threshold': 0.35, 'new_data': {'features': ['checking_status', 'credit_history',
'personal_status', 'existing_credits', 'age', 'credit_amount'], 'values': ['<0', "'existing
paid'", "'male mar/wid'", 1, 45, 3893]}}])
```

```
data["model"]
```

```
{'type': 'glm', 'family': 'binomial'}
```

```
data["decision"]
```

```
{'threshold': 0.35,
'new_data': {'features': ['checking_status',
'credit_history',
'personal_status',
'existing_credits',
'age',
'credit_amount'],
'values': ['<0', "'existing paid'", "'male mar/wid'", 1, 45, 3893]}}
```

```
for i in data['data'].items():
print(i)
```

```
('target', 'class')
('features', ['checking_status', 'credit_history', 'personal_status', 'existing_credits', 'age',
'credit_amount'])
```

Escritura del fichero

De igual forma, un *diccionario* se puede guardar como un fichero *yaml*

```
datos_personales = {'Nombre':'Pablo', 'Apellido':'Sanchez', 'Residencia':'Madrid'}
```

```
with open(os.path.join(path,'datos_personales.yaml'), 'w') as file:
yaml.dump(datos_personales, file)
```

2. Numpy

Numpy es una librería de fácil manejo y eficiente para el cálculo numérico y matricial.

```
import numpy as np

from numpy import linalg
```

```
print(f'Numpy - {np.__version__}')
```

```
Numpy - 1.24.3
```

Skip to main content

2.1 Vectores

- Definición y operaciones básicas

```
a = np.array([1,2,-2,4,0,6,7,-3,9])
```

```
type(a)
```

```
numpy.ndarray
```

```
a.shape # tamaño del array
```

```
(9,)
```

```
print(f'El elemento más grande del vector es: {np.max(a)}')
```

```
El elemento más grande del vector es: 9
```

```
print(f'El elemento más pequeño del vector es: {np.min(a)}')
```

```
El elemento más pequeño del vector es: -3
```

```
print(f'Suma de los elementos del vector: {np.sum(a)}')
```

```
Suma de los elementos del vector: 24
```

```
print(f'El vector A tiene media {np.mean(a)} y desviación estandar {np.std(a)}')
```

```
El vector A tiene media 2.6666666666666665 y desviación estandar 3.8873012632302
```

- Vector de *unos* y de *ceros*

Skip to main content

```
np.ones(9)
```

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

```
np.zeros(9)
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

- Uso de las funciones **arange** y **linespace**

```
b = np.arange(start=1, stop=10, step=1)
b
```

```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
b.shape
```

```
(9,)
```

```
c = np.linspace(start= 0, stop= 5, num= 10)
c
```

```
array([0. , 0.55555556, 1.11111111, 1.66666667, 2.22222222,
2.77777778, 3.33333333, 3.88888889, 4.44444444, 5. ])
```

```
c.shape
```

```
(10,)
```

```
np.round(c, 0) # redondear un vector
```

```
array([0., 1., 1., 2., 2., 3., 3., 4., 4., 5.])
```

Skip to main content

- Operaciones con vectores

```
a[3]<2
```

```
False
```

```
3*a
```

```
array([ 3, 6, -6, 12, 0, 18, 21, -9, 27])
```

```
3+a
```

```
array([ 4, 5, 1, 7, 3, 9, 10, 0, 12])
```

```
a+b
```

```
array([ 2, 4, 1, 8, 5, 12, 14, 5, 18])
```

```
a*b
```

```
array([ 1, 4, -6, 16, 0, 36, 49, -24, 81])
```

```
a/b
```

```
array([ 1. , 1. , -0.66666667, 1. , 0. ,
1. , 1. , -0.375 , 1. ])
```

2.2 Matrices

```
matriz = np.array([[1,2], [5,6]], dtype=float)
```

Skip to main content

```
matriz
```

```
array([[1., 2.],
[5., 6.]])
```

```
type(matriz)
```

```
numpy.ndarray
```

```
matriz.shape
```

```
(2, 2)
```

```
matriz.T # matriz transpuesta
```

```
array([[1., 5.],
[2., 6.]])
```

```
np.dot(matriz.T, matriz) # producto matricial
```

```
array([[26., 32.],
[32., 40.]])
```

```
linalg.det(matriz) # determinante de la matriz

# np.linalg.det(matriz)
```

```
-3.999999999999999
```

```
linalg.eigvals(matriz) # valores propios de la matriz

# np.linalg.eigvals(matriz)
```

Skip to main content

```
array([-0.53112887, 7.53112887])
```

2.3 Otras acciones

- Concatenar vectores y matrices

```
v1 = np.array([1.4, 2.3, 3.3, 6.9, 11.0])

v1
```

```
array([ 1.4, 2.3, 3.3, 6.9, 11. ])
```

```
v2 = np.array([4.1, 3.2, 3.3, 9.6, 0.11])

v2
```

```
array([4.1 , 3.2 , 3.3 , 9.6 , 0.11])
```

```
np.concatenate((v1, v2), axis=0)
```

```
array([ 1.4 , 2.3 , 3.3 , 6.9 , 11. , 4.1 , 3.2 , 3.3 , 9.6 ,
0.11])
```

```
m1 = np.array([[5,4,0], [6,2,9]], dtype=float)

m1
```

```
array([[5., 4., 0.],
[6., 2., 9.]])
```

```
m2 = np.array([[2,1], [0, 3], [4,1]])

m2
```

```
array([[2, 1],
[0, 3],
[4, 1]])
```

Skip to main content

```
np.concatenate((m1, m2.T), axis=0)
```

```
array([[5., 4., 0.],
[6., 2., 9.],
[2., 0., 4.],
[1., 3., 1.]])
```

```
np.concatenate((m1, m2.T), axis=1)
```

```
array([[5., 4., 0., 2., 0., 4.],
[6., 2., 9., 1., 3., 1.]])
```

- Redefinir el tamaño de un vector

```
a
```

```
array([ 1, 2, -2, 4, 0, 6, 7, -3, 9])
```

```
a.shape
```

```
(9,)
```

```
a1 = np.reshape(a, (-1,1)) # de gran utilidad para trabajar con data.frames
```

```
a1
```

```
array([[ 1],
[ 2],
[-2],
[ 4],
[ 0],
[ 6],
[ 7],
[-3],
[ 9]])
```

```
a1.shape
```

Skip to main content

```
(9, 1)
```

```
a2 = np.reshape(a, (1,-1)) # no tiene mucho uso
```

```
a2
```

```
array([[ 1, 2, -2, 4, 0, 6, 7, -3, 9]])
```

```
a2.shape
```

```
(1, 9)
```

3. Pandas

Pandas es una librería para análisis y exploración de datos en formato tabla (filas, columnas). Este paquete está construido bajo la librería Numpy

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

```
print(f'Pandas - {pd.__version__}')
print(f'Numpy - {np.__version__}')
```

```
Pandas - 1.5.3
Numpy - 1.24.3
```

3.1 DataFrames

```
diccionario = {'NOMBRE': ['Pablo', 'Luis', 'Hector', 'Maria', 'Veronica'],
'EDAD': [26,30,34,43,27],
'CIUDAD': ['Avila', 'Madrid', 'Santander', 'Cordoba', 'Madrid'],
'EDUCACION': ['Universitaria', 'Elemental', 'Universitaria', 'Bachillerato', 'Universitaria']
}

df=pd.DataFrame(diccionario)
df
```

	NOMBRE	EDAD	CIUDAD	EDUCACION
0	Pablo	26	Avila	Universitaria
1	Luis	30	Madrid	Elemental
2	Hector	34	Santander	Universitaria
3	Maria	43	Cordoba	Bachillerato
4	Veronica	27	Madrid	Universitaria

```
matriz = np.array([('Pablo','Luis', 'Hector', 'Maria', 'Veronica'),
(26, 30, 34, 43, 27),
('Avila', 'Madrid', 'Santander', 'Cordoba', 'Madrid'),
('Universitaria', 'Elemental', 'Universitaria', 'Bachillerato', 'Universitaria')])

pd.DataFrame(matriz.T,
columns=['NOMBRE', 'EDAD', 'CIUDAD', 'EDUCACION'],
index=np.arange(start=1, stop=len(matriz[0])+1))
```

	NOMBRE	EDAD	CIUDAD	EDUCACION
1	Pablo	26	Avila	Universitaria
2	Luis	30	Madrid	Elemental
3	Hector	34	Santander	Universitaria
4	Maria	43	Cordoba	Bachillerato
5	Veronica	27	Madrid	Universitaria

```
df.shape
```

```
(5, 4)
```

```
print(f'Filas: {df.shape[0]}')
print(f'Columnas: {df.shape[1]}')
```

```
Filas: 5
Columnas: 4
```

Skip to main content

```
df.columns
```

```
Index(['NOMBRE', 'EDAD', 'CIUDAD', 'EDUCACION'], dtype='object')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 4 columns):
# Column Non-Null Count Dtype
--- ------ -------------- -----
0 NOMBRE 5 non-null object
1 EDAD 5 non-null int64
2 CIUDAD 5 non-null object
3 EDUCACION 5 non-null object
dtypes: int64(1), object(3)
memory usage: 292.0+ bytes
```

```
df.describe()
```

	EDAD
count	5.000000
mean	32.000000
std	6.892024
min	26.000000
25%	27.000000
50%	30.000000
75%	34.000000
max	43.000000

3.2 Importar y Exportar ficheros

Un fichero puede ser guardado en distintos formatos (Excel, CSV, Text, HDFS, json, Pickle, SQL, SAS, SPSS,...). Se presenta como ejemplo la importación de un fichero en formato CSV; si bien, *Pandas* permite la importación desde diferentes formatos a partir de una sintaxis análoga:

https://pandas.pydata.org/pandas-docs/stable/reference/io.html

```
iris = pd.read_csv("iris.csv") # tener en cuenta la ruta del fichero
```

```
type(iris)
```

```
pandas.core.frame.DataFrame
```

```
iris.head() # cinco primeros elementos
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
iris.tail() # cinco últimos elementos
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

```
iris.shape
```

```
(150, 6)
```

```
iris.columns
```

```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
'Species'],
dtype='object')
```

```
iris.dtypes
```

Skip to main content

```
Id int64
SepalLengthCm float64
SepalWidthCm float64
PetalLengthCm float64
PetalWidthCm float64
Species object
dtype: object
```

```
iris['Id'] = iris['Id'].astype(str)

iris['Species'] = iris['Species'].astype('category')
```

```
iris.dtypes
```

```
Id object
SepalLengthCm float64
SepalWidthCm float64
PetalLengthCm float64
PetalWidthCm float64
Species category
dtype: object
```

```
iris.describe()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

3.3 Operaciones Básicas sobre DataFrames

- Categorías de la variable

```
iris['Species'].unique()
```

Skip to main content

```
['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
Categories (3, object): ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
```

```
iris['Species'].value_counts()

#iris['Species'].value_counts(normalize=True)
```

```
Iris-setosa 50
Iris-versicolor 50
Iris-virginica 50
Name: Species, dtype: int64
```

- Seleccionar filas y columnas

```
iris[iris['SepalLengthCm'] >= 6]
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
50	51	7.0	3.2	4.7	1.4	Iris-versicolor
51	52	6.4	3.2	4.5	1.5	Iris-versicolor
52	53	6.9	3.1	4.9	1.5	Iris-versicolor
54	55	6.5	2.8	4.6	1.5	Iris-versicolor
56	57	6.3	3.3	4.7	1.6	Iris-versicolor
...	...	...	...	...	...	...
144	145	6.7	3.3	5.7	2.5	Iris-virginica
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica

67 rows × 6 columns

```
iris[(iris['SepalLengthCm'] >= 6.0) & (iris['PetalWidthCm'] < 2.5)]
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
50	51	7.0	3.2	4.7	1.4	Iris-versicolor
51	52	6.4	3.2	4.5	1.5	Iris-versicolor
52	53	6.9	3.1	4.9	1.5	Iris-versicolor
54	55	6.5	2.8	4.6	1.5	Iris-versicolor
56	57	6.3	3.3	4.7	1.6	Iris-versicolor
...	...	...	...	...	...	...
143	144	6.8	3.2	5.9	2.3	Iris-virginica
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica

64 rows × 6 columns

```
iris[(iris['SepalLengthCm'] < 4.5) | (iris['PetalWidthCm'] < 0.2)]
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa
12	13	4.8	3.0	1.4	0.1	Iris-setosa
13	14	4.3	3.0	1.1	0.1	Iris-setosa
32	33	5.2	4.1	1.5	0.1	Iris-setosa
34	35	4.9	3.1	1.5	0.1	Iris-setosa
37	38	4.9	3.1	1.5	0.1	Iris-setosa
38	39	4.4	3.0	1.3	0.2	Iris-setosa
42	43	4.4	3.2	1.3	0.2	Iris-setosa

```
iris.loc[25]
```

```
Id 26
SepalLengthCm 5.0
SepalWidthCm 3.0
PetalLengthCm 1.6
PetalWidthCm 0.2
Species Iris-setosa
Name: 25, dtype: object
```

Skip to main content

```
iris[iris['Species'] == 'Iris-setosa'][0:10]
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

```
iris[['Species', 'Id']]
```

	Species	Id
0	Iris-setosa	1
1	Iris-setosa	2
2	Iris-setosa	3
3	Iris-setosa	4
4	Iris-setosa	5
...	...	...
145	Iris-virginica	146
146	Iris-virginica	147
147	Iris-virginica	148
148	Iris-virginica	149
149	Iris-virginica	150

150 rows × 2 columns

```
variables = ['Id', 'SepalLengthCm', 'PetalLengthCm', 'SepalWidthCm', 'PetalWidthCm']
iris.loc[(iris['Species'] == 'Iris-setosa') & (iris['SepalWidthCm'] > 4), variables]
```

Skip to main content

	Id	SepalLengthCm	PetalLengthCm	SepalWidthCm	PetalWidthCm
15	16	5.7	1.5	4.4	0.4
32	33	5.2	1.5	4.1	0.1
33	34	5.5	1.4	4.2	0.2

- Sumarización y agrupaciones

```
np.max(iris['SepalLengthCm'])
```

```
7.9
```

```
np.argmax(iris['SepalLengthCm'])
```

```
131
```

```
iris.groupby('Species').size()
```

```
Species
Iris-setosa 50
Iris-versicolor 50
Iris-virginica 50
dtype: int64
```

```
iris.groupby('Species', as_index=False).agg({"SepalLengthCm":"mean"})
```

	Species	SepalLengthCm
0	Iris-setosa	5.006
1	Iris-versicolor	5.936
2	Iris-virginica	6.588

```
iris.groupby('Species', as_index=False)[['SepalLengthCm', 'SepalWidthCm']].max()
```

	Species	SepalLengthCm	SepalWidthCm
0	Iris-setosa	5.8	4.4
1	Iris-versicolor	7.0	3.4

Skip to main content

```
iris.groupby('Species', as_index=False)[['SepalLengthCm',
'SepalWidthCm']].agg({'SepalLengthCm':['min', 'max', 'mean', 'median'],
'SepalWidthCm':['min', 'max', 'mean', 'median']})
```

	Species	SepalLengthCm				SepalWidthCm			
		min	max	mean	median	min	max	mean	median
0	Iris-setosa	4.3	5.8	5.006	5.0	2.3	4.4	3.418	3.4
1	Iris-versicolor	4.9	7.0	5.936	5.9	2.0	3.4	2.770	2.8
2	Iris-virginica	4.9	7.9	6.588	6.5	2.2	3.8	2.974	3.0

- Ordenación y renombrar variables

```
iris.sort_values(by=['SepalLengthCm', 'PetalLengthCm'], ascending=False)[0:10] # ordenación de
mayor a menor
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
131	132	7.9	3.8	6.4	2.0	Iris-virginica
118	119	7.7	2.6	6.9	2.3	Iris-virginica
117	118	7.7	3.8	6.7	2.2	Iris-virginica
122	123	7.7	2.8	6.7	2.0	Iris-virginica
135	136	7.7	3.0	6.1	2.3	Iris-virginica
105	106	7.6	3.0	6.6	2.1	Iris-virginica
130	131	7.4	2.8	6.1	1.9	Iris-virginica
107	108	7.3	2.9	6.3	1.8	Iris-virginica
109	110	7.2	3.6	6.1	2.5	Iris-virginica
125	126	7.2	3.2	6.0	1.8	Iris-virginica

```
iris.sort_values(by='SepalLengthCm', ascending=False)[0:10] # ordenación de mayor a menor
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
131	132	7.9	3.8	6.4	2.0	Iris-virginica
135	136	7.7	3.0	6.1	2.3	Iris-virginica
122	123	7.7	2.8	6.7	2.0	Iris-virginica
117	118	7.7	3.8	6.7	2.2	Iris-virginica
118	119	7.7	2.6	6.9	2.3	Iris-virginica
105	106	7.6	3.0	6.6	2.1	Iris-virginica
130	131	7.4	2.8	6.1	1.9	Iris-virginica
107	108	7.3	2.9	6.3	1.8	Iris-virginica
125	126	7.2	3.2	6.0	1.8	Iris-virginica
109	110	7.2	3.6	6.1	2.5	Iris-virginica

```
iris_rename = iris.rename(columns={'SepalLengthCm': 'Longitud_Sepalo',
'SepalWidthCm': 'Ancho_Sepalo',
'PetalLengthCm': 'Longitud_Petalo',
'PetalWidthCm': 'Ancho_Petalo',
'Species': 'Especies'})
```

```
iris_rename.tail(10)
```

	Id	Longitud_Sepalo	Ancho_Sepalo	Longitud_Petalo	Ancho_Petalo	Especies
140	141	6.7	3.1	5.6	2.4	Iris-virginica
141	142	6.9	3.1	5.1	2.3	Iris-virginica
142	143	5.8	2.7	5.1	1.9	Iris-virginica
143	144	6.8	3.2	5.9	2.3	Iris-virginica
144	145	6.7	3.3	5.7	2.5	Iris-virginica
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

- Manejo de las columnas

a) separación de columnas

Skip to main content

```
iris_rename[['Nombre_Planta', 'Nombre_Especie']] = iris_rename['Especies'].str.split('-
',expand=True)

iris_rename.head()
```

	Id	Longitud_Sepalo	Ancho_Sepalo	Longitud_Petalo	Ancho_Petalo	Especies	Nombre_Planta	Nom
0	1	5.1	3.5	1.4	0.2	Iris-setosa	Iris	
1	2	4.9	3.0	1.4	0.2	Iris-setosa	Iris	
2	3	4.7	3.2	1.3	0.2	Iris-setosa	Iris	
3	4	4.6	3.1	1.5	0.2	Iris-setosa	Iris	
4	5	5.0	3.6	1.4	0.2	Iris-setosa	Iris	

b) nuevas variables como condición de otras

```
iris_rename['Var1'] = np.where(iris_rename['Especies'].str.contains('osa'), 1, 0)
```

```
condition = [iris_rename['Longitud_Sepalo']<5.0,
(iris_rename['Ancho_Petalo']>1.0) | (iris_rename['Nombre_Especie']=="virginica")
]

value = ['A', 'B']

iris_rename['Var2'] = np.select(condition, value, default='C')
```

c) asignar nuevas variables al dataframe

```
iris_rename = iris_rename.assign(Cociente_Sepalo = iris_rename['Longitud_Sepalo'] /
iris_rename['Ancho_Sepalo'])
```

```
iris_rename.tail()
```

Skip to main content

	Id	Longitud_Sepalo	Ancho_Sepalo	Longitud_Petalo	Ancho_Petalo	Especies	Nombre_Planta	N
145	146	6.7	3.0	5.2	2.3	Iris-virginica	Iris	
146	147	6.3	2.5	5.0	1.9	Iris-virginica	Iris	
147	148	6.5	3.0	5.2	2.0	Iris-virginica	Iris	
148	149	6.2	3.4	5.4	2.3	Iris-virginica	Iris	
149	150	5.9	3.0	5.1	1.8	Iris-virginica	Iris	

3.4 Otras operaciones

- El método **apply**: este método se utiliza para aplicar una función a una fila o columa del data.frame

```
c = "pablo"
```

```
c.upper()
```

```
'PABLO'
```

Ejemplo 1

```
def Mayusculas(x):
return x.upper()
```

```
iris_rename['Nombre_Especie_MAY'] = iris_rename['Nombre_Especie'].apply(Mayusculas)
```

```
iris_rename[['Nombre_Especie_MAY', 'Nombre_Especie']].head()
```

	Nombre_Especie_MAY	Nombre_Especie
0	SETOSA	setosa
1	SETOSA	setosa
2	SETOSA	setosa
3	SETOSA	setosa
4	SETOSA	setosa

Ejemplo 2

```
def suma_pond (x,y):
return np.average(x,weights=y)
```

```
iris.groupby('Species').apply(lambda x: suma_pond(x['SepalLengthCm'], x['PetalWidthCm']))
```

```
Species
Iris-setosa 5.048361
Iris-versicolor 5.977225
Iris-virginica 6.611747
dtype: float64
```

Ejemplo 3

```
def Cociente(x,y):
return x / y
```

```
iris_rename['Cociente_Sepalo_2'] = iris_rename.apply(lambda x:
Cociente(x.Longitud_Sepalo,x.Ancho_Sepalo),axis=1)
```

```
iris_rename[['Longitud_Sepalo', 'Ancho_Sepalo', 'Cociente_Sepalo', 'Cociente_Sepalo_2']].head()
```

	Longitud_Sepalo	Ancho_Sepalo	Cociente_Sepalo	Cociente_Sepalo_2
0	5.1	3.5	1.457143	1.457143
1	4.9	3.0	1.633333	1.633333
2	4.7	3.2	1.468750	1.468750
3	4.6	3.1	1.483871	1.483871
4	5.0	3.6	1.388889	1.388889

- Otras cosas de interés

Skip to main content

```
pd.crosstab(iris_rename['Var1'], iris_rename['Nombre_Especie']) # tabla de frecuencias
```

Nombre_Especie	setosa	versicolor	virginica
Var1			
0	0	50	50
1	50	0	0

```
pd.crosstab(iris_rename['Var2'], iris_rename['Nombre_Especie']) # tabla de frecuencias
```

Nombre_Especie	setosa	versicolor	virginica
Var2			
A	20	1	1
B	0	43	49
C	30	6	0

Pivot-table

```
pivot_table = pd.pivot_table(iris_rename,
values='Longitud_Sepalo',
columns='Var2',
index=['Nombre_Especie', 'Var1'],
aggfunc=np.mean)

pivot_table = pivot_table.fillna(0)
```

```
pivot_table
```

	Var2	A	B	C
Nombre_Especie	Var1			
setosa	**1**	4.67	0.000000	5.23
versicolor	**0**	4.90	6.020930	5.50
virginica	**0**	4.90	6.622449	0.00

3.5 Tratamiento de ficheros

- Lectura de ficheros en formato *csv*

Skip to main content

```
personas = pd.read_csv('personas.csv', sep=';')
paises = pd.read_csv('paises.csv', sep=';')
```

Skip to main content

```
---------------------------------------------------------------------------
FileNotFoundError Traceback (most recent call last)
Cell In[131], line 1
----> 1 personas = pd.read_csv('personas.csv', sep=';')
2 paises = pd.read_csv('paises.csv', sep=';')

File ~\anaconda3\Lib\site-packages\pandas\util\_decorators.py:211, in deprecate_kwarg.
<locals>._deprecate_kwarg.<locals>.wrapper(*args, **kwargs)
209 else:
210 kwargs[new_arg_name] = new_arg_value
--> 211 return func(*args, **kwargs)

File ~\anaconda3\Lib\site-packages\pandas\util\_decorators.py:331, in
deprecate_nonkeyword_arguments.<locals>.decorate.<locals>.wrapper(*args, **kwargs)
325 if len(args) > num_allow_args:
326 warnings.warn(
327 msg.format(arguments=_format_argument_list(allow_args)),
328 FutureWarning,
329 stacklevel=find_stack_level(),
330 )
--> 331 return func(*args, **kwargs)

File ~\anaconda3\Lib\site-packages\pandas\io\parsers\readers.py:950, in
read_csv(filepath_or_buffer, sep, delimiter, header, names, index_col, usecols, squeeze, prefix,
mangle_dupe_cols, dtype, engine, converters, true_values, false_values, skipinitialspace,
skiprows, skipfooter, nrows, na_values, keep_default_na, na_filter, verbose, skip_blank_lines,
parse_dates, infer_datetime_format, keep_date_col, date_parser, dayfirst, cache_dates, iterator,
chunksize, compression, thousands, decimal, lineterminator, quotechar, quoting, doublequote,
escapechar, comment, encoding, encoding_errors, dialect, error_bad_lines, warn_bad_lines,
on_bad_lines, delim_whitespace, low_memory, memory_map, float_precision, storage_options)
935 kwds_defaults = _refine_defaults_read(
936 dialect,
937 delimiter,
(...)
946 defaults={"delimiter": ","},
947 )
948 kwds.update(kwds_defaults)
--> 950 return _read(filepath_or_buffer, kwds)

File ~\anaconda3\Lib\site-packages\pandas\io\parsers\readers.py:605, in
_read(filepath_or_buffer, kwds)
602 _validate_names(kwds.get("names", None))
604 # Create the parser.
--> 605 parser = TextFileReader(filepath_or_buffer, **kwds)
607 if chunksize or iterator:
608 return parser

File ~\anaconda3\Lib\site-packages\pandas\io\parsers\readers.py:1442, in
TextFileReader.__init__(self, f, engine, **kwds)
1439 self.options["has_index_names"] = kwds["has_index_names"]
1441 self.handles: IOHandles | None = None
-> 1442 self._engine = self._make_engine(f, self.engine)

File ~\anaconda3\Lib\site-packages\pandas\io\parsers\readers.py:1735, in
TextFileReader._make_engine(self, f, engine)
1733 if "b" not in mode:
1734 mode += "b"
-> 1735 self.handles = get_handle(
1736 f,
1737 mode,
1738 encoding=self.options.get("encoding", None),
1739 compression=self.options.get("compression", None),
1740 memory_map=self.options.get("memory_map", False),
1741 is_text=is_text,
1742 errors=self.options.get("encoding_errors", "strict"),
1743 storage_options=self.options.get("storage_options", None),
1744 )
```

Skip to main content

```
File ~\anaconda3\Lib\site-packages\pandas\io\common.py:856, in get_handle(path_or_buf, mode,
encoding, compression, memory_map, is_text, errors, storage_options)
851 elif isinstance(handle, str):
852 # Check whether the filename is to be opened in binary mode.
853 # Binary mode does not support 'encoding' and 'newline'.
854 if ioargs.encoding and "b" not in ioargs.mode:
855 # Encoding
--> 856 handle = open(
857 handle,
858 ioargs.mode,
859 encoding=ioargs.encoding,
860 errors=errors,
861 newline="",
862 )
863 else:
864 # Binary mode
865 handle = open(handle, ioargs.mode)

FileNotFoundError: [Errno 2] No such file or directory: 'personas.csv'
```

```
print(f'Tablón Personas - filas {personas.shape[0]} y columnas {personas.shape[1]}')
print(f'Tablón Paises - filas {paises.shape[0]} y columnas {paises.shape[1]}')
```

```
Tablón Personas - filas 6 y columnas 3
Tablón Paises - filas 4 y columnas 2
```

```
personas
```

	ID	Nombre	Nacionalidad
0	1	Laura	ES
1	2	John	US
2	3	Esteban	ES
3	4	Catherine	UK
4	5	Ilayda	TR
5	6	Pierre	IT

```
paises
```

	Codigo	Descripcion
0	ES	España
1	UK	Reino Unido
2	US	Estados Unidos de América
3	DE	Alemania

Skip to main content

- Unión de los dataframes

Left

```
pd.merge(personas, paises,
left_on='Nacionalidad',
right_on='Codigo',
how='left')
```

	ID	Nombre	Nacionalidad	Codigo	Descripcion
0	1	Laura	ES	ES	España
1	2	John	US	US	Estados Unidos de América
2	3	Esteban	ES	ES	España
3	4	Catherine	UK	UK	Reino Unido
4	5	Ilayda	TR	NaN	NaN
5	6	Pierre	IT	NaN	NaN

Right

```
pd.merge(personas,
paises,
left_on='Nacionalidad',
right_on='Codigo',
how='right')
```

	ID	Nombre	Nacionalidad	Codigo	Descripcion
0	1.0	Laura	ES	ES	España
1	3.0	Esteban	ES	ES	España
2	4.0	Catherine	UK	UK	Reino Unido
3	2.0	John	US	US	Estados Unidos de América
4	NaN	NaN	NaN	DE	Alemania

```
merge_right[merge_right["Nombre"].isna()]
```

	ID	Nombre	Nacionalidad	Codigo	Descripcion
4	NaN	NaN	NaN	DE	Alemania

Inner

```
pd.merge(personas,
paises,
left_on='Nacionalidad',
right_on='Codigo',
how='inner')
```

	ID	Nombre	Nacionalidad	Codigo	Descripcion
0	1	Laura	ES	ES	España
1	3	Esteban	ES	ES	España
2	2	John	US	US	Estados Unidos de América
3	4	Catherine	UK	UK	Reino Unido

```
pd.merge(personas,
paises,
left_on='Nacionalidad',
right_on='Codigo',
how='outer')
```

	ID	Nombre	Nacionalidad	Codigo	Descripcion
0	1.0	Laura	ES	ES	España
1	3.0	Esteban	ES	ES	España
2	2.0	John	US	US	Estados Unidos de América
3	4.0	Catherine	UK	UK	Reino Unido
4	5.0	Ilayda	TR	NaN	NaN
5	6.0	Pierre	IT	NaN	NaN
6	NaN	NaN	NaN	DE	Alemania

Outer

```
pd.merge(personas,
paises,
left_on='Nacionalidad',
right_on='Codigo', how='outer')
```

	ID	Nombre	Nacionalidad	Codigo	Descripcion
0	1.0	Laura	ES	ES	España
1	3.0	Esteban	ES	ES	España
2	2.0	John	US	US	Estados Unidos de América
3	4.0	Catherine	UK	UK	Reino Unido
4	5.0	Ilayda	TR	NaN	NaN
5	6.0	Pierre	IT	NaN	NaN
6	NaN	NaN	NaN	DE	Alemania

4. Visualización de datos

Matplotlib es la librería base para gráficos en python. Por su parte, seaborn es una librería de alto nivel

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
```

```
print(f'Pandas - {pd.__version__}')
print(f'Numpy - {np.__version__}')
print(f'Seaborn - {sns.__version__}')
```

```
Pandas - 1.5.3
Numpy - 1.24.3
Seaborn - 0.12.2
```

```
iris = pd.read_csv('iris.csv')
```

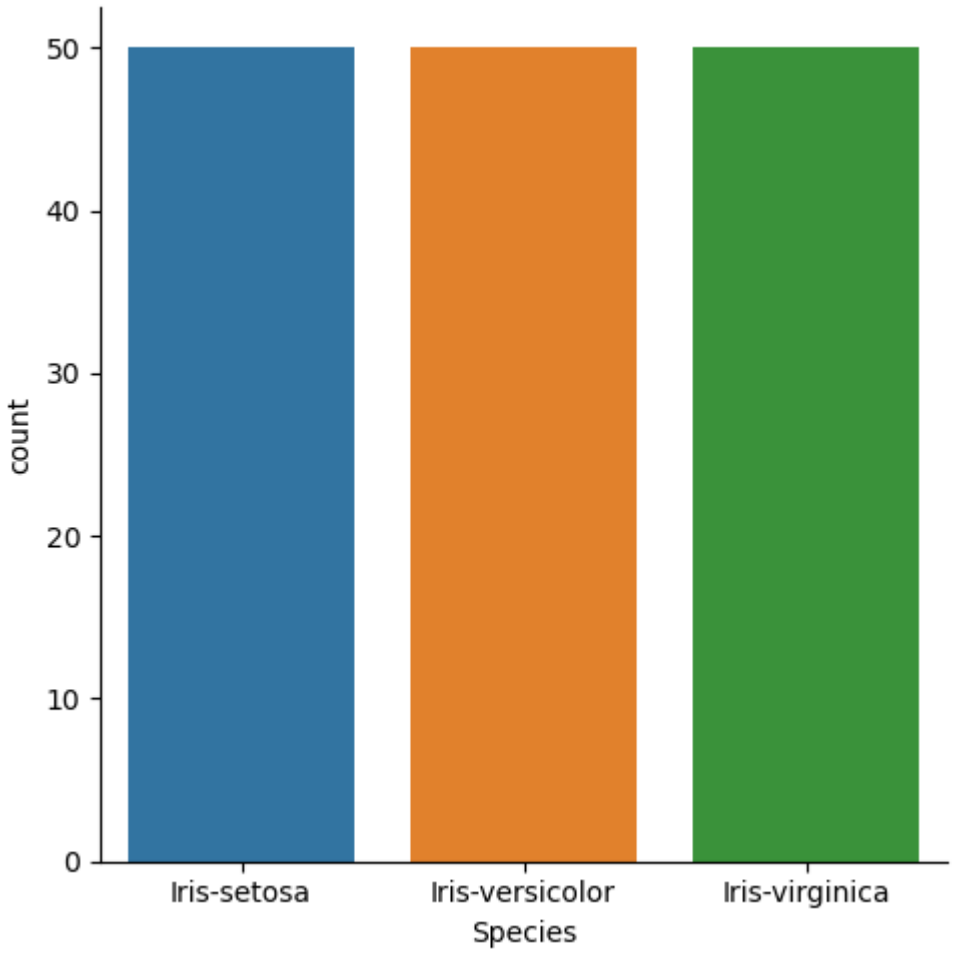
```
iris.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

Skip to main content

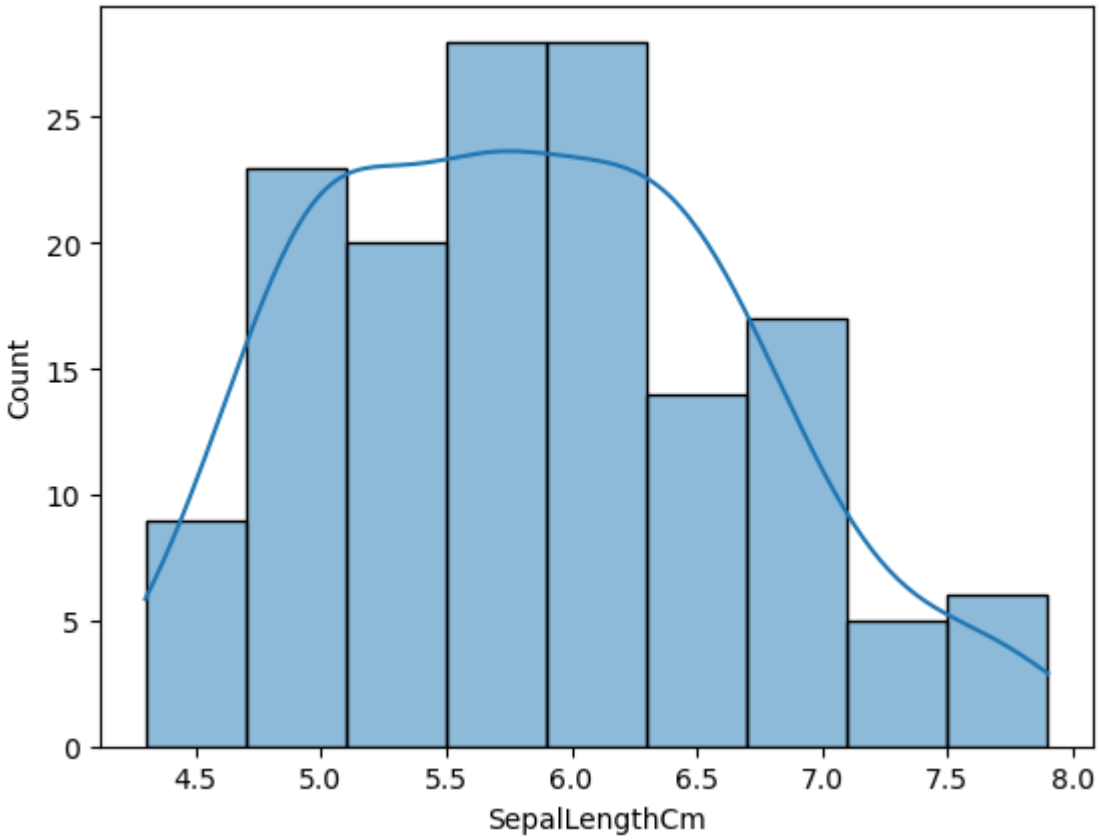
- Gráfico de barras

```
sns.catplot(data=iris, x='Species', kind='count')
plt.show()
```



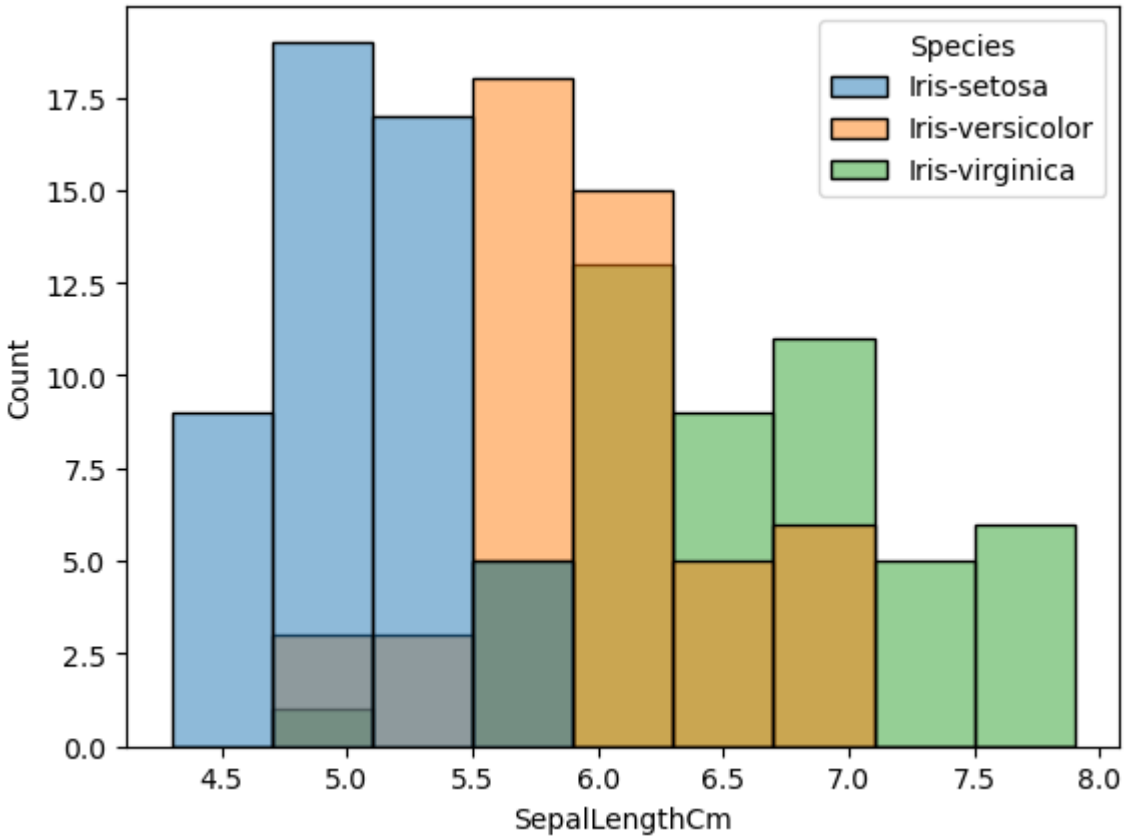
- Histograma

```
sns.histplot(data=iris, x='SepalLengthCm', kde=True)
plt.show()
```



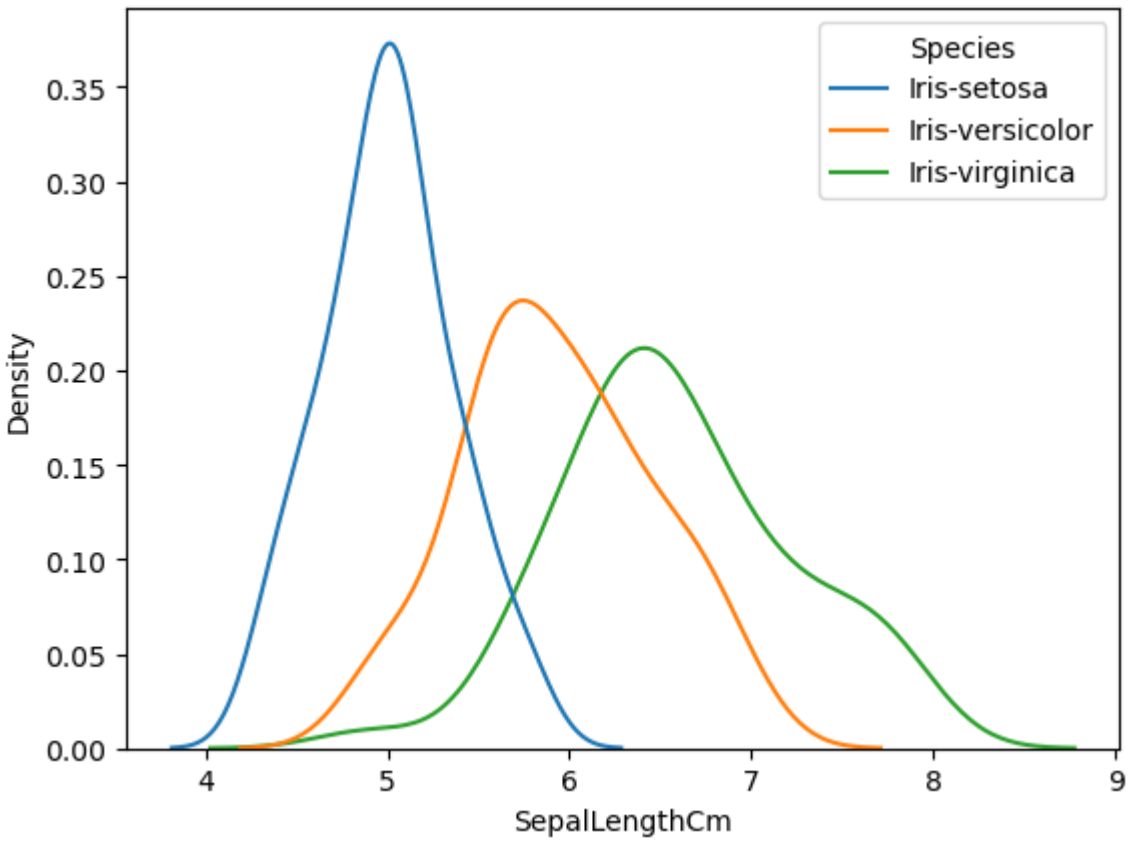

```
sns.histplot(data=iris, x='SepalLengthCm', hue='Species')
plt.show()
```

Skip to main content




```
sns.kdeplot(data=iris, x='SepalLengthCm', hue='Species')
plt.show()
```

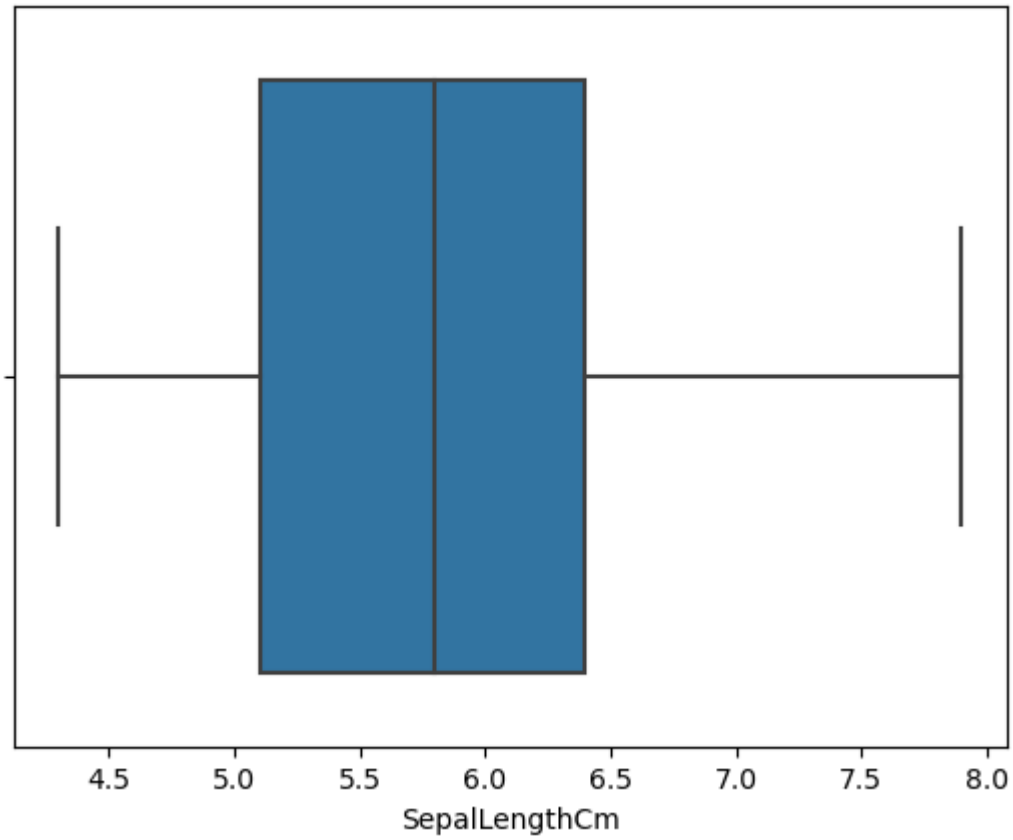
Skip to main content



- Gráfico de cajas

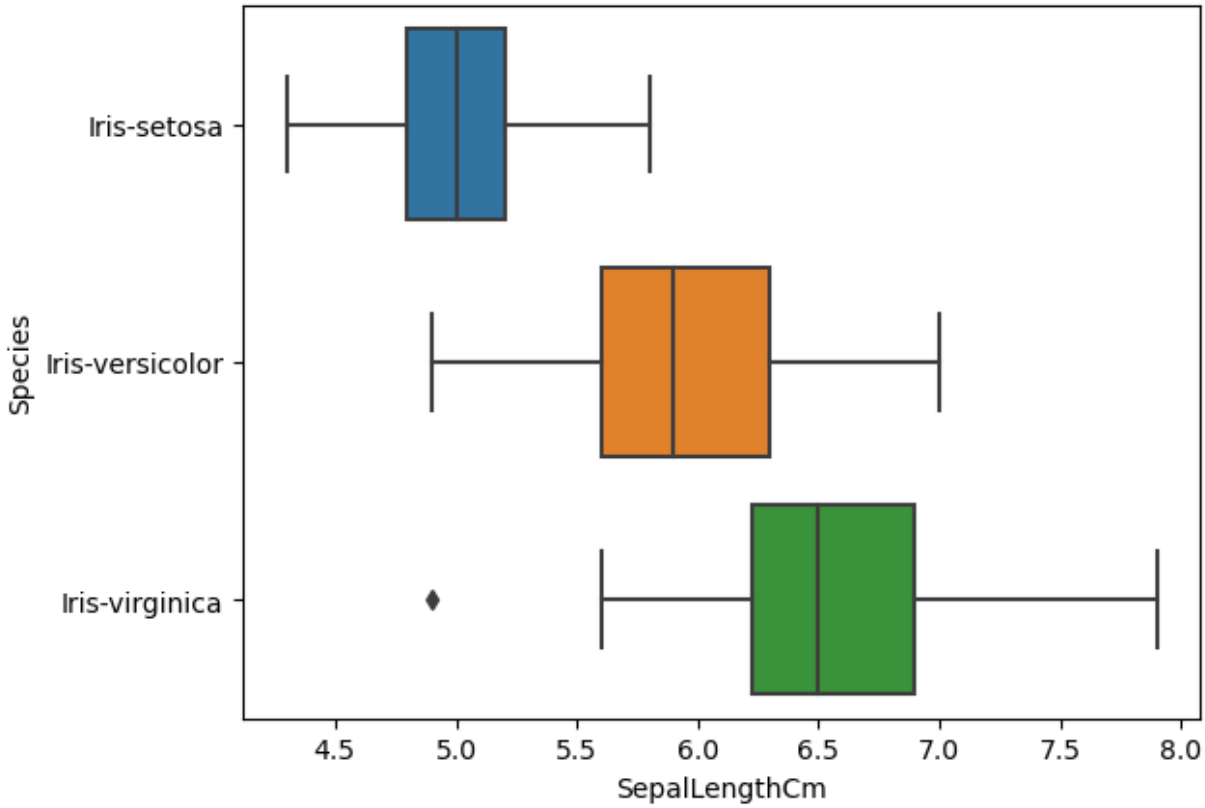
```
sns.boxplot(data=iris, x='SepalLengthCm')
plt.show()
```

Skip to main content




```
sns.boxplot(data=iris, x='SepalLengthCm', y='Species')
plt.show()
```

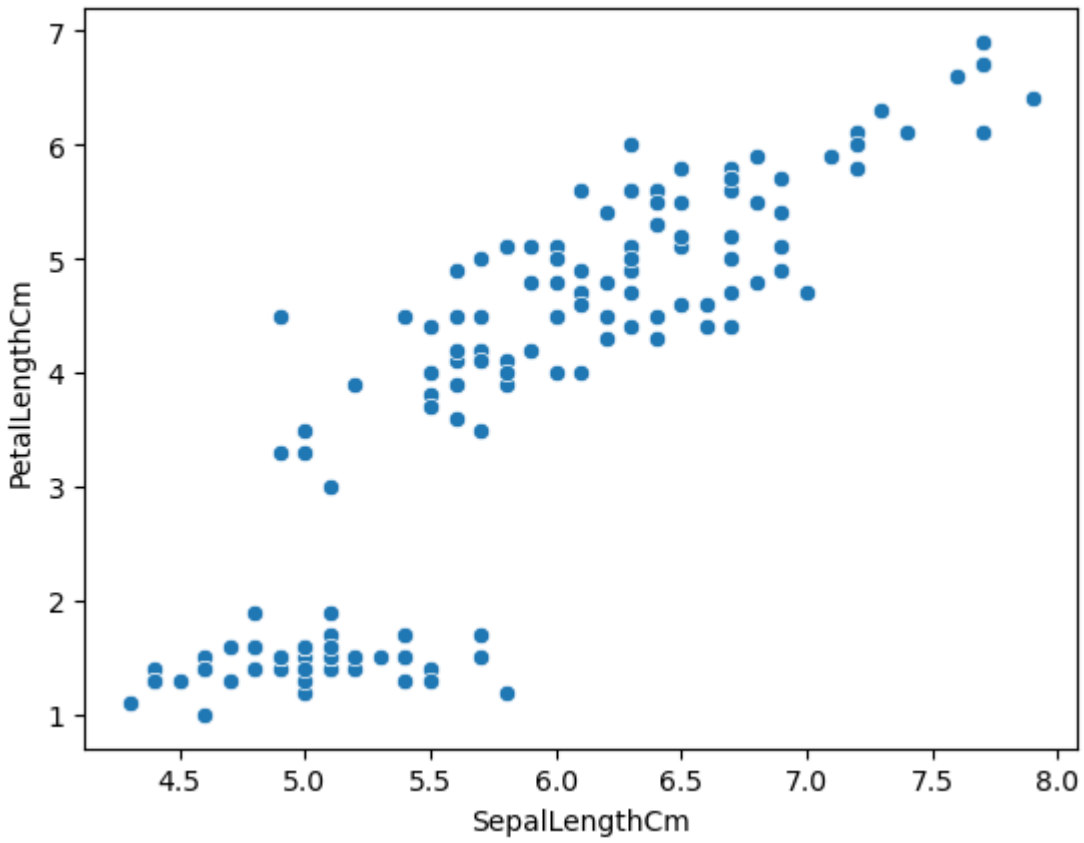
Skip to main content



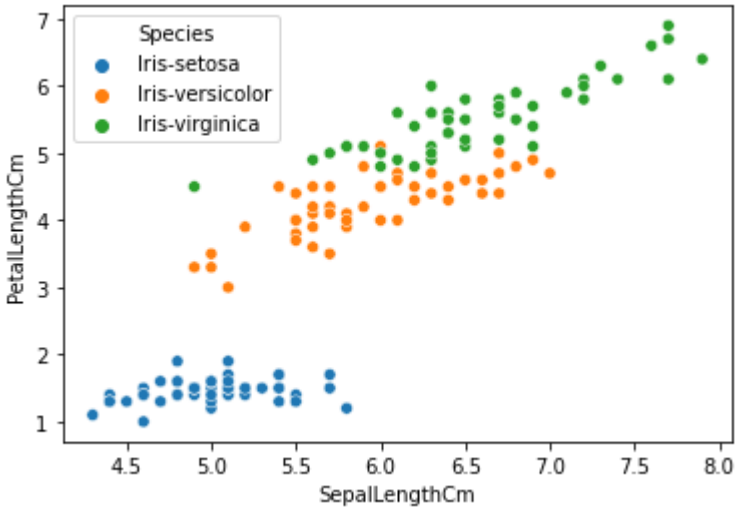
- Gráficos de dispersión

```
sns.scatterplot(data=iris, x='SepalLengthCm', y='PetalLengthCm')
plt.show()
```

Skip to main content

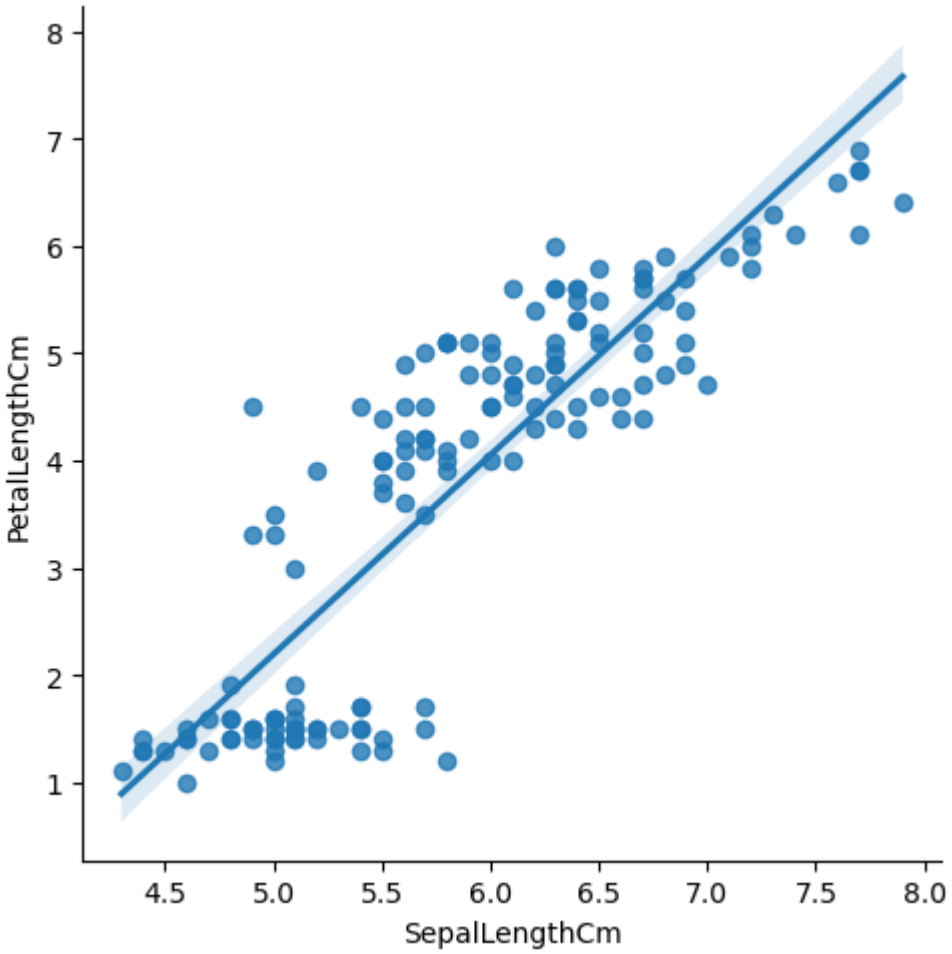



```
sns.scatterplot(data=iris, x='SepalLengthCm', y='PetalLengthCm', hue='Species')
plt.show()
```




```
sns.lmplot(data=iris, x='SepalLengthCm', y='PetalLengthCm')
plt.show()
```

Skip to main content



- Gráfico de correlación

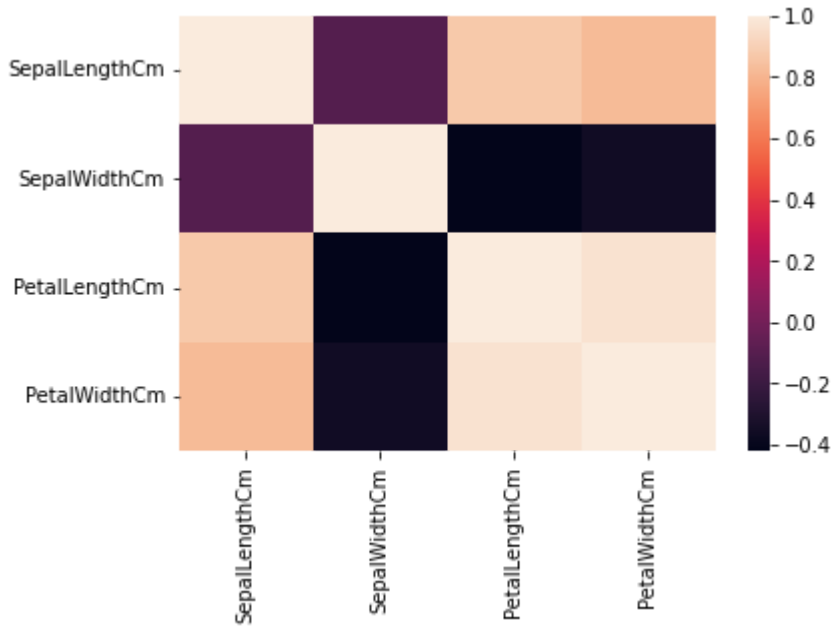
```
correlacion = iris[['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']].corr()
```

```
correlacion
```

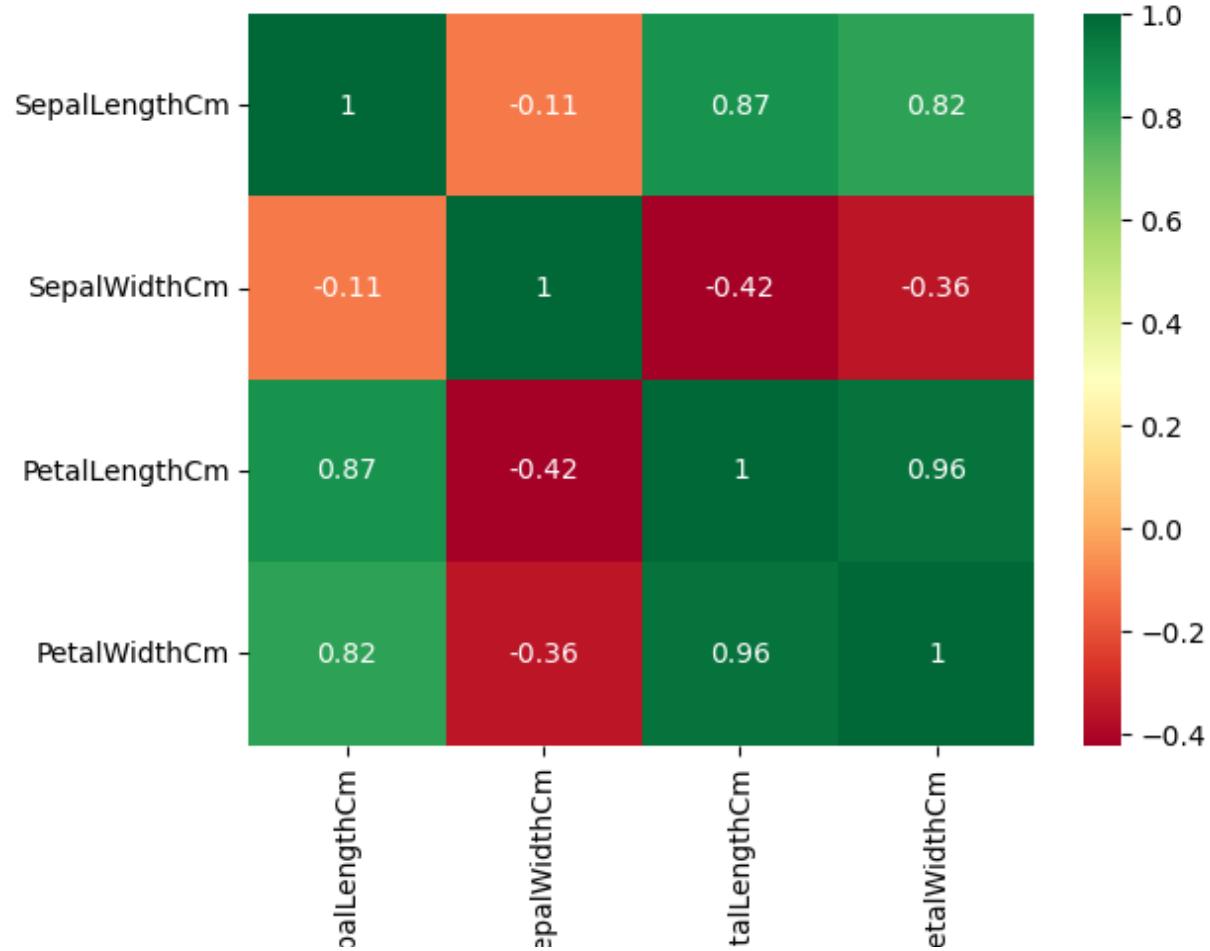
	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
SepalLengthCm	1.000000	-0.109369	0.871754	0.817954
SepalWidthCm	-0.109369	1.000000	-0.420516	-0.356544
PetalLengthCm	0.871754	-0.420516	1.000000	0.962757
PetalWidthCm	0.817954	-0.356544	0.962757	1.000000

```
sns.heatmap(correlacion)
plt.show()
```

Skip to main content




```
sns.heatmap(correlacion, cmap='RdYlGn', annot=True)
plt.show()
```

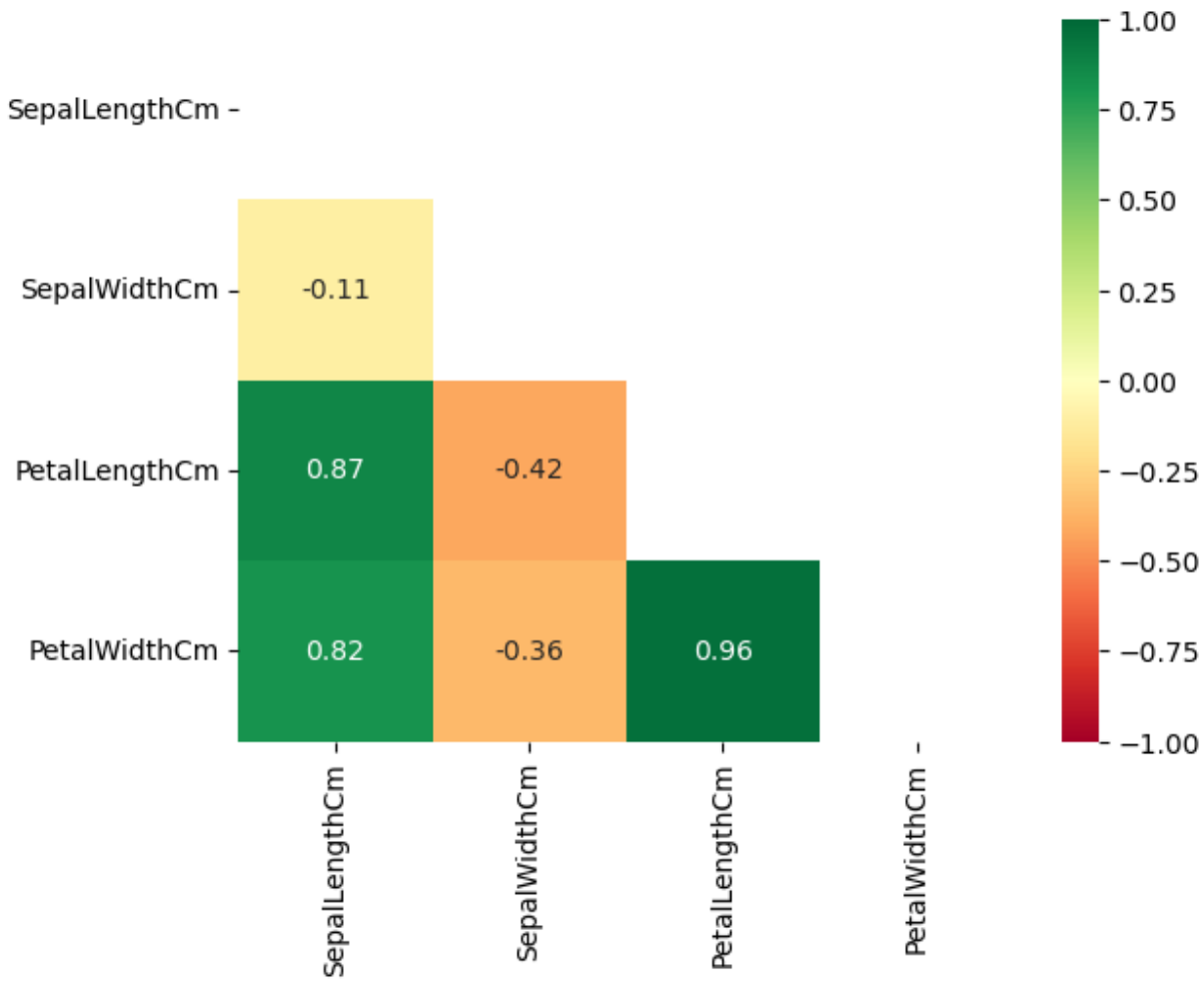


Skip to main content

Como cor(x,x)=1 y cor(x,y)=cor(y,x) se puede disponer solo de la diagonal inferior.

```
#La siguiente función elimina la parte superior de la matriz de correlaciones
mask = np.zeros(correlacion.shape, dtype=bool)
mask[np.triu_indices(len(mask))] = True

sns.heatmap(correlacion, annot = True, vmin = -1, vmax = 1, cmap = 'RdYlGn', mask=mask)
#cmap para elegir la gama de colores y vmin y vmax identificar todo el espectro de valores de
la correlación de pearson [-1,1]
plt.show()
```



Skip to main content