

# Taller de programación en Python para estadística descriptiva

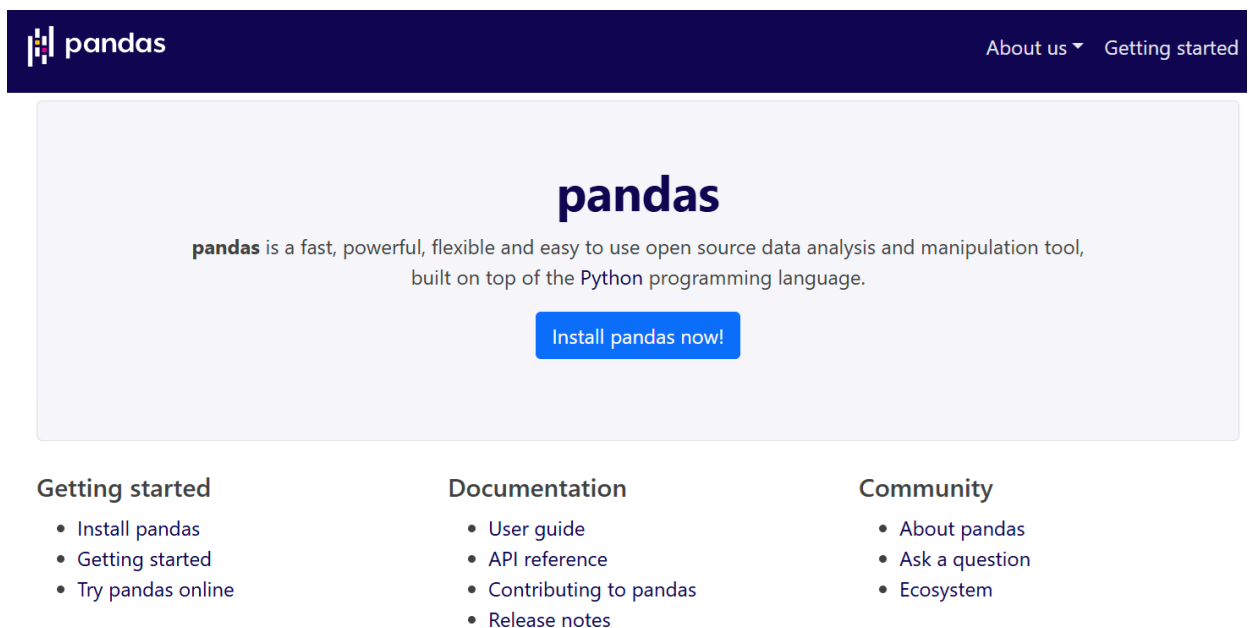
Elaborado por Joshua Martínez Domínguez

22/03/2025

Este documento contiene los temas desarrollados del *Taller de programación en Python para estadística descriptiva* con ejemplos en celdas de código. Este notebook corresponde a la sesión 4.

## Pandas

Pandas es una herramienta poderosa, rápida, flexible y de fácil uso de análisis y manipulación de datos de código abierto, contruida en el lenguaje de programación Python.



**pandas**

**pandas** is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

[Install pandas now!](#)

<b>Getting started</b> <ul style="list-style-type: none"><li>• <a href="#">Install pandas</a></li><li>• <a href="#">Getting started</a></li><li>• <a href="#">Try pandas online</a></li></ul>	<b>Documentation</b> <ul style="list-style-type: none"><li>• <a href="#">User guide</a></li><li>• <a href="#">API reference</a></li><li>• <a href="#">Contributing to pandas</a></li><li>• <a href="#">Release notes</a></li></ul>	<b>Community</b> <ul style="list-style-type: none"><li>• <a href="#">About pandas</a></li><li>• <a href="#">Ask a question</a></li><li>• <a href="#">Ecosystem</a></li></ul>
---	--	--

La instalación más sencilla y fácil de realizar es por medio de la distribución Anaconda, una distribución multiplataforma para análisis de datos y cómputo científico. El administrador del paquete Conda es el método de instalación recomendado para la mayoría de los usuarios.

También es posible instalarlo usando el comando

```
pip install pandas
```

Para comenzar a usarlo utilizamos el comando

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

In [2]:

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

## NumPy

NumPy es un paquete fundamental para cómputo científico en Python. Es una librería de Python que brinda un objeto de arreglos multidimensional, varios objetos derivados (como arreglos enmascarados y matrices) y una variedad de rutinas para operaciones rápidas con arreglos, incluyendo operaciones matemáticas, lógicas, manipulación de forma, ordenamiento, selección, álgebra lineal básica, operaciones estadísticas básicas, simulación aleatoria y mucho más.

El corazón de la paquetería es el objeto `ndarray`. Esto encapsula arreglos  $n$ -dimensionales de tipos de datos homogéneos. Hay varias diferencias entre las secuencias o *arreglos* de NumPy y las secuencias estándar de Python:

- Los arreglos NumPy tienen un tamaño fijo al momento de su creación. A diferencia de las listas de Python que pueden crecer dinámicamente. Cambiar el tamaño de un *ndarray* creará un nuevo arreglo y borrará el original.
- Los elementos de un arreglo NumPy requieren datos de un mismo tipo y serán del mismo tamaño en la memoria. La excepción es que se pueden tener arreglos de objetos, que permitiría arreglos de elementos de diferentes tamaños.
- Los arreglos NumPy facilitan operaciones matemáticas avanzadas y otro tipo de operaciones en grandes números de datos.
- Paqueterías de Python usan arreglos NumPy para hacer software eficiente.

Los atributos de un objeto *ndarray* son:

- `ndarray.ndim`

El número de dimensiones de un arreglo

- `ndarray.shape`

Es una tupla de enteros que indica el tamaño de un arreglo en cada dimensión. Para una matriz con  $n$  filas y  $m$  columnas, `shape` sería de  $(n, m)$

- `ndarray.size`

El total de elementos del arreglo

- `ndarray.dtype`

Describe el tipo de elementos en el arreglo. Por ejemplo: `numpy.int32`, `numpy.int16`, and `numpy.float64`.

In [5]:

```
a = np.arange(15).reshape(3, 5)
```

```
a
```

```
Out[5]: array([[ 0,  1,  2,  3,  4],
               [ 5,  6,  7,  8,  9],
               [10, 11, 12, 13, 14]])
```

```
In [6]: a.shape
```

```
Out[6]: (3, 5)
```

```
In [7]: a.ndim
```

```
Out[7]: 2
```

```
In [9]: a.dtype.name
```

```
Out[9]: 'int32'
```

```
In [10]: a.itemsize
```

```
Out[10]: 4
```

```
In [11]: a.size
```

```
Out[11]: 15
```

```
In [12]: type(a)
```

```
Out[12]: numpy.ndarray
```

```
In [13]: b = np.array([6, 7, 8])
b
```

```
Out[13]: array([6, 7, 8])
```

```
In [14]: type(b)
```

```
Out[14]: numpy.ndarray
```

array transforma secuencias de secuencias en arreglos de dos dimensiones; secuencias de secuencias de secuencias en arreglos de tresdimensiones y así sucesivamente

```
In [15]: b = np.array([(1.5, 2, 3), (4, 5, 6)])
b
```

```
Out[15]: array([[1.5, 2. , 3. ],
               [4. , 5. , 6. ]])
```

La función `zeros` crea un arreglo lleno de ceros, la función `ones` crea un arreglo lleno de unos y la función `empty` crea un arreglo cuyo contenido inicial es aleatorio y depende del estado de la memoria

```
In [16]: np.zeros((3,4))
```

```
Out[16]: array([[0., 0., 0., 0.],
               [0., 0., 0., 0.],
               [0., 0., 0., 0.]])
```

```
In [18]: np.ones((2, 3, 4), dtype = np.int16)
```

```
Out[18]: array([[1, 1, 1, 1],
               [1, 1, 1, 1],
               [1, 1, 1, 1]],

               [[1, 1, 1, 1],
               [1, 1, 1, 1],
               [1, 1, 1, 1]]], dtype=int16)
```

```
In [19]: np.empty((2, 3))
```

```
Out[19]: array([[1.5, 2. , 3. ],
               [4. , 5. , 6. ]])
```

Para crear secuencias de números, NumPy brinda la función `arange` que es análoga a la construida en Python `range`, pero devuelve un arreglo

```
In [20]: np.arange(10, 30, 5)
```

```
Out[20]: array([10, 15, 20, 25])
```

Es posible generar números aleatorios en formato de arreglo, similar a las funciones `np.zeros` y `np.ones` usando la función `np.random.randn`

```
In [9]: np.random.rand(3,2)
```

```
Out[9]: array([[0.61545379, 0.89856205],
               [0.62030844, 0.32823341],
               [0.98895611, 0.51997913]])
```

## Números Aleatorios

Es posible generar números aleatorios desde NumPy.

```
In [10]: default_rng = np.random.default_rng()
         default_rng
```

```
Out[10]: Generator(PCG64) at 0x209ED0A92E0
```

Desde la versión 1.17, NumPy utiliza el algoritmo PCG64 ( generador congruencial permutado-64 ), más eficiente. Este algoritmo produce números menos predecibles, como lo demuestra su

rendimiento en la prueba estadística TestU01 , estándar de la industria. PCG64 también es más rápido y requiere menos recursos.

```
In [11]: default_rng.random()
```

```
Out[11]: 0.9909977318781663
```

```
In [12]: default_rng.random()
```

```
Out[12]: 0.6546357810957991
```

Por defecto, `Generator.random()` devuelve un valor flotante de 64 bits en el intervalo semiabierto `[0.0, 1.0]`. Esta notación se utiliza para definir un rango de números. El `[` es el parámetro cerrado e indica inclusividad.

## Distribuciones

El bloque de funciones de distribuciones ofrece numerosas funciones que permiten generar un array de números aleatorios a partir de distribuciones de todo tipo:

- normal

La función `numpy.random.normal` genera un array del tamaño indicado a partir de una distribución normal o gaussiana de una cierta media y desviación estándar:

```
In [15]: # Generar un arreglo de 10 números con distribución Normal de media y desviación estándar
default_rng.normal(loc = 5, scale = 2, size= 10)
```

```
Out[15]: array([3.63438934, 6.7636528 , 0.49117109, 3.72654303, 7.20807751,
                1.72275769, 3.10143125, 2.87967712, 1.33990512, 5.02979564])
```

En este ejemplo hemos generado un arreglo de una dimensión y diez valores con números aleatorios a partir de una distribución gaussiana de media 5 y desviación estándar 2.

```
In [13]: # Generar un arreglo de 10 números con distribución Normal Estandar
default_rng.standard_normal(10)
```

```
Out[13]: array([-1.19453041,  0.55036602,  1.05802593, -0.08169782, -1.33396343,
                -1.25209265, -1.01817469, -1.38059364, -0.96921581,  1.44737375])
```

```
In [16]: np.random.normal(loc = 5, scale = 2, size= 10)
```

```
Out[16]: array([6.29918833, 3.3860609 , 6.66436928, 3.55172745, 9.06425688,
                5.51878877, 2.14225939, 5.13640607, 8.22679591, 9.03214867])
```

```
In [18]: np.random.random(3)
```

```
Out[18]: array([0.23943589, 0.81701782, 0.57662235])
```

Algunas otras distribuciones disponibles:

`numpy.random.beta` : Genera muestras aleatorias a partir de una distribución beta

`numpy.random.chisquare` : Genera muestras aleatorias a partir de una distribución chi-cuadrado

`numpy.random.exponential` : Genera muestras aleatorias a partir de una distribución exponencial

`numpy.random.poisson` : Genera muestras aleatorias a partir de una distribución de Poisson

## Estructuras básicas de datos en Pandas

Pandas tiene dos tipos de clases para manejar datos:

- Serie: Un arreglo de una dimensión que contiene datos de cualquier tipo como enteros, cadenas, objetos de Python, etc.
- DataFrame: Una estructura de datos de dos dimensiones que contiene arreglos de filas y comunas.

```
In [3]: #Creamos una Serie
s = pd.Series([1, 3, 5, np.nan, 6, 8])
s
```

```
Out[3]: 0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

Es posible crear un `DataFrame` usando un diccionario de objetos donde las llaves son las etiquetas de las columnas y los valores son los valores de las columnas.

```
In [3]: #Creamos un DataFrame con diccionario
df = pd.DataFrame({"columna1": [1, 2, 3, 4], "columna2": ["a", "b", "c", "d"]})
df
```

```
Out[3]:
```

	columna1	columna2
0	1	a
1	2	b
2	3	c
3	4	d

Es posible crear un `DataFrame` usando un arreglo NumPy.

```
In [6]: #Creamos un DataFrame con arreglo NumPy
minutos = np.arange(10, 30, 5)
minutos
```

Out[6]: array([10, 15, 20, 25])

In [8]: `df2 = pd.DataFrame(np.random.randn(4, 3), index = minutos, columns = list("123"))`  
`df2`

Out[8]:

	1	2	3
10	-0.122705	-1.374800	-0.558771
15	-0.416494	-1.974314	-1.529811
20	0.614818	0.156526	-0.831648
25	0.594211	1.160695	1.049659

In [19]: `# Veamos los tipos de datos diferentes presentes en las columnas`  
`df.dtypes`

Out[19]: `columna1 int64`  
`columna2 object`  
`dtype: object`

## Importar y exportar datos

### CSV

Es posible escribir un archivo csv usando `DataFrame.to_csv()`

In [20]: `df.to_csv("probando_csv")`

Es posible leer un archivo csv usando `read_csv()`

In [21]: `pd.read_csv("probando_csv")`

Out[21]:

	Unnamed: 0	columna1	columna2
0	0	1	a
1	1	2	b
2	2	3	c
3	3	4	d

### Excel

Es posible escribir un archivo csv usando `DataFrame.to_excel()`

In [22]: `df.to_excel("probando_excel.xlsx", sheet_name = "Sheet1")`

Es posible leer um archivo de excel usando `read_excel()`

```
In [23]: pd.read_excel("probando_excel.xlsx", "Sheet1", index_col = None, na_values = ["NA"])
```

```
Out[23]:
```

Unnamed: 0	columna1	columna2	
0	0	1	a
1	1	2	b
2	2	3	c
3	3	4	d

## R

Usualmente los científicos de datos guardan sus datos de R en archivos csv y los importan a Python. El paquete estadístico R es similar a la combinación de Python y Pandas y varios científicos usan ambos; manipulando datos en Python y análisis estadístico en R o viceversa, dependiendo de sus paquetes preferidos.

Es posible usar el paquete `pyreadr`. No obstante pueden presentarse incompatibilidades.

Importaremos un conjunto de datos precargados usados en R:

The screenshot shows the RStudio environment. The main window displays the 'iris' dataset with columns: Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, and Species. The console shows the R version 4.4.1 and a message about the license. The Environment pane on the right shows the 'Global Environment' with an empty environment. The Packages pane on the right lists installed packages, including 'askpass', 'backports', 'base64enc', 'bit', 'bit64', 'blob', 'brew', 'broom', 'bslib', 'cachem', 'callr', 'cellranger', 'cli', 'clipr', 'collections', 'colorspace', 'commonmark', and 'conflicted'.

```
write.csv(iris, file = "C:/Users/josh_/Desktop/Libros/UNISA/Pregrado/2024 - 2025/Taller de python para estadística/iris.csv", row.names=FALSE)
```



```
In [29]: pd.read_csv("iris.csv")
```

Out[29]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

```
In [30]: iris = pd.read_csv("iris.csv")
#Podemos ver porciones de los datos cuando son numerosos
iris.head()
```

Out[30]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
In [31]: iris.tail()
```

Out[31]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

```
In [32]: iris.index
```

Out[32]: RangeIndex(start=0, stop=150, step=1)

```
In [34]: iris.columns
```

Out[34]: Index(['Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width',  
              'Species'],  
              dtype='object')

```
In [36]: df.to_numpy()
```

Out[36]: array([[1, 'a'],  
                 [2, 'b'],  
                 [3, 'c'],  
                 [4, 'd']], dtype=object)

```
In [38]: iris.dtypes
```

Out[38]: Sepal.Length    float64  
Sepal.Width     float64  
Petal.Length    float64  
Petal.Width     float64  
Species         object  
dtype: object

```
In [39]: #Es posible transponer los datos  
iris.T
```

Out[39]:

	0	1	2	3	4	5	6	7	8	9	...	140
Sepal.Length	5.1	4.9	4.7	4.6	5.0	5.4	4.6	5.0	4.4	4.9	...	6.7
Sepal.Width	3.5	3.0	3.2	3.1	3.6	3.9	3.4	3.4	2.9	3.1	...	3.1
Petal.Length	1.4	1.4	1.3	1.5	1.4	1.7	1.4	1.5	1.4	1.5	...	5.6
Petal.Width	0.2	0.2	0.2	0.2	0.2	0.4	0.3	0.2	0.2	0.1	...	2.4
Species	setosa	setosa	setosa	setosa	setosa	setosa	setosa	setosa	setosa	setosa	...	virginica

5 rows × 150 columns



```
In [41]: #Se pueden renombrar las columnas  
nombres = ["Longitud_Sepalo", "Ancho_Sepalo", "Longitud_Petalo", "Ancho_Petalo", "Especi  
iris.columns = nombres
```

```
In [42]: iris.head()
```

Out[42]:

	Longitud_Sepalo	Ancho_Sepalo	Longitud_Petalo	Ancho_Petalo	Especie
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [43]:

```
iris.columns
```

Out[43]: Index(['Longitud\_Sepalo', 'Ancho\_Sepalo', 'Longitud\_Petalo', 'Ancho\_Petalo', 'Especie'], dtype='object')

In [46]:

```
# Seleccionar columnas
iris[["Longitud_Sepalo", "Longitud_Petalo"]]
```

Out[46]:

	Longitud_Sepalo	Longitud_Petalo
0	5.1	1.4
1	4.9	1.4
2	4.7	1.3
3	4.6	1.5
4	5.0	1.4
...	...	...
145	6.7	5.2
146	6.3	5.0
147	6.5	5.2
148	6.2	5.4
149	5.9	5.1

150 rows × 2 columns

In [47]:

```
# Seleccionas una sola columna que devuelva un formato Series
iris["Especie"]
```

Out[47]:

0	setosa
1	setosa
2	setosa
3	setosa
4	setosa
...	...
145	virginica
146	virginica
147	virginica

148     virginica  
149     virginica  
Name: Especie, Length: 150, dtype: object

```
In [48]: # Seleccionar filas
iris[:5]
```

Out[48]:

	Longitud_Sepalo	Ancho_Sepalo	Longitud_Petalo	Ancho_Petalo	Especie
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
In [50]: # Seleccionar cruces de filas y columnas
iris.iloc[[5, 6, 7, 8, 9, 10], [1, 4]]
```

Out[50]:

	Ancho_Sepalo	Especie
5	3.9	setosa
6	3.4	setosa
7	3.4	setosa
8	2.9	setosa
9	3.1	setosa
10	3.7	setosa

```
In [51]: # Filtrado de datos
iris[iris["Longitud_Sepalo"] > 5]
```

Out[51]:

	Longitud_Sepalo	Ancho_Sepalo	Longitud_Petalo	Ancho_Petalo	Especie
0	5.1	3.5	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
10	5.4	3.7	1.5	0.2	setosa
14	5.8	4.0	1.2	0.2	setosa
15	5.7	4.4	1.5	0.4	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica

	Longitud_Sepalo	Ancho_Sepalo	Longitud_Petalo	Ancho_Petalo	Especie
149	5.9	3.0	5.1	1.8	virginica

118 rows × 5 columns

In [53]:

iris[iris["Especie"] == "setosa"]

Out[53]:

	Longitud_Sepalo	Ancho_Sepalo	Longitud_Petalo	Ancho_Petalo	Especie
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
6	4.6	3.4	1.4	0.3	setosa
7	5.0	3.4	1.5	0.2	setosa
8	4.4	2.9	1.4	0.2	setosa
9	4.9	3.1	1.5	0.1	setosa
10	5.4	3.7	1.5	0.2	setosa
11	4.8	3.4	1.6	0.2	setosa
12	4.8	3.0	1.4	0.1	setosa
13	4.3	3.0	1.1	0.1	setosa
14	5.8	4.0	1.2	0.2	setosa
15	5.7	4.4	1.5	0.4	setosa
16	5.4	3.9	1.3	0.4	setosa
17	5.1	3.5	1.4	0.3	setosa
18	5.7	3.8	1.7	0.3	setosa
19	5.1	3.8	1.5	0.3	setosa
20	5.4	3.4	1.7	0.2	setosa
21	5.1	3.7	1.5	0.4	setosa
22	4.6	3.6	1.0	0.2	setosa
23	5.1	3.3	1.7	0.5	setosa
24	4.8	3.4	1.9	0.2	setosa
25	5.0	3.0	1.6	0.2	setosa
26	5.0	3.4	1.6	0.4	setosa
27	5.2	3.5	1.5	0.2	setosa

	Longitud_Sepalo	Ancho_Sepalo	Longitud_Petalo	Ancho_Petalo	Especie
28	5.2	3.4	1.4	0.2	setosa
29	4.7	3.2	1.6	0.2	setosa
30	4.8	3.1	1.6	0.2	setosa
31	5.4	3.4	1.5	0.4	setosa
32	5.2	4.1	1.5	0.1	setosa
33	5.5	4.2	1.4	0.2	setosa
34	4.9	3.1	1.5	0.2	setosa
35	5.0	3.2	1.2	0.2	setosa
36	5.5	3.5	1.3	0.2	setosa
37	4.9	3.6	1.4	0.1	setosa
38	4.4	3.0	1.3	0.2	setosa
39	5.1	3.4	1.5	0.2	setosa
40	5.0	3.5	1.3	0.3	setosa
41	4.5	2.3	1.3	0.3	setosa
42	4.4	3.2	1.3	0.2	setosa
43	5.0	3.5	1.6	0.6	setosa
44	5.1	3.8	1.9	0.4	setosa
45	4.8	3.0	1.4	0.3	setosa
46	5.1	3.8	1.6	0.2	setosa
47	4.6	3.2	1.4	0.2	setosa
48	5.3	3.7	1.5	0.2	setosa
49	5.0	3.3	1.4	0.2	setosa

```
In [62]: #Operaciones con columnas
iris["Longitud_Sepalo"] / iris["Longitud_Petalo"]
```

```
Out[62]: 0      3.642857
1      3.500000
2      3.615385
3      3.066667
4      3.571429
...
145    1.288462
146    1.260000
147    1.250000
148    1.148148
149    1.156863
Length: 150, dtype: float64
```

```
In [63]: # Agregar la columna
indice_long_sep_pet = iris["Longitud_Sepalo"] / iris["Longitud_Petalo"]
```

```
iris["indice_long_sep_pet"] = indice_long_sep_pet

      Longitud_Sepalo  Ancho_Sepalo  Longitud_Petalo  Ancho_Petalo  Especie \
0              5.1          3.5          1.4          0.2    setosa
1              4.9          3.0          1.4          0.2    setosa
2              4.7          3.2          1.3          0.2    setosa
3              4.6          3.1          1.5          0.2    setosa
4              5.0          3.6          1.4          0.2    setosa
..              ...          ...          ...          ...      ...
145             6.7          3.0          5.2          2.3  virginica
146             6.3          2.5          5.0          1.9  virginica
147             6.5          3.0          5.2          2.0  virginica
148             6.2          3.4          5.4          2.3  virginica
149             5.9          3.0          5.1          1.8  virginica

      indice_long_sep_pet
0              3.642857
1              3.500000
2              3.615385
3              3.066667
4              3.571429
..              ...
145             1.288462
146             1.260000
147             1.250000
148             1.148148
149             1.156863

[150 rows x 6 columns]
```

```
In [64]: iris
```

Out[64]:

	Longitud_Sepalo	Ancho_Sepalo	Longitud_Petalo	Ancho_Petalo	Especie	indice_long_sep_pet
0	5.1	3.5	1.4	0.2	setosa	3.642857
1	4.9	3.0	1.4	0.2	setosa	3.500000
2	4.7	3.2	1.3	0.2	setosa	3.615385
3	4.6	3.1	1.5	0.2	setosa	3.066667
4	5.0	3.6	1.4	0.2	setosa	3.571429
...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica	1.288462
146	6.3	2.5	5.0	1.9	virginica	1.260000
147	6.5	3.0	5.2	2.0	virginica	1.250000
148	6.2	3.4	5.4	2.3	virginica	1.148148
149	5.9	3.0	5.1	1.8	virginica	1.156863

150 rows × 6 columns

# Estadísticas

La función describe() mostrara un resumen rápido de los datos

```
In [61]: iris.describe()
```

```
Out[61]:
```

	Longitud_Sepalo	Ancho_Sepalo	Longitud_Petalo	Ancho_Petalo
<b>count</b>	150.000000	150.000000	150.000000	150.000000
<b>mean</b>	5.843333	3.057333	3.758000	1.199333
<b>std</b>	0.828066	0.435866	1.765298	0.762238
<b>min</b>	4.300000	2.000000	1.000000	0.100000
<b>25%</b>	5.100000	2.800000	1.600000	0.300000
<b>50%</b>	5.800000	3.000000	4.350000	1.300000
<b>75%</b>	6.400000	3.300000	5.100000	1.800000
<b>max</b>	7.900000	4.400000	6.900000	2.500000

```
In [55]: # Media
iris["Longitud_Petalo"].mean()
```

```
Out[55]: 3.75800000000000027
```

```
In [56]: # Mediana
iris["Longitud_Petalo"].median()
```

```
Out[56]: 4.35
```

```
In [57]: # Varianza
iris["Longitud_Petalo"].var()
```

```
Out[57]: 3.1162778523489942
```

```
In [ ]: # Desviacion estandar
iris["Longitud_Petalo"].std()
```

```
In [60]: # Percentiles
p25 = iris["Longitud_Petalo"].quantile(q = 0.25)
p50 = iris["Longitud_Petalo"].quantile(q = 0.5)
p75 = iris["Longitud_Petalo"].quantile(q = 0.75)

print(p25, p50, p75)
```

```
1.6 4.35 5.1
```

#### Ejercicio 4

Importe un conjunto de datos real, o bien, cree un set de datos ficticio con las herramientas de Python o obtenga las medidas descriptivas de ese conjunto



## Referencias

[https://pandas.pydata.org/docs/getting\\_started/install.html](https://pandas.pydata.org/docs/getting_started/install.html)

<https://numpy.org/doc/stable/>

<https://numpy.org/doc/2.1/reference/random/index.html>

<https://www.datacamp.com/es/tutorial/pandas-tutorial-dataframe-python>

<https://www.youtube.com/watch?v=PvNKKrPE0AI>

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.quantile.html>

<https://4geeks.com/es/how-to/anadir-columna-dataframe-python>

In [ ]: