

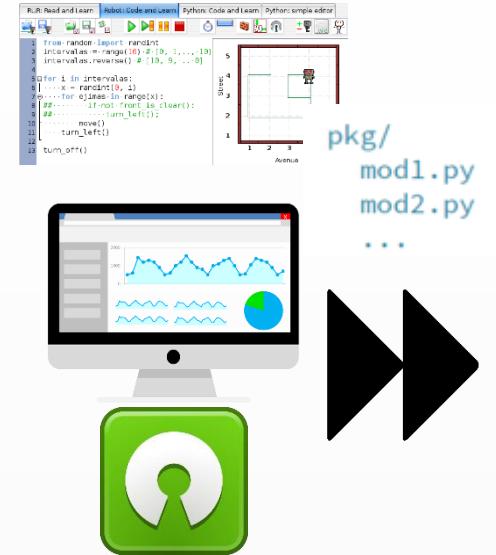


# BLOQUE 1:

## Introducción al Análisis de Datos con Python

# ¿Qué es Python y qué nos proporciona para el análisis de datos?

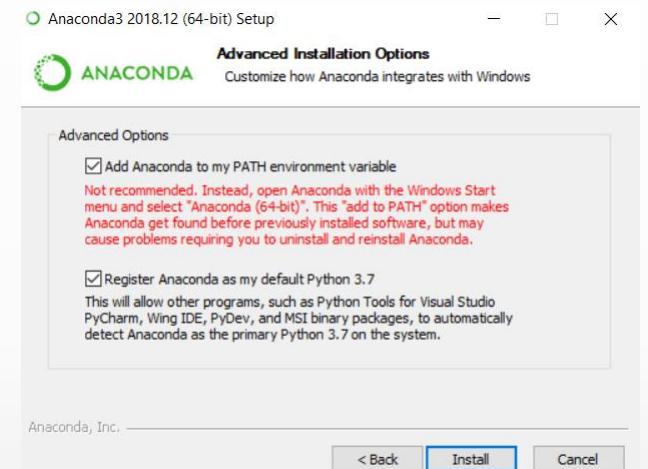
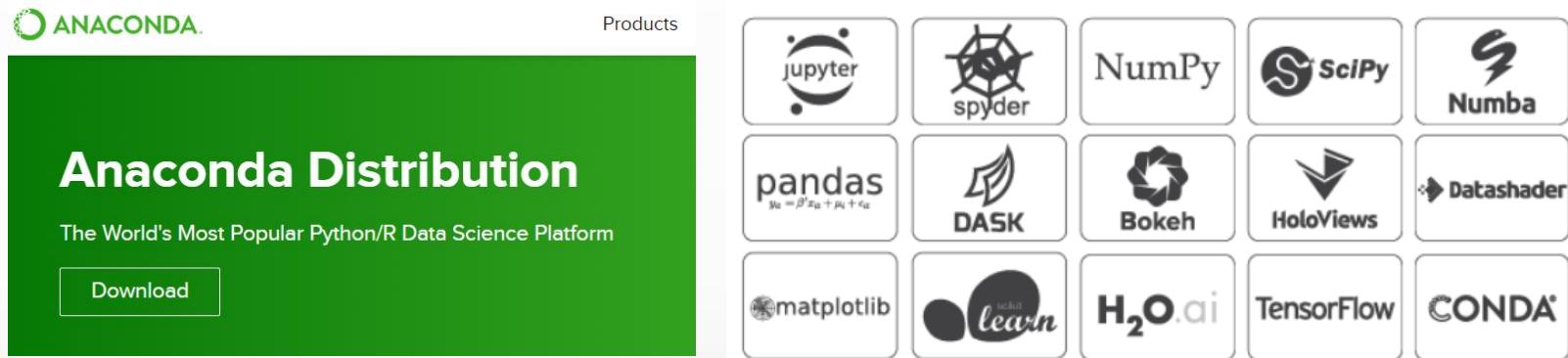
- Python es un lenguaje de programación orientado a objetos de **alto nivel** y de propósito general.
- Python se basa en la **simpleza** y en la facilidad de lectura de código disminuyendo costes de mantenimiento y **facilitando su aprendizaje**.
- Soporta módulos y librerías que favorecen la **reutilización de código**.
- Dispone de potentes estructuras de datos y **librerías enfocadas al análisis de datos**.
- Es un lenguaje **interpretado**, se ejecuta el código línea a línea. No hay un paso de compilación como en otros lenguajes → Rapidez de ejecución.
- Python es **open source** y **multiplataforma** (Windows, Linux, Mac,...).



# Instalación Python + Jupyter

- **ANACONDA:** Distribución de Python con múltiples librerías pre-instaladas para Data Science.

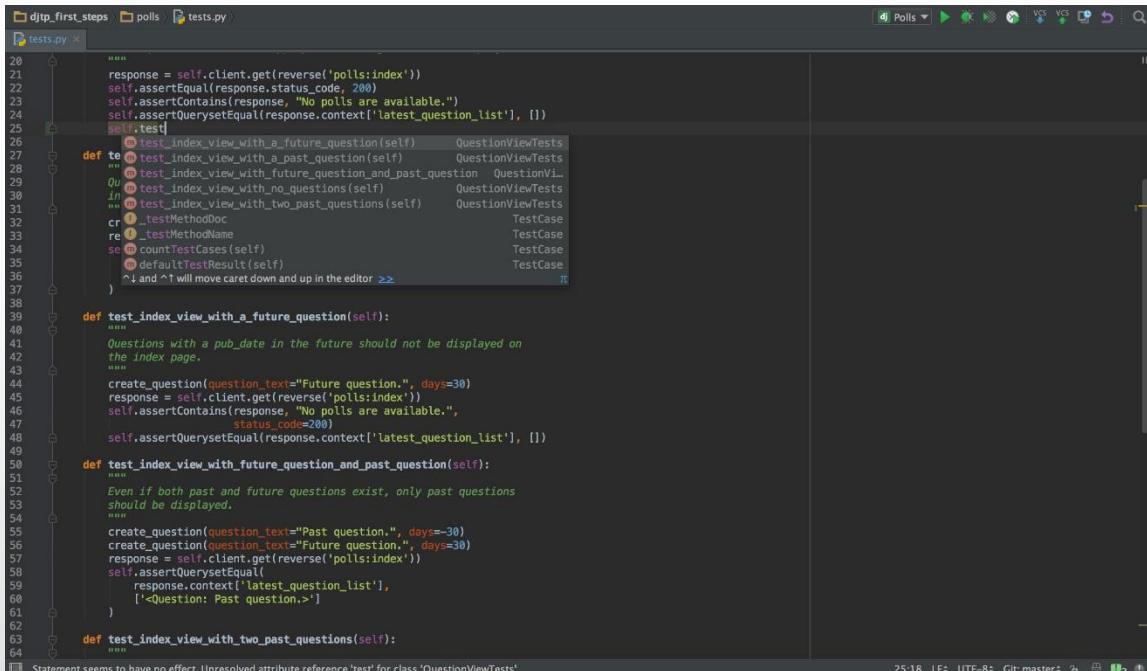
<https://www.anaconda.com/distribution/#download-section>



- **Definir el PATH:** Definir la ruta de la instalación (Python.exe / conda.exe) para mayor facilidad de uso a la hora de instalar cualquier nueva librería:
  - **Ruta habitual de Python:**
  - C:\Users\tu\_nombre\_de\_usuario\Anaconda3\Scripts
  - **Ruta habitual de “conda”** (actualización de paquetes):
  - C:\Users\tu\_nombre\_de\_usuario\Anaconda3\Scripts
- Panel de Control > system > advanced > | Variables de Entorno | > system variables  
→ Añadir a la variable PATH la ruta concreta con ":"

# Instalación Python + Jupyter

**IDE (Programación enfocada a scripts y desarrollos extensos)**



```

20
21     response = self.client.get(reverse('polls:index'))
22     self.assertEqual(response.status_code, 200)
23     self.assertContains(response, 'No polls are available.')
24     self.assertQuerysetEqual(response.context['latest_question_list'], [])
25
26     self.test
27
28     def test_index_view_with_a_future_question(self):
29         """Test index view with a future question."""
30         # Create a question with a pub_date in the future
31         question = Question.objects.create(question_text="Future question.", pub_date=timezone.now() + timedelta(days=30))
32         response = self.client.get(reverse('polls:index'))
33         self.assertEqual(response.status_code, 200)
34         self.assertContains(response, "No polls are available.")
35         self.assertQuerysetEqual(response.context['latest_question_list'], [])
36
37     def test_index_view_with_a_past_question(self):
38         """Test index view with a past question."""
39         # Create a question with a pub_date in the past
40         question = Question.objects.create(question_text="Past question.", pub_date=timezone.now() - timedelta(days=30))
41         response = self.client.get(reverse('polls:index'))
42         self.assertEqual(response.status_code, 200)
43         self.assertContains(response, "Past question")
44         self.assertQuerysetEqual(response.context['latest_question_list'], [question])
45
46     def test_index_view_with_future_and_past_question(self):
47         """Test index view with both past and future questions."""
48         # Create two questions, one past and one future
49         question1 = Question.objects.create(question_text="Past question.", pub_date=timezone.now() - timedelta(days=30))
50         question2 = Question.objects.create(question_text="Future question.", pub_date=timezone.now() + timedelta(days=30))
51         response = self.client.get(reverse('polls:index'))
52         self.assertEqual(response.status_code, 200)
53         self.assertContains(response, "Past question")
54         self.assertContains(response, "Future question")
55         self.assertQuerysetEqual(response.context['latest_question_list'], [question1])
56
57     def test_index_view_with_no_questions(self):
58         """Test index view with no questions available."""
59         response = self.client.get(reverse('polls:index'))
60         self.assertEqual(response.status_code, 200)
61         self.assertContains(response, "No polls are available.")
62         self.assertQuerysetEqual(response.context['latest_question_list'], [])
63
64     def test_index_view_with_two_past_questions(self):
65
66     # Test for unresolved attribute reference 'test' for class 'QuestionViewTests'.
67

```

**JUPYTER NOTEBOOK (Programación interactiva)**



# Importar librerías y fuentes de datos

- **Librerías en Python:**

- Librería = Directorio de Scripts en Python (cada script un modulo con funciones, métodos,...)
- Inmensa variedad de librerías ya definidas para el análisis de datos (Numpy, Pandas, Matplotlib, Scikit-learn,...)
- Para instalar librería: **conda install <librería>**
- Para importar un paquete:

```
pkg/  
  mod1.py  
  mod2.py  
  ...
```

```
In [1]: import numpy as np  
  
In [2]: array_ejemplo = np.array([1,2,3])  
  
In [3]: array_ejemplo  
  
Out[3]: array([1, 2, 3])
```

- **Ruta habitual librerías:**

- C:\Users\tu\_nombre\_de\_usuario\Anaconda3\Lib\site-packages

# Importar librerías y fuentes de datos

## Librería Pandas

- Elemento clave: Dataframe

	País	Población	Área
0	México	129	1973.0
1	España	46	505.0
2	Venezuela	32	916.0

Tipo: Objeto int64 float64

- Convención: import pandas as pd

## Ejercicio:

- Importar fichero csv con la información de población, esperanza de vida y renta per cápita de cada país:

	País	Población	Renta per capita	Esperanza de vida
0	United States	325084756	59939	78,9
1	China	1421021791	8612	76,7
2	Japan	127502725	38214	84,5
3	Germany	82658409	44680	81,2
4	India	1338676785	1980	69,4



## Importar de un CSV

```
df = pd.read_csv(r'file.csv', index_col = 0, nrows=5,  
encoding = "ISO-8859-1", delimiter=';')
```

# Visualización básica con Matplotlib

- Librería de visualización
- Todo tipo de gráficos con amplia configuración

## • Line Plot

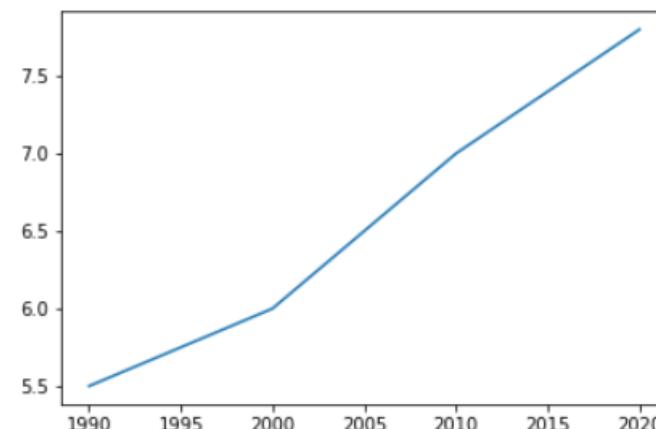
```
In [1]: import matplotlib.pyplot as plt
```

```
In [2]: año = [1990, 2000, 2010, 2020]
pob = [5.5, 6, 7, 7.8]
```

```
In [3]: plt.plot(año, pob)
```

```
Out[3]: <matplotlib.lines.Line2D at 0x912b208>
```

```
In [4]:
```



## • Scatter Plot

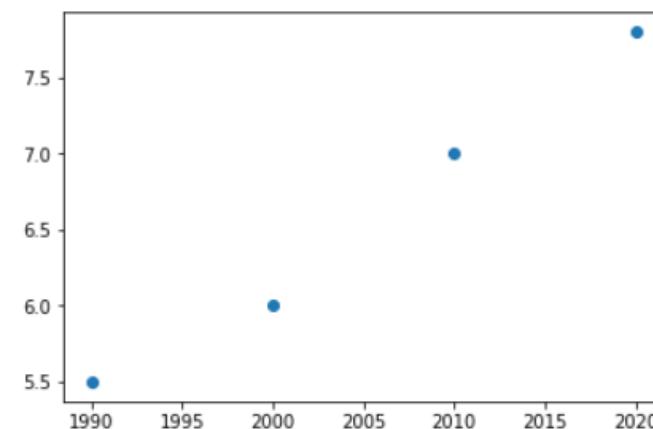
```
In [1]: import matplotlib.pyplot as plt
```

```
In [2]: año = [1990, 2000, 2010, 2020]
pob = [5.5, 6, 7, 7.8]
```

```
In [5]: plt.scatter(año, pob)
```

```
Out[5]: <matplotlib.collections.PathCollection at 0x95d0a58>
```

```
In [6]:
```



# Visualización básica con Matplotlib

- ¿Cómo visualizamos variables de un dataframe de Pandas?

```
In [34]: import pandas as pd
df = pd.DataFrame(dic)
```

```
In [35]: df
```

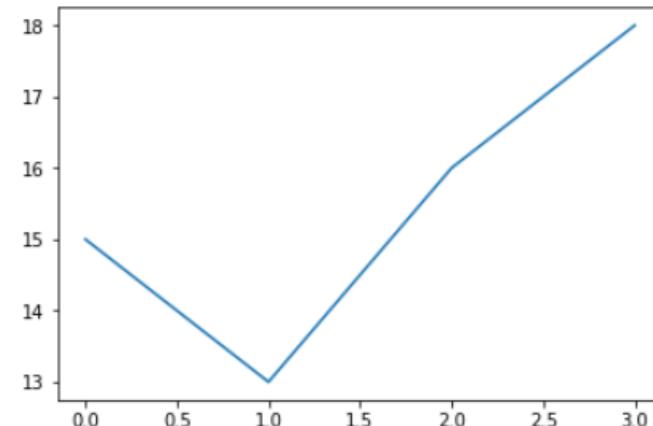
```
Out[35]:
   hum  temp
0    60    15
1    80    13
2    45    16
3    55    18
```

```
In [36]: plt.plot(df["temp"])
```

```
Out[36]: [

```

```
In [37]: plt.show()
```

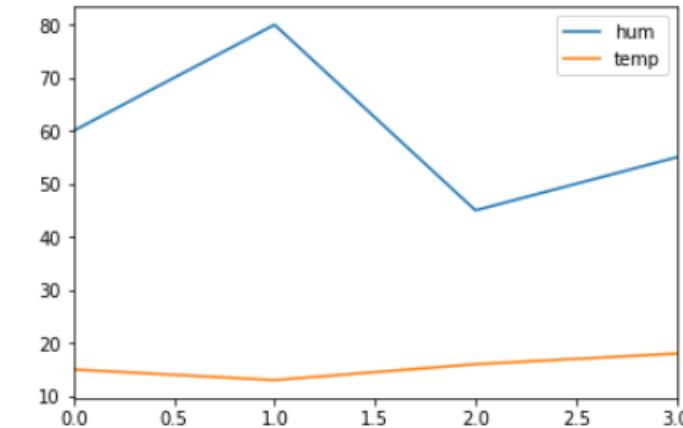


Df["nombre\_columna"].plot()

```
In [38]: df.plot()
```

```
Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0xc130438>
```

```
In [39]: plt.show()
```

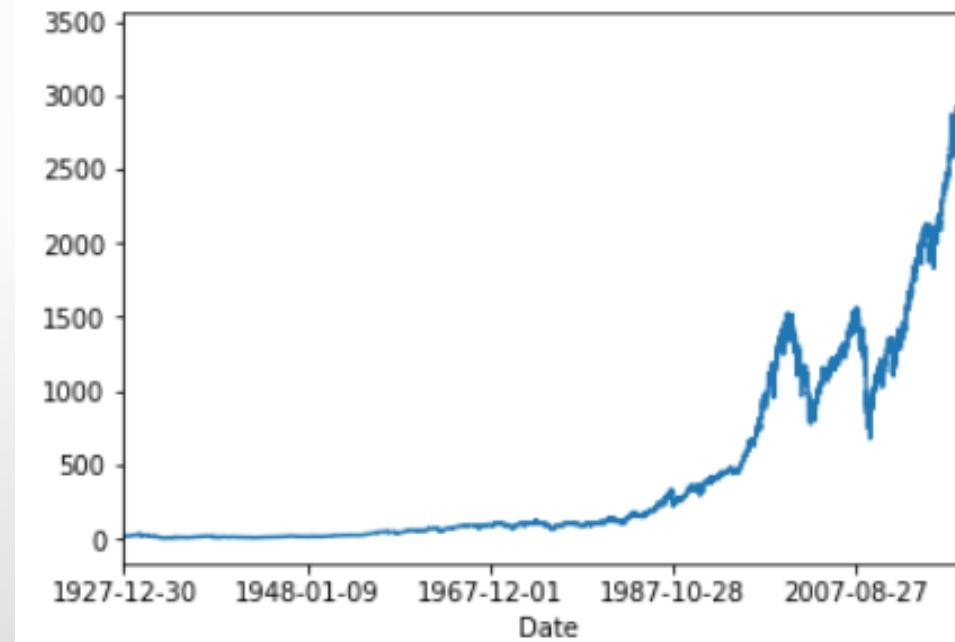


<https://matplotlib.org/gallery/index.html>



# Visualización básica con Matplotlib

- **EJERCICIO:** Visualizar la evolución de la cotización en bolsa del índice SP500 (delimitador csv = ",", si quiere poner como índice la columna de fecha: df.index = df[“Date”])



# Visualización básica con Matplotlib – Caso práctico

- **Ejercicio:** Visualizar un gráfico de Esperanza de Vida frente a Renta per capita:
  - ¿Existe una correlación entre estas variables?



# Flujograma de un proyecto Data Science

## 1. Importación de datos

Obtener datos desde fuentes heterogéneas.



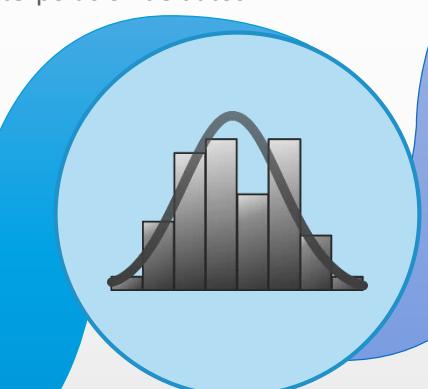
## 2. Limpieza de inconsistencias

Eliminar información errónea, redundante, transformación de tipos de datos.



## 3. Limpieza estadística y transformación

Resumen estadístico, búsqueda de outliers (boxplots) e interpolación de datos.



## 4. Visualización de datos

Mostrar la distribución de nuestra información con diferentes visualizaciones como histogramas, CDFs, correlaciones, series temporales,...



## 5. Análisis y conclusiones

Análisis de datos y generación de conclusiones para la toma de decisiones.



matplotlib



Seaborn





# BLOQUE 2:

## Fundamentos del lenguaje Python

# Variables en Python

- ¿Cómo definimos una variable?

```
In [1]: a = 5
```

```
In [2]: b = "Hola"
```

```
In [3]: c = 6.73
```

- ¿Cuáles son los tipos de datos disponibles en Python?

- Int → Números enteros
- Float → Números reales
- Str → Cadena de texto
- Bool → True / False

- ¿Es necesario definir el tipo de dato de una variable?
  - **NO es necesario**

- ¿Podemos crear nuevas variables a partir de otras previamente definidas?

```
In [7]: d = c**2
```

```
In [8]: d
```

```
Out[8]: 45.2929
```

- **EJERCICIO:** Calcular el Índice de Masa Corporal a partir de 2 variables (altura y peso).

$$IMC = \frac{peso \ [kg]}{altura[m]^2}$$

```
In [4]: type(a)
```

```
Out[4]: int
```

```
In [5]: type(b)
```

```
Out[5]: str
```

```
In [6]: type(c)
```

```
Out[6]: float
```

```
In [9]: altura = 1.75
```

```
In [10]: peso = 70
```

```
In [11]: IMC = peso / altura**2
```

```
In [12]: IMC
```

```
Out[12]: 22.857142857142858
```

# Creación de listas y extracción de datos

- Definición de listas [a, b, c]: Colección de valores de cualquier tipo de dato.

- Creación**

```
In [9]: dimensiones = [14, 18, 22, 13, 27, "mesa"]
In [10]: dimensiones
Out[10]: [14, 18, 22, 13, 27, 'mesa']
```

- Selección dato**

```
In [2]: dimensiones
Out[2]: [14, 18, 22, 13, 27, 'mesa']
```

Posición: 0, 1, 2, 3, 4, 5

Posición: -6, -5, -4, -3, -2, -1

```
In [6]: dimensiones[4]
```

```
Out[6]: 27
```

```
In [7]: dimensiones[-2]
```

```
Out[7]: 27
```

- Slice [inicio (incluido) : final (excluido)]**

```
In [10]: dimensiones
Out[10]: [14, 18, 22, 13, 27, 'mesa']

In [13]: dimensiones[1:3]
Out[13]: [18, 22]
```

- EJERCICIO:** Seleccionar en la lista dimensiones desde 2º valor hasta el penúltimo

```
In [5]: dimensiones[1:-1]
Out[5]: [18, 22, 13, 27]
```

# Conceptos avanzados de creación de listas

- ¿Cómo modificar elementos en una lista?

```
In [1]: temp = [35.6, 36.2, 35.7, 35.9, 36.3, 36.5]
```

```
In [2]: temp[1] = 35.9
```

```
In [3]: temp
```

```
Out[3]: [35.6, 35.9, 35.7, 35.9, 36.3, 36.5]
```

- ¿Cómo añadir elementos en una lista?

```
In [4]: temp = temp + [36.2, 35.8]
```

```
In [5]: temp
```

```
Out[5]: [35.6, 35.9, 35.7, 35.9, 36.3, 36.5, 36.2, 35.8]
```

- ¿Cómo eliminar elementos en una lista?

```
In [6]: del(temp[2])
```

```
In [7]: temp
```

```
Out[7]: [35.6, 35.9, 35.9, 36.3, 36.5, 36.2, 35.8]
```

- ¿Podemos crear nuevas listas en base a otras?

- **MODO 1:** Crear una lista con un puntero

```
In [8]: x = [2, 5, 6]
```

```
In [9]: y = x
```

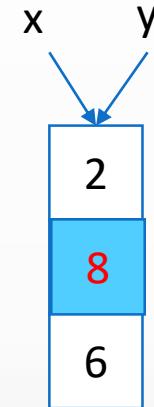
```
In [10]: y[1] = 8
```

```
In [11]: x
```

```
Out[11]: [2, 8, 6]
```

```
In [12]: y
```

```
Out[12]: [2, 8, 6]
```



- **MODO 2:** Crear una lista independiente

```
In [18]: x = [2, 5, 6]
```

```
In [19]: y = list(x)
```

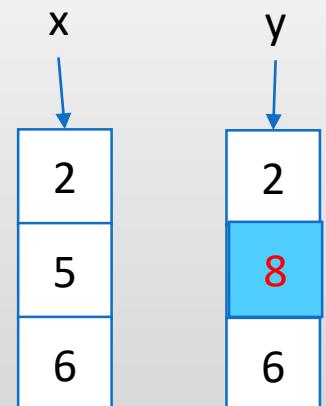
```
In [22]: y[1] = 8
```

```
In [23]: x
```

```
Out[23]: [2, 5, 6]
```

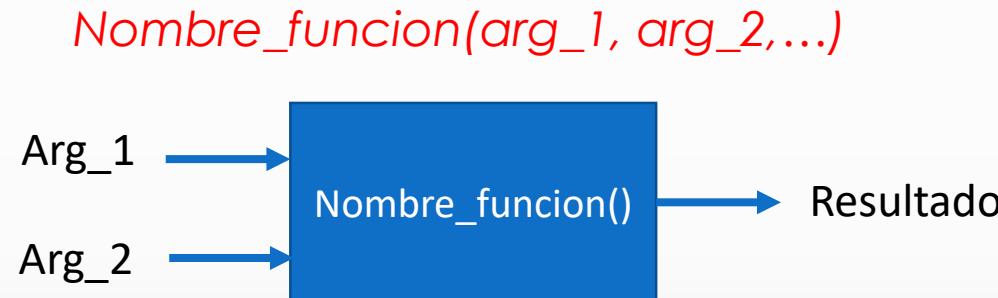
```
In [24]: y
```

```
Out[24]: [2, 8, 6]
```



# Uso de funciones en Python (in-built)

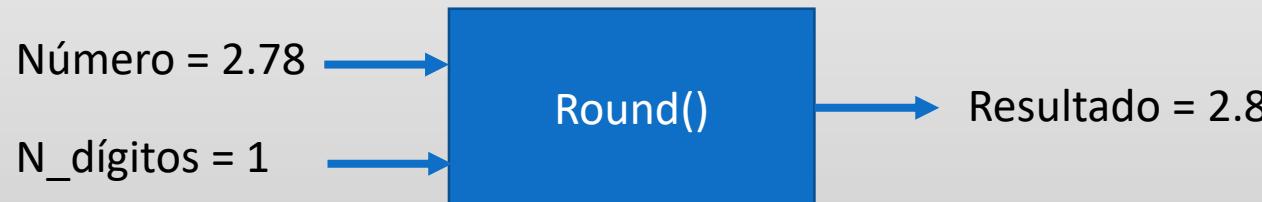
- Función:** Código reusable que resuelve una tarea particular y nos evita escribir código.
- ¿Cómo utilizamos una función?**
- ¿Qué funciones built-in existen en Python?**



- Ejemplo Función built-in de Python**

*Round(número, N\_dígitos)*

*Round(2.78, 1)*



Built-in Functions				
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

<https://docs.python.org/3/library/functions.html>

# Creación de funciones en Python y argumentos flexibles

- **Sintaxis**

```
Def nombre_funcion(arg_1, arg_2,...):
    """ docstrings """
    código_función
    return valor
```

- **Ejemplo**

```
def potencia(valor_base, valor_exponente):
    "Función que eleva valor_base a la potencia dada en valor_exponente"
    resultado = valor_base ** valor_exponente
    return resultado
```

Arguments

Lógica

docstring

- **¿Cómo invocamos a la función?**

In [5]: `potencia(2,4)`

Out[5]: 16

- **¿Podemos crear argumentos por defecto?**

```
def potencia(valor_base, valor_exponente=2):
```

`potencia(2)`

4

- **¿Es posible tener argumentos flexibles?**

Ejemplo: Multiplicar todos los números insertados

```
def multi(*args):
    mult_todos = 1

    for n in args:
        mult_todos = mult_todos * n

    return mult_todos
```

`multi(4,7,8)`

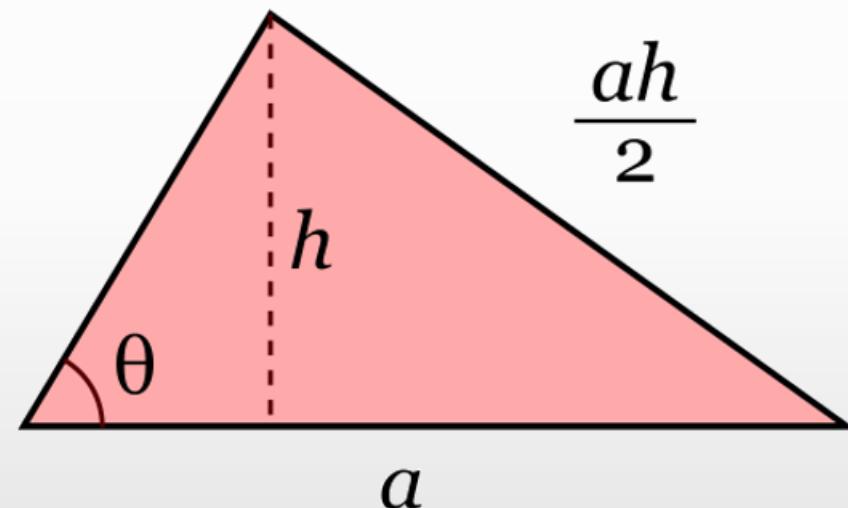
224

`multi(4,7,8,2,3)`

1344

# Creación de funciones en Python y argumentos flexibles

- **EJERCICIO:** Crear una función que calcule el área de un triángulo dados como argumentos la base y la altura. Comprueba que el resultado es correcto para un triángulo de base = 5 y altura = 3.



```
def area_tri(base, altura):
    """Función para calcular el área de un triángulo"""
    area = (base * altura)/2
    return area
```

```
area_tri(5,3)
```

7.5

# Funciones lambda

- **Sintaxis**

Nombre\_función = lambda arg\_1, arg\_2,...: código\_función

- **Ejemplo (Elevar un número a una potencia)**

```
potencia = lambda base, exponente: base**exponente
```



```
potencia(2,3)  
8
```

- **Función map:** map(func, secuencia): Aplica la función a todos los elementos de una secuencia (una lista, una columna de un dataframe,...)
- **Ejemplo (Elevar una lista de números al cuadrado)**

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



```
print(list(lista_al_cuadrado))  
[16, 25, 4]
```

- **EJERCICIO:** Crear función lambda que convierta de \$ a € una lista de precios en \$ (asumir que 1€ = 1,1\$)

```
precios_dol = [10.82, 11.95, 14.49, 24.95]
```



```
precios_eur = map(lambda x:x*1.1, precios_dol)  
print(list(precios_eur))  
[9.836363636363636, 10.863636363636362, 13.172727272727272, 22.68181818181818]
```

```
lista_al_cuadrado = map(lambda x:x**2,lista)
```

# Métodos en Python

- **Método:** Funciones que pertenecen a un tipo de objeto.

- **Sintaxis**

*Nombre\_objeto.nombre\_método()*

Tipo Objeto	Ejemplo de método
<pre>In [1]: nombre = "susana"</pre>	<p><i>str</i></p> <p>Capitalize() Replace(<i>cad_ini</i>, <i>cad_fin</i>)</p>
<pre>In [4]: alturas = [1.8, 1.75, 1.9, 1.85]</pre>	<p><i>list</i></p> <p>Insert(<i>pos,valor</i>)</p>

\* Una librería contiene multiples funciones y métodos que podemos reutilizar.



# Cómo crear diccionarios en Python

- Diccionario:** Objeto que nos proporciona una listado **<clave> : <valor>**
- Sintaxis:** `Nombre_dic = {"clave_1": valor_1, "clave_2": valor_2, ...}`
- Ejemplo (Diccionario con las notas de cada alumno)**

```
In [1]: alumnos_dic = {"Raúl": 9, "María": 7.5, "Rubén": 8, "Verónica": 9.5, "Diego": 6, "Esteban": 7}
```

- ¿Qué nos aporta frente a una lista?**

## Listas

```
alumnos
['Raúl', 'María', 'Rubén', 'Verónica', 'Diego', 'Esteban']

notas_alumnos
[9, 7.5, 8, 9.5, 6, 7]
```



¿Cómo obtenemos la nota de Rubén?

```
In [24]: ind_rub = alumnos.index("Rubén")
In [25]: ind_rub
Out[25]: 2
In [26]: notas_alumnos[ind_rub]
Out[26]: 8
```

Muy poco intuitivo y dependiente del orden (indexación por número)

## Diccionario

```
alumnos_dic
{'Raúl': 9,
'María': 7.5,
'Rubén': 8,
'Verónica': 9.5,
'Diego': 6,
'Esteban': 7}
```



```
alumnos_dic["Rubén"]
8
```

clave  
valor

Fácil de usar e indexación por claves

# Uso de función zip para creación de diccionarios en base a listas

- ¿Podemos crear un diccionario a partir de 2 listas? → Sí, con la función “zip”

```
alumnos
['Raúl', 'María', 'Rubén', 'Verónica', 'Diego', 'Esteban']

notas_alumnos
[9, 7.5, 8, 9.5, 6, 7]
```



```
dic_alumnos = dict(zip(alumnos, notas_alumnos))

dic_alumnos
{'Raúl': 9,
 'María': 7.5,
 'Rubén': 8,
 'Verónica': 9.5,
 'Diego': 6,
 'Esteban': 7}
```

- **EJERCICIO:** Crear 2 listas, una de nombre de país y otra de población. A partir de estas listas, crear el diccionario “**dic\_pais**” y mostrar el valor de “**México**”.

```
país = ["España", "México", "Venezuela", "Colombia", "Perú", "Argentina"]

poblacion = [47, 128, 32, 50, 33, 45]
```

```
dic_pais = dict(zip(país, poblacion))

dic_pais["México"]
128
```

# Operadores en Python

## COMPARADORES

<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que
==	Igual que
!=	No igual que

In [1]: `x = 3`

In [4]: `x > 5`  
Out[4]: `False`

## BOOLEANOS

and	Solo retorna Verdadero si ambas condiciones son verdaderas
or	Retorna Verdadero si alguna o ambas de las condiciones son verdaderas
not	Retorna Verdadero si la condición es falsa y viceversa

In [5]: `x = 3`

In [6]: `y = 5`

Falso Verdadero  
In [7]: `x > 4 and y > 4`  
Out[7]: `False`

Falso Verdadero  
In [8]: `x > 4 or y > 4`  
Out[8]: `True`

# Bucles en Python

## IF

- Sintaxis:**

```
if condición_1:  
    expresión_1  
elif condición_2:  
    expresión_2  
else:  
    expresión_3
```

Se evalúa expresión\_1 si condición\_1 es Verdadera

Se evalúa expresión\_2 si condición\_1 es Falsa y condición\_2 Verdadera

Se evalúa expresión\_3 si condición\_1 y condición\_2 son Falsas

- Ejemplo (Cálculo de áreas en función del tipo de objeto geométrico)**

```
In [22]: objeto = "cuadrado"  
  
In [23]: radio = 5  
  
In [24]: lado = 3  
  
In [25]: if objeto == "cuadrado":  
    area = lado ** 2  
    print("El área del cuadrado es: " + str(area))  
elif objeto == "círculo":  
    area = 3.14 * radio ** 2  
    print("El área del círculo es: " + str(area))  
else:  
    print("No se puede calcular el área del objeto indicado")  
  
El área del cuadrado es: 9
```

## FOR

- Sintaxis:**

```
for var in sec:  
    expresión
```

Se evalúa la expresión para cada “var” del objeto “sec”

- Ejemplo (Iterarar lista de longitudes)**

```
In [26]: longitudes = [12, 9, 14]
```

```
In [28]: for i in longitudes:  
    print("Longitud de la mesa " + str(i))
```

```
Longitud de la mesa 12  
Longitud de la mesa 9  
Longitud de la mesa 14
```

```
In [31]: for ind, it in enumerate(longitudes):  
    print("Longitud de la mesa " + str(ind+1) + " " + str(it))
```

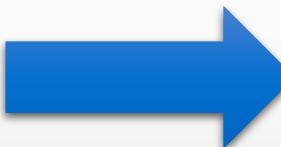
```
Longitud de la mesa 1 12  
Longitud de la mesa 2 9  
Longitud de la mesa 3 14
```

# Compreensión de listas en python

- **Objetivo:** Sintetizar un bucle for en una línea de código.
- **¿Cómo podemos realizar un cálculo a cada elemento de una lista? (Ejemplo sumar 5 a cada valor de una lista)**

## FOR

```
In [9]: numeros = [3, 5, 7, 2, 8]
In [10]: numeros_sum_5 = []
In [11]: for num in numeros:
           numeros_sum_5.append(num+5)
In [12]: numeros_sum_5
Out[12]: [8, 10, 12, 7, 13]
```



## Compreensión de listas

```
In [14]: numeros_sum_5_cl = [num+5 for num in numeros]
In [15]: numeros_sum_5_cl
Out[15]: [8, 10, 12, 7, 13]
```

- **¿Podemos añadir condiciones? (Ejemplo sumar 5 solo si el número es menor que 5)**

```
In [19]: numeros_sum_5_cl_c = [num+5 if num<5 else num for num in numeros]
```

```
In [20]: numeros_sum_5_cl_c
Out[20]: [8, 5, 7, 7, 8]
```

- **EJERCICIO:** Crear lista que eleve al cuadrado los valores pares de la lista (condiciones para evaluar si es par: num % 2 == 0)

```
In [23]: numeros = [3, 6, 7, 2, 8]
In [24]: numeros_cuadrado = [num**2 if num%2==0 else num for num in numeros]
```



# BLOQUE 3:

## Conceptos de Estadística y Análisis de Datos

# Variables y Conceptos básicos

**Variable:** Representa una cantidad que cambia su valor.

**Variable Discreta:** Es una variable cuyo valor se obtiene contando, por tanto tiene un número discreto de valores.

**Ejemplo:** *Número de estudiantes de las clases [15, 18, 20, 19]*

- **Media**

$$\bar{x} = \frac{\sum_{i=1}^n x_i}{n}$$

**Variable\_1:** 4, 6, 3, 5, 8, 9, 2, 12, 16, 4, 7, 42, 13, 6, 7

Media: 9,6

- **Mediana**

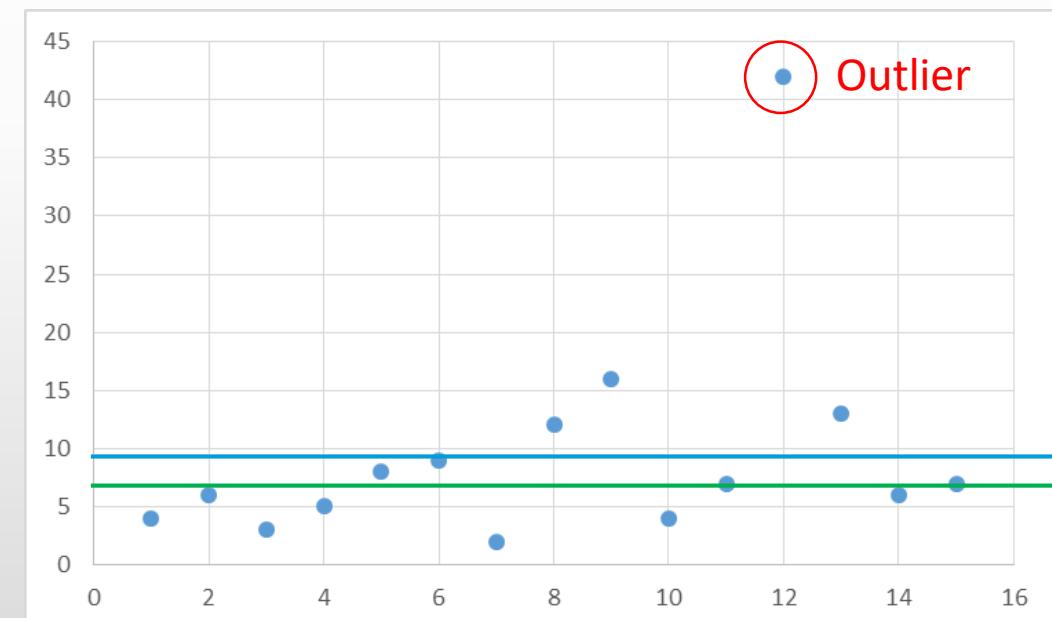
Representa el valor de la variable en la posición central en un conjunto de datos ordenados.

**Variable\_1:** 2, 3, 4, 4, 5, 6, 6, 7, 7, 8, 9, 12, 13, 16, 42

Mediana: 7

**Variable Continua:** Es una variable cuyo valor se obtiene midiendo.

**Ejemplo:** *Temperatura en Madrid [29.58, 29.43, 29.27, 28.81]*



# Variables y Conceptos básicos

- ¿Qué es más importante, la media o la mediana?

Variable\_1: 4, 6, 3, 5, 8, 9, 2, 12, 16, 4, 7, **242**, 13, 6, 7

• Media

Media: 22,9

• Mediana

Mediana: 7

Variable\_2: 21, 23, 25, 20, 21, 19, 24, 26, 20, 27, 25, 24, 23, 22, 24

Media: 22,9

Mediana: 23

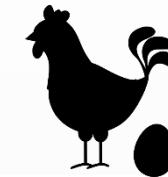
- El valor medio es un dato estadístico importante, pero habitualmente la mediana es más representativo de la distribución de la variable y evita outliers.

# Varianza de una variable

**Varianza:** Indica la dispersión de un conjunto de datos respecto a su valor medio:

- Si la varianza es alta, entonces los valores de la variable están más alejados del valor medio y viceversa

$$\sigma^2 = \frac{\sum(x - \bar{x})^2}{n}$$



- Ejemplo (Variable x: Peso huevos puestos por una gallina)

○ Huevo 1 = 60g	50
○ Huevo 2 = 56g	76
○ Huevo 3 = 61g	51
○ Huevo 4 = 68g	78
○ Huevo 5 = 51g	41
○ Huevo 6 = 53g	63
○ Huevo 7 = 69g	59
○ Huevo 8 = 54g	64

• Media

$$\bar{x} = \frac{\sum x}{n} = \frac{472}{8} = 59g$$

59g

Valor (x)	(x - $\bar{x}$ ) <sup>2</sup>
60	50
56	76
61	51
68	78
51	41
53	63
69	59
54	64
Total	320

$$\sigma^2 = \frac{\sum(x - \bar{x})^2}{n} = \frac{320}{8} = 40$$

140

• Desviación estándar

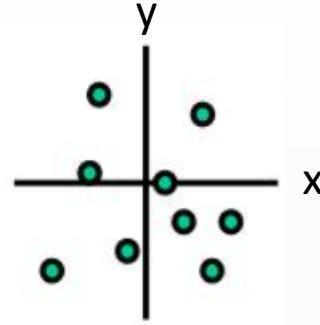
$$\sigma = \sqrt{\sigma^2} = \sqrt{40} = 6.32g$$

11,9g

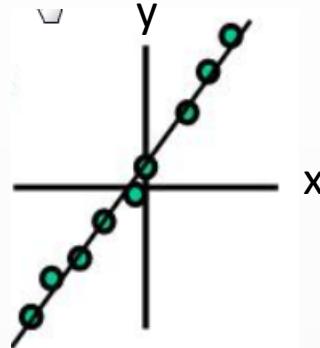
\*La varianza proporciona una medida de la volatilidad e incertidumbre de una variable (si todos los valores de la variable fueran iguales, el resultado de la varianza es 0, es decir, no hay incertidumbre).

# Correlación de variables

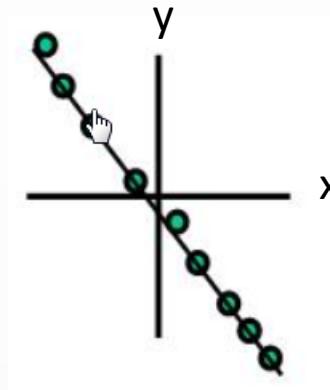
**Correlación:** Indica si existe relación entre 2 variables y la dirección de la relación.



Sin correlación  
( $r \sim 0$ )

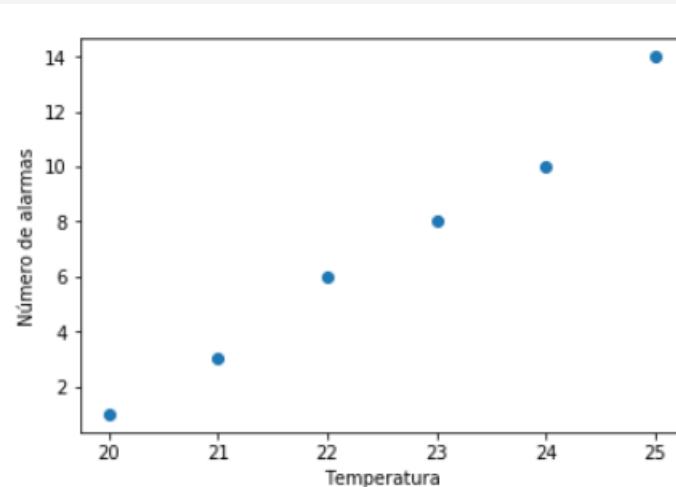


Correlación positiva  
( $r \sim 1$ )



Correlación negativa  
( $r \sim -1$ )

¿Existe relación entre la temperatura y el número de alarmas en el edificio?



Correlación positiva  
( $r = 0,85$ )

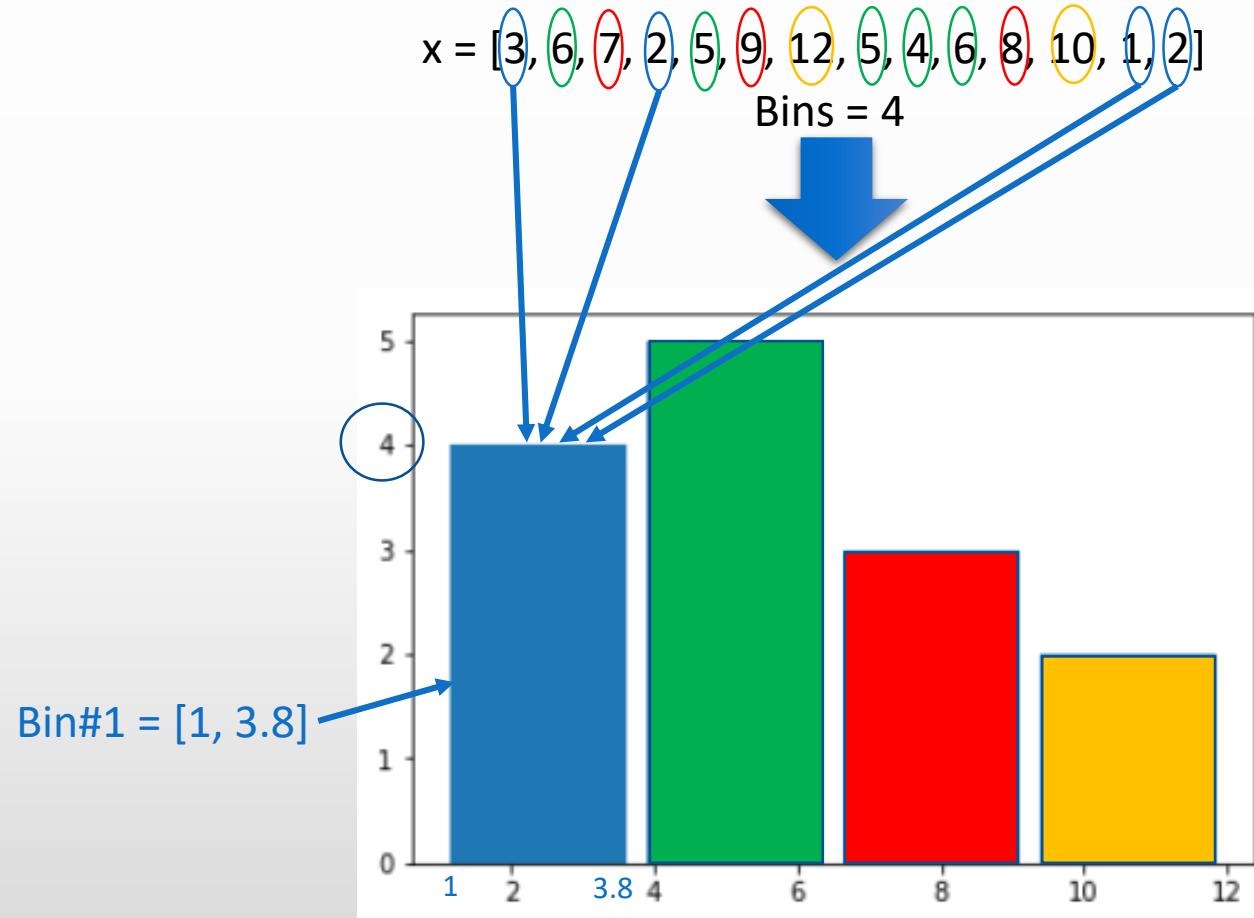
$$r = \frac{n \cdot \sum x_i \cdot y_i - \sum x_i \cdot \sum y_i}{\sqrt{[n \cdot \sum x_i^2 - (\sum x_i)^2] \cdot [n \cdot \sum y_i^2 - (\sum y_i)^2]}}$$

$-1 \leq r \leq 1$

# Histogramas

**Histograma:** Agrupación de los valores de una variables en **bins**. Proporciona la distribución de la variable.

x: Número de pedidos de mis clientes

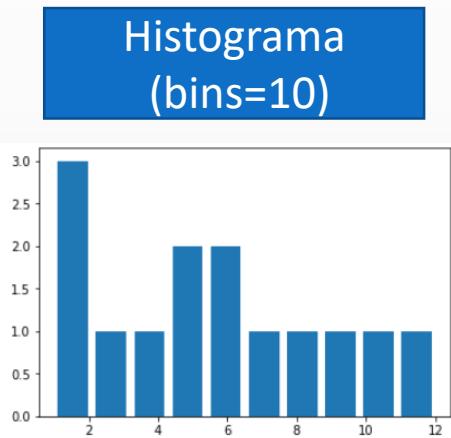


# Análisis con percentiles (CDF)

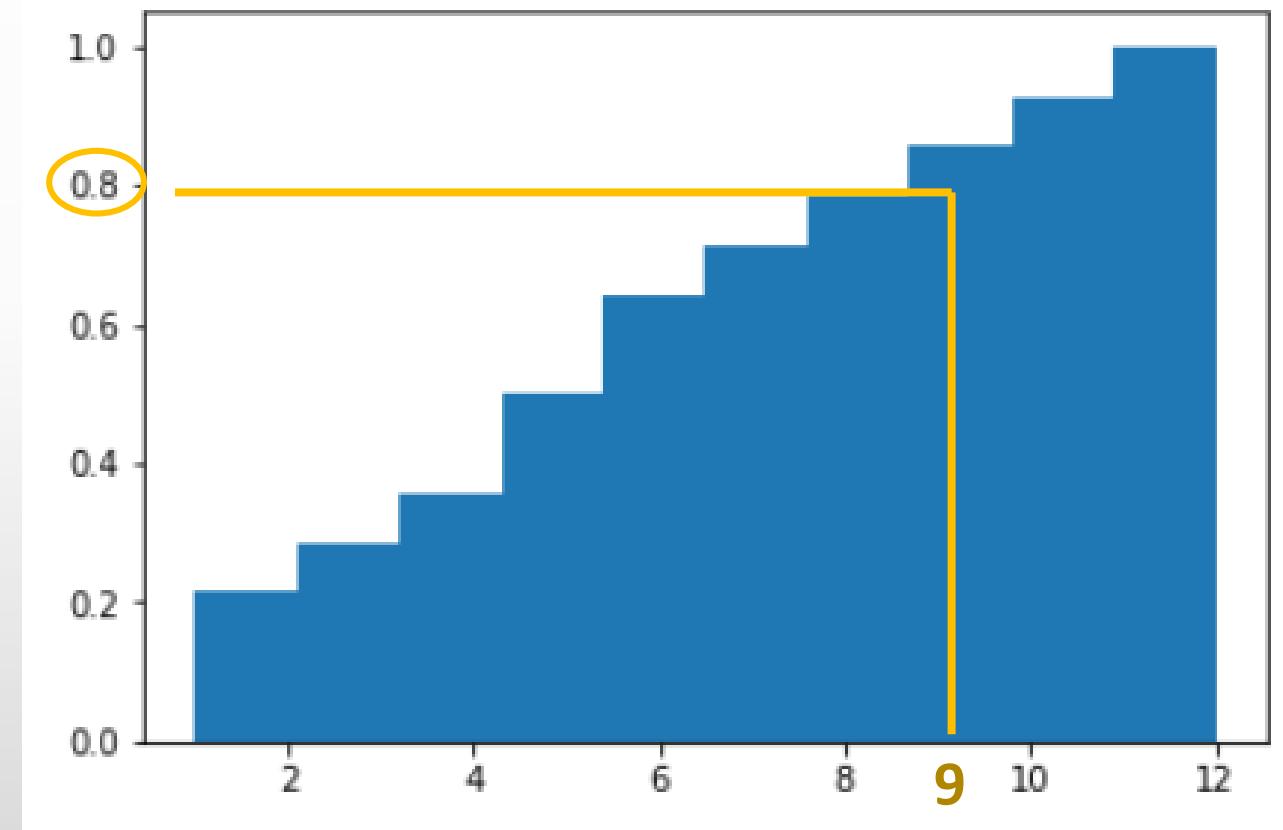
**Percentil x%:** Indica el valor que NO se supera un x%. Responde preguntas como “*¿Qué porcentaje de mis clientes realizan un pedido NO superior a un valor de unidades en el x% de las ocasiones?*”

x: Número de pedidos de mis clientes

x = [3, 6, 7, 2, 5, 9, 12, 5, 4, 6, 8, 10, 1, 2]



Percentil 80%  
Gráfico percentil  
(CDF) →  
Histograma  
acumulado y  
normalizado

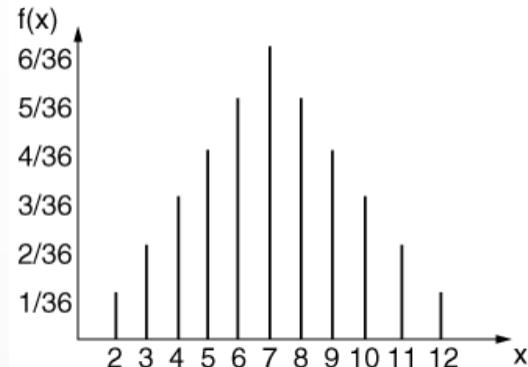


*“El 80% de los pedidos no superan las 9 unidades”*

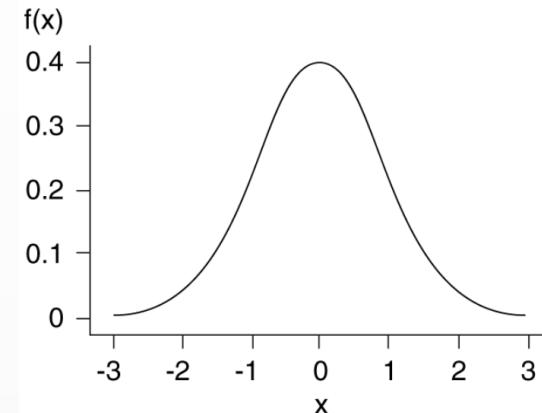
# Funciones densidad de probabilidad

**Función densidad probabilidad:** Función que describe la probabilidad de obtener cada uno de los valores de una variable.

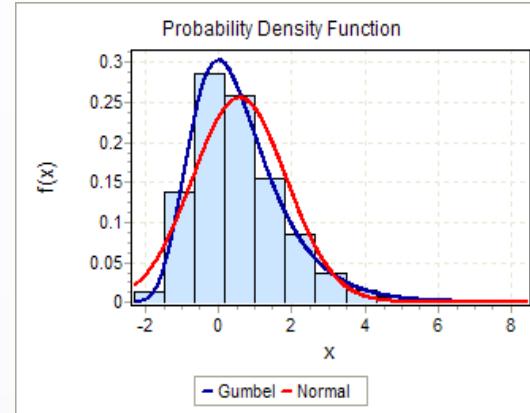
Variable Discreta



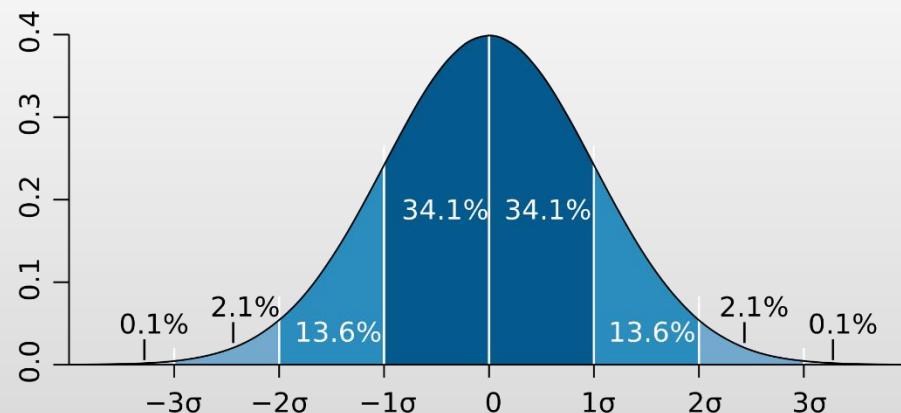
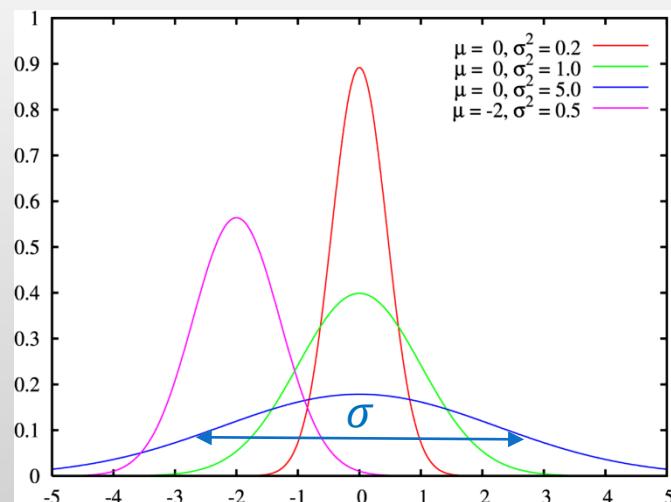
Variable Continua



Histograma → Distribución



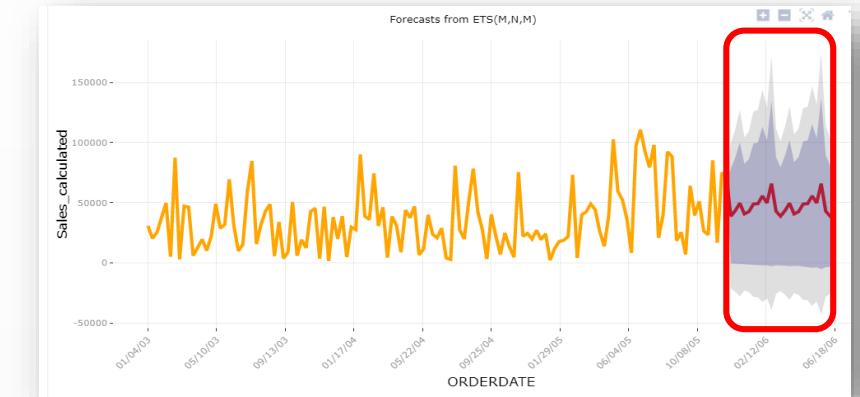
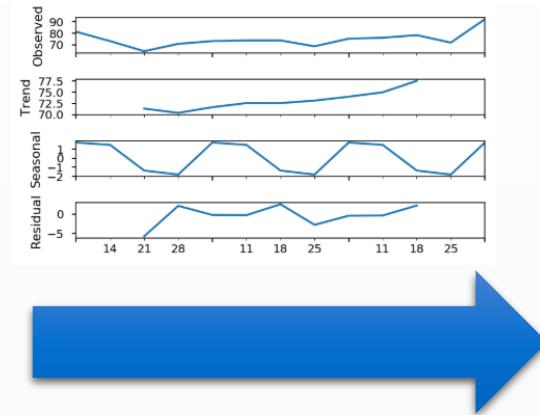
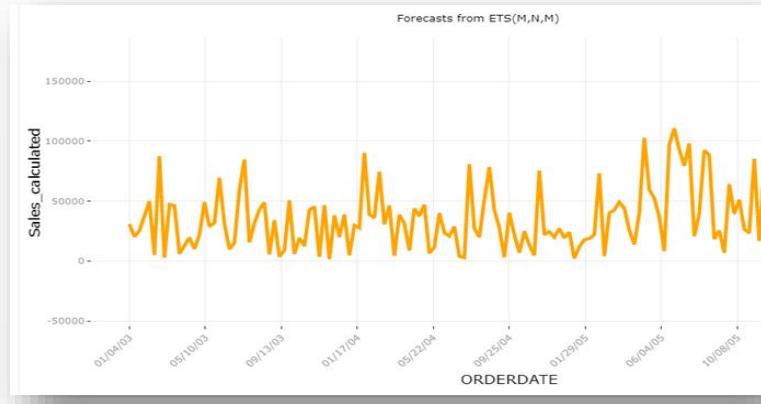
**Distribución normal (Gauss):** Función que permite modelar numerosos fenómenos naturales, sociales y psicológicos. Se caracteriza por el **valor medio y la varianza**.



- 68% de los datos se encuentran en el intervalo:  $\bar{x} - \sigma < x < \bar{x} + \sigma$
- 95% de los datos se encuentran en el intervalo:  $\bar{x} - 2\sigma < x < \bar{x} + 2\sigma$
- 99% de los datos se encuentran en el intervalo:  $\bar{x} - 3\sigma < x < \bar{x} + 3\sigma$

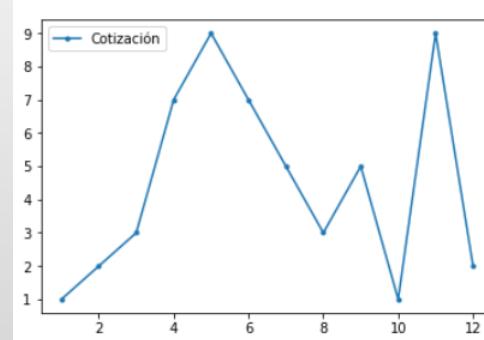
# Cálculo de previsiones (forecast) y media móvil

- Forecast:** Previsión del valor futuro de una variable en base a su información histórica (cuidado si hay eventos especiales)

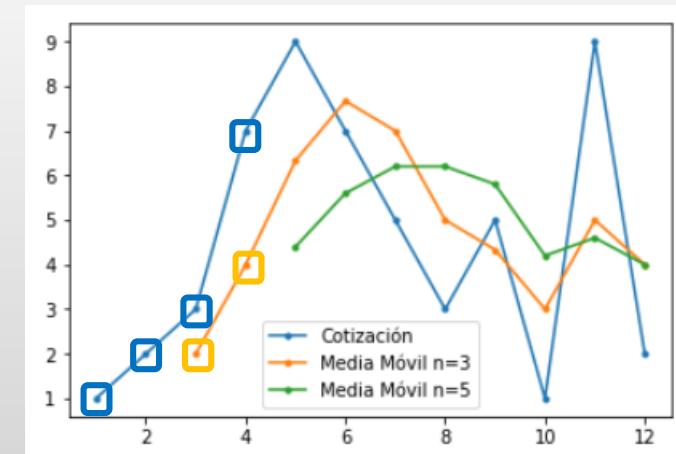
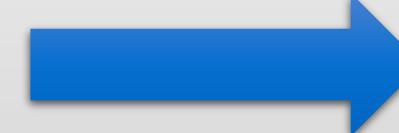


- Media Móvil (n):** Crea una serie de promedios de los últimos n puntos → Suaviza las fluctuaciones de plazos cortos y se centra en detectar tendencias a medio/largo plazo.

Valor\_cotización = [1, 2, 3, 7, 9, 7, 5, 3, 5, 1, 9, 2]



Media móvil n = 3  
Media móvil n = 5



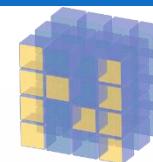
\*Se puede utilizar de manera ponderada para dar más peso a los valores más recientes.



# BLOQUE 4:

## Análisis numérico con Numpy

# Introducción a la librería NumPy



- Librería enfocada al cálculo numérico sencilla y rápida.
- Su objeto base es el Array NumPy (alternativa a la lista).
- Cálculos realizados sobre todo el Array.
- Solo pueden contener un tipo de dato (normalmente número) a diferencia de las listas.
- Es la base de cálculo para otras librerías como Pandas.
  
- **¿Por qué necesitamos NumPy?**

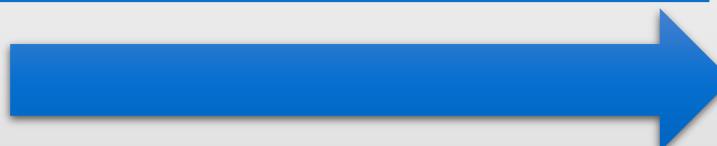
**Problema: Cálculos masivos con una lista  
(ejemplo IMC = peso / (altura<sup>2</sup>))**

```
In [1]: altura = [1.7, 1.65, 1.82]
peso = [67, 55, 72]
```

```
In [2]: peso / altura**2
```

```
-----  
TypeError                                 Traceback (most recent call last)
<ipython-input-2-d996fb66a542> in <module>
      1 peso / altura**2
----> 2 TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

**Solución: NumPy Array  
Cálculos a lo largo de todo el Array**



**Cálculo elemento a elemento**

```
In [10]: import numpy as np
```

```
In [11]: np_alt = np.array([1.7, 1.65, 1.82])
```

```
In [12]: np_peso = np.array([67, 55, 72])
```

```
In [13]: bmi = np_peso / np_alt**2
```

```
In [14]: bmi
```

```
Out[14]: array([23.183391 , 20.2020202 , 21.73650525])
```

# Selección de datos con array Numpy

- ¿Cómo seleccionamos datos dentro de un array NumPy?
- Slice [inicio (incluido) : final (excluido)]

```
In [4]: bmi
Out[4]: array([23.183391 , 20.2020202 , 21.73650525])
          Posición: 0           1           2
          Posición: -3          -2          -1
```



```
In [5]: bmi[1]
Out[5]: 20.202020202020204
```

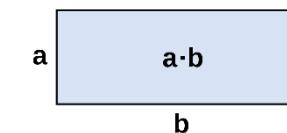
- ¿Podemos calcular si los valores cumplen una condición?
  - Ejemplo: Valores de un array superan el valor 21

```
In [6]: bmi>21
Out[6]: array([ True, False, True])
```

- ¿Podríamos retornar solo los valores que cumplen una condición?

```
In [7]: bmi[bmi>21]
Out[7]: array([23.183391 , 21.73650525])
```

- **EJERCICIO:** Calcular masivamente el área de 5 rectángulos y devolver únicamente aquéllos con área > 10:



```
bases_rec = np.array([5,2,4,7,8])
alturas_rec = np.array([3,4,1,4,3])
```

```
In [1]: import numpy as np
In [4]: bases_rec = np.array([5,2,4,7,8])
In [5]: alturas_rec = np.array([3,4,1,4,3])
In [6]: area_rec = bases_rec * alturas_rec
In [7]: area_rec[area_rec > 10]
Out[7]: array([15, 28, 24])
```

# Arrays 2D en Numpy

- ¿Cómo podemos crear un Array 2D de Numpy?

- Sintaxis

Nombre\_array = np.array([[valores fila 1], [valores fila 2],...])

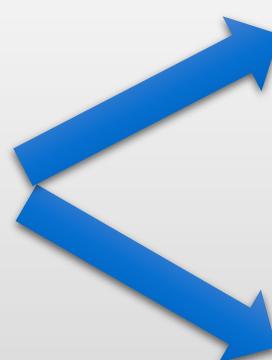
$$\begin{pmatrix} 2 & 7 & 8 \\ 4 & 8 & 10 \end{pmatrix}$$

```
In [1]: import numpy as np
In [3]: np_2d = np.array([[2,7,8],[4,8,10]])
In [4]: np_2d
Out[4]: array([[ 2,  7,  8],
               [ 4,  8, 10]])
```

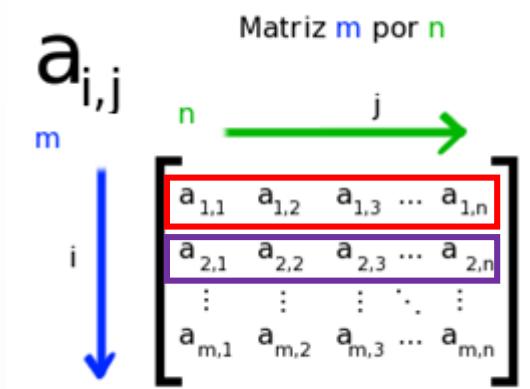
- ¿Cómo seleccionamos datos en un Array 2D?

```
In [4]: np_2d
Out[4]: array([[ 2,  7,  8],
               [ 4,  8, 10]])
      0   1   2
      |   |   |
      0   1   2
```

Seleccionar el valor 10 de la matriz



Seleccionar todas las filas pero solo la 1<sup>a</sup> y 2<sup>a</sup> columna



```
In [5]: np_2d[1,2]
Out[5]: 10
```

Ind\_fil, ind\_col

```
In [10]: np_2d[:,0:2]
Out[10]: array([[ 2,  7],
                [ 4,  8]])
```

# Cálculo estadístico con NumPy

- Funciones estadísticas de NumPy

<https://docs.scipy.org/doc/numpy-1.15.1/reference/routines.statistics.html>



```
In [1]: import numpy as np
In [2]: temp = np.array([12, 13.5, 13, 14, 13.2, 14.8, 15, 15.6, 16, 16.2, 15.7, 17, 17.2, 16.8, 14, 14.2, 14.7, 16, 17.5,
```

Cálculo estadístico	Función Numpy	Ejemplo
Media	mean(array)	<pre>In [13]: np.mean(temp) Out[13]: 16.106666666666667</pre>
Mediana	median(array)	<pre>In [5]: np.median(temp) Out[5]: 16.5</pre>
Mínimo / Máximo	min(array) / max(array)	<div style="display: flex; justify-content: space-around;"> <div> <pre>In [7]: np.min(temp) Out[7]: 12.0</pre> </div> <div> <pre>In [6]: np.max(temp) Out[6]: 19.0</pre> </div> </div>
Varianza / Desviación estándar	var(array) / std(array)	<div style="display: flex; justify-content: space-around;"> <div> <pre>In [8]: np.var(temp) Out[8]: 3.480622222222223</pre> </div> <div> <pre>In [9]: np.std(temp) Out[9]: 1.8656425762246698</pre> </div> </div>
Percentil (k)	percentile(array, k)	<pre>In [12]: np.percentile(temp, 90) Out[12]: 18.5</pre>

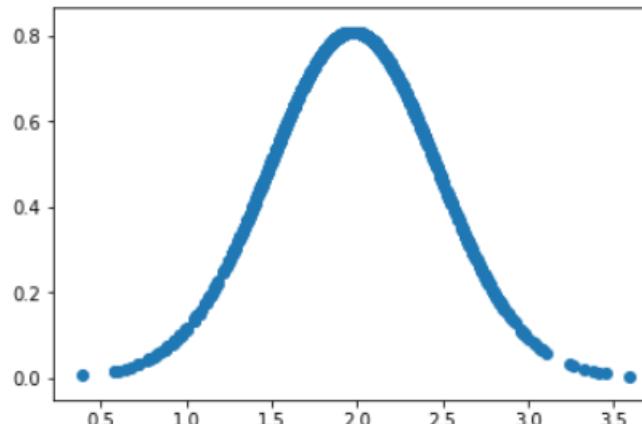
# Cálculo estadístico con NumPy

- ¿Podemos generar datos con Numpy tomando como partida parámetros estadísticos?

- **Sintaxis**      `Nombre_array = np.random.normal(media, desviación_estándar, número_muestras)`

In [26]: `array_gaus=np.random.normal(2, 0.5, 1000)`

```
In [27]: array_gaus
Out[27]: array([2.19531837, 1.54893028, 1.84174814, 2.71252481, 1.78945017,
 1.16268791, 2.46759922, 2.41621972, 1.28684017, 2.23130793,
 2.09491099, 2.46795843, 1.96403028, 2.02750671, 1.75079085,
 2.5899993 , 1.56008728, 1.164454 , 2.33810322, 1.89490481,
 1.88453102, 2.67632767, 1.84736897, 1.8542424 , 1.39079344,
 2.15690501, 1.73025284, 2.7754719 , 2.04859684, 1.94547824,
 0.87679857, 2.34229649, 2.1228844 , 1.26602608, 2.20146826,
 2.48883442, 2.85425271, 2.71963614, 2.24809112, 2.81917752,
 1.87916421, 1.7405027 , 1.54689144, 2.03739449, 2.75952405,
 1.88569036, 1.84373247, 2.02509188, 1.75365256, 2.24203324,
 2.10704141, 1.92574762, 1.38780252, 2.39752816, 2.10716858,
 2.13182693, 2.33117295, 2.16443958, 1.67299997, 1.66932945,
 1.6492649 , 2.58752937, 1.91900059, 1.93346217, 2.12764473,
 1.70315536, 1.33809121, 2.1560502 , 2.43501958, 2.22134755,
 1.78431443, 1.65916513, 1.61913641, 1.4010404 , 0.94603863,
 1.54680675, 3.07668221, 2.23106935, 1.76364306, 2.26277779,
 1.91418254, 2.00371678, 2.54976657, 1.36856749, 1.99605979,
 1.87989039, 1.14796156, 1.36898177, 1.72329343, 1.63086428,
 1.94021016, 1.58267271, 2.01643331, 2.03488019, 1.91802927,
```



- **EJERCICIO:** Crear distribución gaussiana de 10.000 puntos con media 3 y desviación estándar 1.5. Obtener el rango en el que se encuentran el 95% de nuestros valores ( $\bar{x} - 2\sigma < x < \bar{x} + 2\sigma$ )

## Ejercicio

```
In [36]: array_gaus_2=np.random.normal(3, 1.5, 1000)
In [38]: media = np.mean(array_gaus_2)
In [39]: desv_est = np.std(array_gaus_2)
In [40]: rango_95 = [media-2*desv_est, media+2*desv_est]
In [41]: rango_95
Out[41]: [-0.03748381813514223, 6.015381950484478]
```



# BLOQUE 5:

## Análisis de datos con **Pandas**



# Introducción a la librería Pandas ¿qué es un dataframe?

Pandas

- Construida sobre Numpy
- Permite manipular gran cantidad de datos de fuentes heterogéneas (BigData)
- Elemento clave: Dataframe

**Etiquetas de filas (índice)**

**Etiquetas de columnas**

	Pais	Poblacion	Area
0	Mexico	129	1973.0
1	Espana	46	505.0
2	Venezuela	32	916.0

Tipo: str int float

- Columna → Variable
- Filas → Observaciones

- 2D Numpy array solo un tipo de datos.
- Convención: `import pandas as pd`

# Creación de un dataframe a partir de un diccionario

Objetivo: Dataframe país-capital-superficie

	país	capital	superficie (Mkm2)
0	Brasil	Brasilia	8.5
1	México	Ciudad de México	1.9
2	España	Madrid	1.2
3	Perú	Lima	0.5
4	Colombia	Bogotá	1.1
5	Venezuela	Caracas	0.9

Definición diccionario

Nombre\_dic = {"clave\_1": valor\_1, "clave\_2": valor\_2, ...}

```
import pandas as pd
```

```
info_pais = {"país": ["Brasil", "México", "España", "Perú", "Colombia", "Venezuela"],
             "capital": ["Brasilia", "Ciudad de México", "Madrid", "Lima", "Bogotá", "Caracas"],
             "superficie (Mkm2)": [8.5, 1.9, 1.2, 0.5, 1.1, 0.9]}
```

claves → columnas      valor → datos columna a columna que  
forman la observación

```
df_pais = pd.DataFrame(info_pais)
```

\*Importante, la función es “DataFrame”,  
“dataframe” no es una función de Pandas.



# Cómo importar datos desde un fichero de texto plano (txt, csv,...)



1	FECHZ	HORA	IDENTIFICADOR	PEATONES	NÚMERO	DISTRITO	DISTRITO	NOMBRE_VIA	NÚMERO	CÓDIGO_POSTAL	OBSERVACIONES	DIRECCION	LATITUD	LONGITUD
2	01/01/2020	0:00:00	PERM_PEA02_PM01	1497	1	Centro	Calle Fuencarral	22,28004	Calle peatonal Datos en prueba	40,4220090	-3,7008917			
3	01/01/2020	0:15:00	PERM_PEA02_PM01	244	1	Centro	Calle Fuencarral	22,28004	Calle peatonal Datos en prueba	40,4220090	-3,7008917			
4	01/01/2020	0:30:00	PERM_PEA02_PM01	179	1	Centro	Calle Fuencarral	22,28004	Calle peatonal Datos en prueba	40,4220090	-3,7008917			
5	01/01/2020	0:45:00	PERM_PEA02_PM01	174	1	Centro	Calle Fuencarral	22,28004	Calle peatonal Datos en prueba	40,4220090	-3,7008917			
6	01/01/2020	1:00:00	PERM_PEA02_PM01	131	1	Centro	Calle Fuencarral	22,28004	Calle peatonal Datos en prueba	40,4220090	-3,7008917			
7	01/01/2020	1:15:00	PERM_PEA02_PM01	95	1	Centro	Calle Fuencarral	22,28004	Calle peatonal Datos en prueba	40,4220090	-3,7008917			
8	01/01/2020	1:30:00	PERM_PEA02_PM01	107	1	Centro	Calle Fuencarral	22,28004	Calle peatonal Datos en prueba	40,4220090	-3,7008917			
9	01/01/2020	1:45:00	PERM_PEA02_PM01	175	1	Centro	Calle Fuencarral	22,28004	Calle peatonal Datos en prueba	40,4220090	-3,7008917			
10	01/01/2020	2:00:00	PERM_PEA02_PM01	149	1	Centro	Calle Fuencarral	22,28004	Calle peatonal Datos en prueba	40,4220090	-3,7008917			
11	01/01/2020	2:15:00	PERM_PEA02_PM01	126	1	Centro	Calle Fuencarral	22,28004	Calle peatonal Datos en prueba	40,4220090	-3,7008917			
12	01/01/2020	2:30:00	PERM_PEA02_PM01	107	1	Centro	Calle Fuencarral	22,28004	Calle peatonal Datos en prueba	40,4220090	-3,7008917			
13	01/01/2020	2:45:00	PERM_PEA02_PM01	125	1	Centro	Calle Fuencarral	22,28004	Calle peatonal Datos en prueba	40,4220090	-3,7008917			
14	01/01/2020	3:00:00	PERM_PEA02_PM01	138	1	Centro	Calle Fuencarral	22,28004	Calle peatonal Datos en prueba	40,4220090	-3,7008917			
15	01/01/2020	3:15:00	PERM_PEA02_PM01	147	1	Centro	Calle Fuencarral	22,28004	Calle peatonal Datos en prueba	40,4220090	-3,7008917			
16	01/01/2020	3:30:00	PERM_PEA02_PM01	171	1	Centro	Calle Fuencarral	22,28004	Calle peatonal Datos en prueba	40,4220090	-3,7008917			

## Importar de un CSV

```
df = pd.read_csv(r'file.csv', delimiter=";", encoding = "ISO-8859-1", index_col = 0, nrows=5, header=None)
```

\*Argumentos obligatorios

\*Argumentos opcionales

Defecto: ";"

Defecto: None

Defecto: All

Defecto: 0



```
df_data = pd.read_csv('nombre_fichero.txt', delimiter=" ", header=None)
df_data.columns = ["a", "b", "c", "etc."]
```

- EJERCICIO: Cargar los primeros 10.000 registros del fichero CSV de tráfico de peatones (delimitador ";") indicando como índice del Dataframe la columna "IDENTIFICADOR" (2).

```
df = pd.read_csv(r'ruta_loca\PEATONES_2020.csv',
encoding = "ISO-8859-1", delimiter=";",
index_col=2, nrows=10000)
```

# Selección de datos en un dataframe Pandas

- ¿Cómo seleccionar columnas / filas?

- Método Convencional

- Columnas:

- 1 columna → df["nombre\_columna"]

```
In [6]: df_pais["capital"]
Out[6]: 0      Brasilia
        1  Ciudad de México
        2      Madrid
        3      Lima
        4     Bogotá
        5    Caracas
Name: capital  dtype: object
```

```
In [13]: type(df_pais["capital"])
Out[13]: pandas.core.series.Series
```

	país	capital	superficie (Mkm2)
0	Brasil	Brasilia	8.5
1	México	Ciudad de México	1.9
2	España	Madrid	1.2
3	Perú	Lima	0.5
4	Colombia	Bogotá	1.1
5	Venezuela	Caracas	0.9

- ¿Cómo haríamos si queremos seleccionar columnas y filas a la vez?

```
In [18]: df_pais[["capital", "superficie (Mkm2)"], 2:5]
```



TypeError: '(['capital', 'superficie (Mkm2)'), slice(2, 5, None))' is an invalid key



- Varias columnas → df[["nombre\_columna\_1", "nombre\_columna\_2"]]

```
In [10]: df_pais[["capital", "superficie (Mkm2)"]]
Out[10]:
   capital  superficie (Mkm2)
0  Brasilia          8.5
1  Ciudad de México      1.9
2      Madrid          1.2
3      Lima          0.5
4     Bogotá          1.1
5    Caracas          0.9
```

```
In [11]: df_pais[2:5]
Out[11]:
   país  capital  superficie (Mkm2)
2  España  Madrid          1.2
3    Perú    Lima          0.5
4  Colombia  Bogotá          1.1
```

- Filas:

- df[ind\_inicial : ind\_final(excluido)]

- Solución LOC / ILOC

# Selección de datos en un dataframe Pandas

- ¿Cómo seleccionar columnas / filas?

- LOC (basada en etiquetas) / ILOC (basado en índices de posición)

- Sintaxis LOC:

- df.loc[“lista\_nombres\_filas”, “lista\_nombres\_columnas”]

```
In [32]: df_pais.loc[['ES','CO'],['capital', "superficie (Mkm2)"]]
```

```
Out[32]:
```

	capital	superficie (Mkm2)
ES	Madrid	1.2
CO	Bogotá	1.1

- Sintaxis ILOC:

- df.iloc[“lista\_índices\_filas”, “lista\_índices\_columnas”]

```
In [34]: df_pais.iloc[[2,4],[1,2]]
```

```
Out[34]:
```

	capital	superficie (Mkm2)
ES	Madrid	1.2
CO	Bogotá	1.1

Posición:	0	país	1	capital	2	superficie (Mkm2)
0	BR	Brasil		Brasilia		8.5
1	ME	México	Ciudad de México			1.9
2	ES	España		Madrid		1.2
3	PE	Perú		Lima		0.5
4	CO	Colombia		Bogotá		1.1
5	VE	Venezuela		Caracas		0.9

Podemos indicar “:”:

```
In [33]: df_pais.loc[:,["capital", "superficie (Mkm2)"]]
```

```
Out[33]:
```

	capital	superficie (Mkm2)
BR	Brasilia	8.5
ME	Ciudad de México	1.9
ES	Madrid	1.2
PE	Lima	0.5
CO	Bogotá	1.1
VE	Caracas	0.9

Podemos indicar “:”:

```
In [35]: df_pais.iloc[:,[1,2]]
```

```
Out[35]:
```

	capital	superficie (Mkm2)
BR	Brasilia	8.5
ME	Ciudad de México	1.9
ES	Madrid	1.2
PE	Lima	0.5
CO	Bogotá	1.1
VE	Caracas	0.9

# Métodos útiles de un dataframe Pandas

## Información básica Dataframe

Nombre	Evaluación Matemáticas	Evaluación Lengua	
0 Javier	5.0	8	
1 Marta	7.0	9	
2 Lucía	8.0	7	
3 Pedro	4.0	5	
4 Juan	10.0	8	
5 María	NaN		6

Método / Atributo	Descripción	Ejemplo
Head(i)	Visualizar i registros del dataframe (defecto = 5)	In [61]: df_alumnos.head(10)
.shape	Devuelve el número de filas y columnas	In [62]: df_alumnos.shape Out[62]: (6, 3)
.index / .columns	Devuelve información del índice / columnas del dataframe	In [63]: df_alumnos.index Out[63]: RangeIndex(start=0, stop=6, step=1)  In [64]: df_alumnos.columns Out[64]: Index(['Nombre', 'Evaluación Matemáticas', 'Evaluación Lengua'], dtype='object')
Info()	Obtener información de todas las columnas del dataframe	In [65]: df_alumnos.info()  <class 'pandas.core.frame.DataFrame'> RangeIndex: 6 entries, 0 to 5 Data columns (total 3 columns): Nombre          6 non-null object Evaluación Matemáticas  5 non-null float64 Evaluación Lengua      6 non-null int64 dtypes: float64(1), int64(1), object(1) memory usage: 272.0+ bytes
count()	Devuelve el número de valores non-NA	In [66]: df_alumnos.count()  Out[66]: Nombre          6 Evaluación Matemáticas  5 Evaluación Lengua      6 dtype: int64

# Métodos útiles de un dataframe Pandas

## Información estadística Dataframe

Método	Descripción	Ejemplo
Sum()	Suma los valores de una columna	Df[“nombre_columna”].sum()
Mean() / Median()	Calcula el promedio / mediana de una columna	<pre>In [67]: df_alumnos[“Evaluación Matemáticas”].mean() Out[67]: 6.8</pre>
Quantile(k)	Calcula el percentile k de una columna [0,1]	<pre>In [68]: df_alumnos[“Evaluación Matemáticas”].quantile(0.8) Out[68]: 8.4</pre>
Min() / Max()	Obtener el mínimo y máximo de una columna	<pre>In [70]: df_alumnos[“Evaluación Matemáticas”].min() Out[70]: 4.0 In [71]: df_alumnos[“Evaluación Matemáticas”].max() Out[71]: 10.0</pre>
Describe()	Realiza un resumen estadístico para todas las columnas de tipo numérico	<pre>In [72]: df_alumnos.describe() Out[72]:           Evaluación Matemáticas  Evaluación Lengua count      5.000000        6.000000 mean       6.800000        7.166667 std        2.387467        1.471960 min        4.000000        5.000000 25%       5.000000        6.250000 50%       7.000000        7.500000 75%       8.000000        8.000000 max       10.000000       9.000000</pre>

# Métodos útiles de un dataframe Pandas

- **EJERCICIO:** A partir del dataframe con la información de tráfico peatonal adjunto, obtener la información básica del dataframe y la información estadística:
  - ¿Cuántos valores erróneos tiene la columna “PEATONES”?
  - ¿Cuál es la mediana de la variable “PEATONES”?
  - ¿Cuál es el rango de fechas de nuestros datos?
  - Crear una columna acumulativa del número de peatones (método cumsum)

```
In [29]: df.count()
Out[29]: FECHA          288755
          HORA            288755
          PEATONES        288752
          NÚMERO_DISTRITO  284872
          DISTRITO         284872
          NOMBRE_VIAL       284872
          NÚMERO           284872
          CÓDIGO_POSTAL     284872
          OBSERVACIONES_DIRECCION 284872
          LATITUD           284872
          LONGITUD          284872
          dtype: int64
```

```
In [32]: df["PEATONES"].median()
Out[32]: 373.0
```

```
In [34]: df["FECHA"].min()
Out[34]: '01/01/2020'
```

```
In [35]: df["FECHA"].max()
Out[35]: '31/05/2020'
```

```
In [37]: df["PEATONES_cum"] = df["PEATONES"].cumsum()
```

PEATONES_cum
497.0
741.0
920.0
1094.0
1225.0

# Eliminar duplicados, valores erróneos y columnas de un dataframe Pandas

- ¿Cómo limpiar información?

## Remover Duplicados

Duplicados en todas las columnas: df = df.drop\_duplicates()

*df\_prod*

	ID_Prod	Tipo producto	Color
0	A	Pantalón	Azul
1	B	Pantalón	Verde
2	C	Camisa	Rojo
3	C	Camisa	Morado
4	E	Zapato	Gris
5	A	Pantalón	Azul
6	Sin Catalogar	Zapato	Verde

Se queda guardado el resultado  
en el propio dataframe

In [15]: df\_prod.drop\_duplicates(inplace=True)

In [16]: df\_prod

Out[16]:

	ID_Prod	Tipo producto	Color
0	A	Pantalón	Azul
1	B	Pantalón	Verde
2	C	Camisa	Rojo
3	D	Camisa	Morado
4	E	Zapato	Gris
6	Sin Catalogar	Zapato	Verde

Duplicados en una columna: df = df.drop\_duplicates("nombre\_columna")

*df\_prod*

	ID_Prod	Tipo producto	Color
0	A	Pantalón	Azul
1	B	Pantalón	Verde
2	C	Camisa	Rojo
3	C	Camisa	Morado
4	E	Zapato	Gris
5	A	Pantalón	Azul
6	Sin Catalogar	Zapato	Verde

In [23]: df\_prod.drop\_duplicates("ID\_Prod", inplace=True)

In [24]: df\_prod

Out[24]:

	ID_Prod	Tipo producto	Color
0	A	Pantalón	Azul
1	B	Pantalón	Verde
2	C	Camisa	Rojo
4	E	Zapato	Gris
6	Sin Catalogar	Zapato	Verde

# Eliminar duplicados, valores erróneos y columnas de un dataframe Pandas

- ¿Cómo limpiar información?

## Eliminar nan:

```
df.dropna() / df["nombre_columna"].dropna()
```

**df\_prod2**

	ID_Prod	Tipo producto	Color
0	A	Pantalón	Azul
1	B	Pantalón	Verde
2	C	Camisa	Rojo
3	C	Camisa	Morado
4	E	Zapato	Gris
5	A	Pantalón	Azul
6	NaN	Zapato	Verde



In [29]: df\_prod2.dropna(inplace=True)



In [30]: df\_prod2

Out[30]:

	ID_Prod	Tipo producto	Color
0	A	Pantalón	Azul
1	B	Pantalón	Verde
2	C	Camisa	Rojo
3	C	Camisa	Morado
4	E	Zapato	Gris
5	A	Pantalón	Azul

## Eliminar columna

```
df.drop('nombre_columna', axis=1 , inplace=True)
```

In [30]: df\_prod2

Out[30]:

	ID_Prod	Tipo producto	Color
0	A	Pantalón	Azul
1	B	Pantalón	Verde
2	C	Camisa	Rojo
3	C	Camisa	Morado
4	E	Zapato	Gris
5	A	Pantalón	Azul



In [32]: df\_prod2.drop("Tipo producto",axis=1,inplace=True)

In [33]: df\_prod2

Out[33]:

	ID_Prod	Color
0	A	Azul
1	B	Verde
2	C	Rojo
3	C	Morado
4	E	Gris
5	A	Azul

Eliminar varias columnas: df.drop([lista\_columnas], axis=1)

# Eliminar duplicados, valores erróneos y columnas de un dataframe Pandas

- **EJERCICIO:** A partir del dataframe con la información de tráfico peatonal (PEATONES\_2020\_mod.csv), verificar el número de registros y a continuación eliminar duplicados definiendo duplicado como aquéllos registros que tienen los valores de todas las columnas iguales:
  - ¿Cuántos valores duplicados existían?
- Elimina todas los valores erróneos (NaN) del dataframe.
  - ¿Cuántos valores NaN existían?
  - ¿Qué porcentaje de los datos originales eran erróneos?

```
In [35]: df.shape  
Out[35]: (288755, 11)
```



```
In [36]: df2 = df.drop_duplicates()  
  
In [37]: df2.shape  
Out[37]: (288749, 11)
```



```
In [38]: df3 = df2.dropna()  
  
In [39]: df3.shape  
Out[39]: (284863, 11)
```

# Interpolación de datos

- ¿Qué es la **interpolar** datos?



Obtener un nuevo valor a partir del conjunto de datos

- ¿Para qué nos sirve?

- Si necesitamos mayor granularidad en nuestros datos
- Si existen valores erróneos y **NO** queremos descartar el registro.

- ¿Qué estrategias tenemos?

**fillna**

```
df["nombre_columna"].fillna(valor, inplace = True)
```

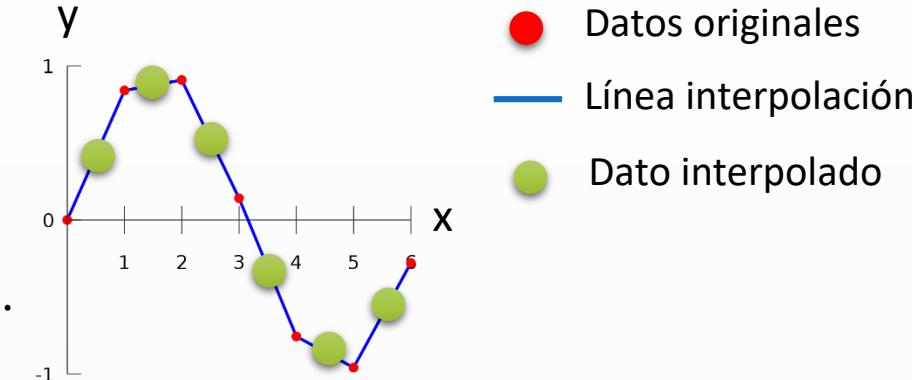
\*Valor puede ser una constante, la mediana de esa columna,...

```
In [29]: df_fill["A"].fillna(df_fill["A"].median(),inplace=True)
```

	A	B	C	D
0	12.0	3.0	20.0	14.0
1	4.0	2.0	16.0	3.0
2	5.0	54.0	NaN	NaN
3	4.5	3.0	3.0	NaN
4	1.0	NaN	8.0	6.0

\*Si queremos insertar valores NaN:  
 Import numpy as np  
 df[df<0]=np.nan

	A	B	C	D
0	12.0	3.0	20.0	14.0
1	4.0	2.0	16.0	3.0
2	5.0	54.0	9.5	4.0
3	3.0	3.0	3.0	5.0
4	1.0	3.0	8.0	6.0



**interpolate**

```
df.interpolate(method ='linear', limit_direction ='forward')
```

```
In [8]: df_interp = df.interpolate(method ='linear', limit_direction ='forward')
```

	A	B	C	D
0	12.0	3.0	20.0	14.0
1	4.0	2.0	16.0	3.0
2	5.0	54.0	9.5	4.0
3	3.0	3.0	3.0	5.0
4	1.0	3.0	8.0	6.0

# Filtrar datos en un dataframe Pandas

## Filtrar Dataframe

`df[df['column_name']] <-> condición]`

	Fecha	País	Temperatura
0	01/01/2021	España	20.0
1	01/01/2021	México	25.0
2	01/01/2021	Venezuela	28.0
3	02/01/2021	España	21.0
4	02/01/2021	México	NaN
5	02/01/2021	Venezuela	30.0

### Tipo de filtrado

Condición en una columna numérica

### Ejemplo

Filtrar registros con Temperatura > 22:

```
In [10]: df_temp[df_temp["Temperatura"] > 22]
```



	Fecha	País	Temperatura
1	01/01/2021	México	25.0
2	01/01/2021	Venezuela	28.0
5	02/01/2021	Venezuela	30.0

Condición en una columna string

Filtrar registros del país “España”:

```
In [16]: df_temp[df_temp["País"]=="España"]
```

```
In [4]: df_temp[df_temp["País"].str.match("España")]
```



	Fecha	País	Temperatura
0	01/01/2021	España	20.0
3	02/01/2021	España	21.0

Condición en una columna fecha

Filtrar registros del día 02/01/2021:

```
In [17]: df_temp[df_temp["Fecha"]=="02/01/2021"]
```



	Fecha	País	Temperatura
3	02/01/2021	España	21.0
4	02/01/2021	México	NaN
5	02/01/2021	Venezuela	30.0

Condición no nulo (NaN)

Filtrar registros cuyo valor de temperatura sea no nulo:

```
df_temp[df_temp["Temperatura"].notnull()]
```



	Fecha	País	Temperatura
0	01/01/2021	España	20.0
1	01/01/2021	México	25.0
2	01/01/2021	Venezuela	28.0
3	02/01/2021	España	21.0
5	02/01/2021	Venezuela	30.0

Múltiples condiciones

Filtrar registros del país “México” que sean no nulos:

```
In [22]: df_temp[(df_temp["País"]=="México") & (df_temp["Temperatura"].notnull())]
```



	Fecha	País	Temperatura
1	01/01/2021	México	25.0

# Filtrar datos en un dataframe Pandas

- **EJERCICIO:** Crear el dataframe (df) a partir del fichero con la información de tráfico peatonal (PEATONES\_2020\_mod.csv):
- Cree un dataframe (df2) con el filtrado de datos necesario si queremos enfocar nuestros análisis en qué ocurrió los días comprendidos en el rango [01/01/2020 – 01/05/2020]. (mm/dd/aaaa)
- Sobre el mismo dataframe df2, realice el filtrado para el distrito “Centro”. Utilice str.contains
- Verifique si hay valores erróneos en la columna “PEATONES” en estas condiciones, en caso afirmativo cree el dataframe df\_interp con la interpolación lineal de df2.

```
In [24]: df2 = df[(df["FECHA"]>="01/01/2020") & (df["FECHA"]<="01/05/2020")]
```

```
In [26]: df2[df2["DISTRITO"].str.contains("Centro")]
```

```
In [35]: df_interp = df2.interpolate(method ='linear', limit_direction ='forward')
```

# Ordenación valores en un dataframe Pandas

- **Sintaxis sort\_values:**

- `df.sort_values(by="nombre_columna", ascending =True/False)`

	País	Poblacion	Renta per capita	Esperanza de vida
0	United States	325084756	59939	78,9
1	China	1421021791	8612	76,7
2	Japan	127502725	38214	84,5
3	Germany	82658409	44680	81,2
4	India	1338676785	1980	69,4
...	...	...	...	...
163	Samoa	195352	4305	73,2
164	Tonga	101998	4193	70,8
165	Palau	17808	16275	73,7
166	Marshall Islands	58058	3517	73,9
167	Kiribati	114158	1626	68,1

- ¿Cuál es el TOP 3 Renta per capita?

In [23]: `df.sort_values(by="Renta per capita", ascending=False)[0:3]`

Columna numérica

Columna texto

Selección Datos

	País	Poblacion	Renta per capita	Esperanza de vida
68	Luxembourg	59191	105280	82,1
26	Norway	5296326	75428	82,3
98	Iceland	334393	73233	82,9

In [15]: `df_ord_renta = df.sort_values(by='Renta per capita', ascending=False)`

In [16]: `df_ord_renta`

Out[16]:

	País	Poblacion	Renta per capita	Esperanza de vida
68	Luxembourg	59191	105280	82,1
26	Norway	5296326	75428	82,3
98	Iceland	334393	73233	82,9
32	Ireland	4753279	69727	82,1
0	United States	325084756	59939	78,9
...	...	...	...	...
117	Mozambique	28649018	441	60,2
151	Central African Republic	4596023	424	52,8
133	Niger	21602382	376	62
136	Malawi	17670196	357	63,8
146	Burundi	10827019	293	61,2

168 rows × 4 columns

In [17]: `df_ord_pais = df.sort_values(by='País', ascending=True)`

In [18]: `df_ord_pais`

Out[18]:

	País	Poblacion	Renta per capita	Esperanza de vida
106	Afghanistan	36296113	538	64,5
50	Algeria	41389189	4048	76,7
147	Andorra	77001	39128	81,8
53	Angola	29816766	4096	60,8
155	Antigua and Barbuda	95426	15825	76,9
...	...	...	...	...
162	Vanuatu	28551	3022	70,3
43	Vietnam	94600648	2366	75,3
92	Yemen	27834819	1123	66,1
96	Zambia	16853599	1535	63,5
103	Zimbabwe	14236595	1548	61,2

# Crear columnas en un dataframe para cadenas de texto

	país	capital	superficie (Mkm2)
0	Brasil	Brasilia	8.5
1	México	Ciudad de México	1.9
2	España	Madrid	1.2
3	Perú	Lima	0.5
4	Colombia	Bogotá	1.1
5	Venezuela	Caracas	0.9

	país	capital	superficie (Mkm2)	país-capital	país-sup	país_etiqueta
0	Brasil	Brasilia	8.5	Brasil-Brasilia	Brasil-8.5	Br
1	México	Ciudad de México	1.9	México-Ciudad de México	México-1.9	Mé
2	España	Madrid	1.2	España-Madrid	España-1.2	Es
3	Perú	Lima	0.5	Perú-Lima	Perú-0.5	Pe
4	Colombia	Bogotá	1.1	Colombia-Bogotá	Colombia-1.1	Co
5	Venezuela	Caracas	0.9	Venezuela-Caracas	Venezuela-0.9	Ve

Tipo Nueva Columna	Ejemplo
Concatenación columnas de tipo string	Crear columna con la concatenación de país – capital: <pre>In [5]: df_pais["pais-capital"] = df_pais["pais"] + " - " + df_pais["capital"]</pre>
Concatenación de columnas string + int/float	Crear columna con la concatenación de país – superficie: <pre>In [9]: df_pais["pais-sup"] = df_pais["pais"] + " - " + df_pais["superficie (Mkm2)"].astype(str)</pre>
Extracción izquierda / derecha / central	Crear columna etiqueta del país con las 2 primeras letras <pre>In [18]: df_pais["pais_etiqueta"] = df_pais["pais"].str[0:2]</pre>

Selección tradicional de datos  
en Python

# Crear columnas en un dataframe a partir de un diccionario con map

*df\_pais*

	país	capital	superficie (Mkm2)	país-capital	país-sup	país_etiqueta
0	Brasil	Brasilia	8.5	Brasil-Brasilia	Brasil-8.5	Br
1	México	Ciudad de México	1.9	México-Ciudad de México	México-1.9	Mé
2	España	Madrid	1.2	España-Madrid	España-1.2	Es
3	Perú	Lima	0.5	Perú-Lima	Perú-0.5	Pe
4	Colombia	Bogotá	1.1	Colombia-Bogotá	Colombia-1.1	Co
5	Venezuela	Caracas	0.9	Venezuela-Caracas	Venezuela-0.9	Ve

- ¿Podemos crear una columna en el dataframe con el continente asociado a partir del diccionario?

- **Sintaxis:**

- `df["nueva_columna"] = df["nombre_columna"].map(diccionario)`



Columna dataframe relacionada con las claves del diccionario

*Diccionario continentes (dic\_pc)*

dic_pc	
'Es':	Europa'
'It':	Europa'
'Al':	Europa'
'Po':	Europa'
'Mé':	América'
'Co':	África'
'Br':	América'
'Cu':	América'
'Ve':	América'
'Pe':	América'
'Ca':	América'
'Chi':	'Asia'
'In':	'Asia'
'Zi':	África'
'Aus':	'Oceanía'

Claves: Etiquetas país

Valores: Continente

In [33]: `df_pais["Continente"] = df_pais["país_etiqueta"].map(dic_pc)`



	país	capital	superficie (Mkm2)	país-capital	país-sup	país_etiqueta	Continente
0	Brasil	Brasilia	8.5	Brasil-Brasilia	Brasil-8.5	Br	América
1	México	Ciudad de México	1.9	México-Ciudad de México	México-1.9	Mé	América
2	España	Madrid	1.2	España-Madrid	España-1.2	Es	Europa
3	Perú	Lima	0.5	Perú-Lima	Perú-0.5	Pe	América
4	Colombia	Bogotá	1.1	Colombia-Bogotá	Colombia-1.1	Co	África
5	Venezuela	Caracas	0.9	Venezuela-Caracas	Venezuela-0.9	Ve	América

# Crear columnas en un dataframe a partir de funciones lambda

- ¿Cómo podemos aplicar funciones lambda un dataframe? **Método apply**
- Sintaxis apply:
- **objeto apply(lambda arg : código lógico)**

## Funciones lambda sobre 1 columna numérica

	Valores
0	4.190832
1	2.726547
2	3.303320
3	1.840289
4	4.594072

In [18]: `df["Valores_cuadrado"] = df["Valores"].apply(lambda x:x**2)`

	Valores	Valores_cuadrado
0	4.190832	17.563075
1	2.726547	7.434056
2	3.303320	10.911924
3	1.840289	3.386662
4	4.594072	21.105498

## Funciones lambda sobre 1 columna string

	Artículo	Precio	Cantidad
0	A	4.85\$	2
1	B	5.23\$	3
2	C	7.85\$	1
3	D	4.25\$	6

In [32]: `df2["Precio"] = df2["Precio"].apply(lambda x:x.replace("$", "€"))`

	Artículo	Precio	Cantidad
0	A	4.85€	2
1	B	5.23€	3
2	C	7.85€	1
3	D	4.25€	6

In [37]: `df2["Precio"] = df2["Precio"].apply(lambda x:x.replace("$", ""))`

	Artículo	Precio	Cantidad
0	A	4.85	2
1	B	5.23	3
2	C	7.85	1
3	D	4.25	6

## Funciones lambda con múltiples columnas

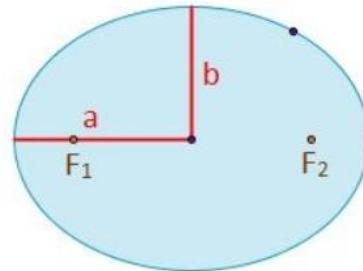
	Artículo	Precio	Cantidad
0	A	4.85	2
1	B	5.23	3
2	C	7.85	1
3	D	4.25	6

In [68]: `df2["Total Venta"] = df2.apply(lambda x:x["Precio"]*x["Cantidad"], axis=1)`

	Artículo	Precio	Cantidad	Total Venta
0	A	4.85	2	9.70
1	B	5.23	3	15.69
2	C	7.85	1	7.85
3	D	4.25	6	25.50

# Crear columnas en un dataframe a partir de funciones lambda

- **EJERCICIO:** Crear el dataframe (df) a partir del fichero iris adjunto.
  - Calcular el área del pétalo para todos los registros considerando cada pétalo como una elipse (puede usar la constante pi = 3.14 o bien importar numpy y usar np.pi)



$$\text{Área} = \pi \cdot a \cdot b$$

Siendo  $a$  y  $b$  los semiejes mayor y menor de la elipse

- ¿Cuál es la mediana del área del pétalo?
- ¿Qué valor de área solo es superado por el 10% de los pétalos?
- ¿Existe correlación entre la longitud y anchura del pétalo? (Nota: Usar método corr() sobre el dataframe)

```
In [74]: df_iris["Área pétalo"] = df_iris.apply(lambda x: np.pi*x["longitud_pétalo"]*x["anchura_pétalo"], axis=1)
```

```
In [76]: df_iris["Área pétalo"].median()
```

```
Out[76]: 17.64004274990669
```

```
In [77]: df_iris["Área pétalo"].quantile(0.9)
```

```
Out[77]: 38.93375775593831
```

```
In [78]: df_iris.corr()
```

```
Out[78]:
```

	longitud_sépalo	anchura_sépalo	longitud_pétalo	anchura_pétalo	Área pétalo
longitud_sépalo	1.00000	-0.109369	0.871754	0.817954	0.857326
anchura_sépalo	-0.109369	1.00000	-0.420516	-0.356544	-0.280612
longitud_pétalo	0.871754	-0.420516	1.00000	0.962757	0.958472
anchura_pétalo	0.817954	-0.356544	0.962757	1.00000	0.980229
Área pétalo	0.857326	-0.280612	0.958472	0.980229	1.00000

# Crear columnas en un dataframe a partir de funciones condicionales

- ¿Cómo podemos crear columnas en base al cumplimiento de condiciones?

- Ejemplo: Crear una columna “Chequeo” que indique si el registro está por encima o por debajo de la media

## • MÉTODO 1: A PARTIR DE LA DEFINICIÓN DE UNA FUNCIÓN + MÉTODO APPLY

```
In [7]: def supinf(x):
    if x>3:
        z = "Valor mayor que la media"
    else:
        z = "Valor menor que la media"
    return z
```

```
In [18]: df["Chequeo"] = df["Valores"].apply(supinf)
```

	Valores	Chequeo
0	2.679733	Valor menor que la media
1	1.853950	Valor menor que la media
2	1.158537	Valor menor que la media
3	1.324490	Valor menor que la media
4	3.191215	Valor mayor que la media

Valores
0 3.801771
1 3.183058
2 3.021595
3 2.601504
4 3.893110

Distribución de valores con media 3 y desviación estándar 1

## • MÉTODO 2: A PARTIR DE MÉTODO “WHERE” DE NUMPY

### • Sintaxis apply:

- np.where(condición, valor\_si\_verdadero, valor\_si\_falso)

```
In [20]: df["Chequeo2"] = np.where(df["Valores"]>3, "Valor mayor que la media", "Valor menor que la media")
```

	Valores	Chequeo	Chequeo2
0	2.679733	Valor menor que la media	Valor menor que la media
1	1.853950	Valor menor que la media	Valor menor que la media
2	1.158537	Valor menor que la media	Valor menor que la media
3	1.324490	Valor menor que la media	Valor menor que la media
4	3.191215	Valor mayor que la media	Valor mayor que la media

- Mayor síntesis
- Más eficiente (cálculo vectorial)

# Renombrar y reordenar columnas de un dataframe Pandas

Claves = nombre\_original

Valores = nombre\_final

df\_pais

- Sintaxis Renombrar columnas:

- df.rename(columns={"nombre\_orig\_1": "nombre\_final\_1", "nombre\_orig\_2": "nombre\_final\_2", ...}, inplace = True)

	país	capital	superficie (Mkm2)
0	Brasil	Brasilia	8.5
1	México	Ciudad de México	1.9
2	España	Madrid	1.2
3	Perú	Lima	0.5
4	Colombia	Bogotá	1.1
5	Venezuela	Caracas	0.9



In [5]: df\_pais.rename(columns={"país": "País", "superficie (Mkm2)": "Superficie (Mkm\_2)"}, inplace=True)

	País	capital	Superficie (Mkm_2)
0	Brasil	Brasilia	8.5
1	México	Ciudad de México	1.9
2	España	Madrid	1.2
3	Perú	Lima	0.5
4	Colombia	Bogotá	1.1
5	Venezuela	Caracas	0.9

- Sintaxis Reordenar columnas:

- df = df[lista\_nombre\_columnas]



In [7]: df\_pais = df\_pais[["País", "Superficie (Mkm\_2)", "capital"]]

	País	Superficie (Mkm_2)	capital
0	Brasil	8.5	Brasilia
1	México	1.9	Ciudad de México
2	España	1.2	Madrid
3	Perú	0.5	Lima
4	Colombia	1.1	Bogotá
5	Venezuela	0.9	Caracas

# Cómo crear pivot tables en Pandas

**df\_ventas**

ID_Pedido	Unidades	Precio Unitario	Total Venta	Fecha	Línea de Producto	Código de Producto	País	Territorio
0	2	30	95,7	2871	24/02/2020	Motorcycles	S10_1678	USA
1	5	34	81,35	2765.9	05/07/2020	Motorcycles	S10_1678	France
2	2	41	94,74	3884.34	07/01/2020	Motorcycles	S10_1678	France
3	6	45	83,26	3746.7	25/08/2020	Motorcycles	S10_1678	USA
4	14	0	100	0	10/10/2020	Motorcycles	S10_1678	USA
...	...	...	...	...	...	...	...	...
2816	15	20	100	2244.4	12/02/2021	Ships	S72_3212	Spain
2817	1	29	100	3978.51	1/31/2022	Ships	S72_3212	Finland
2818	4	86	100	10835.14	03/01/2022	Ships	S72_3212	Spain
2819	1	34	62,24	2116.16	3/28/2022	Ships	S72_3212	France
2820	9	94	65,52	6158.88	05/06/2022	Ships	S72_3212	USA

2821 rows × 9 columns

- ¿Cómo podemos crear una tabla que agregue la información (`pivot_table`)?
- Sintaxis `pivot_table`:
  - `df.pivot_table(index="nombre_columna", columns="nombre_columna", values=" nombre_columna ", aggfunc="método_agregación")`

Columna por la que se agrega el índice

Columna por la que se agrega en las columnas

Valores que se están agregando

Método de agregación (sum, mean, median,...)

```
In [27]: df_ventas.pivot_table(index="País", columns="Línea de Producto", values="Total Venta", aggfunc="sum")
```

Línea de Producto	Classic Cars	Motorcycles	Planes	Ships	Trains	Trucks and Buses	Vintage Cars
País							
Australia	197237.40	100495.19	85935.65	7230.16	1681.35	83945.20	221400.86
Austria	101459.47	26047.66	17860.44	9024.73	NaN	20472.75	35972.64
Belgium	36765.12	NaN	5624.79	31708.01	9017.26	NaN	41925.60
Canada	73767.98	4177.49	25510.07	40309.01	NaN	71398.74	42608.24
Denmark	157182.48	NaN	7586.45	38697.26	11476.33	9588.82	21105.81

Suma de ventas de motocicletas en Austria

# Uso de groupby en Pandas

`df_ventas`

ID_Pedido	Unidades	Precio Unitario	Total Venta	Fecha	Línea de Producto	Código de Producto	País	Territorio
0	2	30	95,7	2871	24/02/2020	Motorcycles	S10_1678	USA
1	5	34	81,35	2765.9	05/07/2020	Motorcycles	S10_1678	France
2	2	41	94,74	3884.34	07/01/2020	Motorcycles	S10_1678	France
3	6	45	83,26	3746.7	25/08/2020	Motorcycles	S10_1678	USA
4	14	0	100	0	10/10/2020	Motorcycles	S10_1678	USA
...	...	...	...	...	...	...	...	...
2816	15	20	100	2244.4	12/02/2021	Ships	S72_3212	Spain
2817	1	29	100	3978.51	1/31/2022	Ships	S72_3212	Finland
2818	4	86	100	10835.14	03/01/2022	Ships	S72_3212	Spain
2819	1	34	62,24	2116.16	3/28/2022	Ships	S72_3212	France
2820	9	94	65,52	6158.88	05/06/2022	Ships	S72_3212	USA

2821 rows × 9 columns

- ¿Cómo podemos agrupar la información (groupby)?

- Sintaxis groupby:

- `df.groupby("lista_columnas_agrupación").["columna_valores"].agg("lista_agregaciones")`

```
In [15]: df_ventas.groupby(["País"])["Total Venta"].agg('sum')
```

País	Total Venta
Australia	697925.81
Austria	210837.69
Belgium	125040.78
Canada	257771.53
Denmark	245637.15
Finland	407378.22
France	1182209.72
Germany	220472.09
Ireland	57756.43
Italy	416184.25
Japan	188167.81
Norway	307463.70
Philippines	94015.73
Singapore	292664.01
Spain	1373565.85
Sweden	241620.93
Switzerland	117713.56
UK	519683.27
USA	3958291.13

```
In [28]: df_ventas.groupby(["Territorio","País"]).agg('sum')
```

Territorio	País	ID_Pedido	Unidades	Precio Unitario	Total Venta
APAC	Australia	1180	6921	15349.14	697925.81
	Singapore	227	1236	2862.93	115498.73
	Austria	320	2076	4759.16	210837.69
	Belgium	164	1191	2887.31	125040.78
	Denmark	399	2197	5503.89	245637.15
	Finland	587	3915	7722.15	407378.22
	France	2014	11816	25820.87	1182209.72
	Germany	388	2148	5184.30	220472.09
	Ireland	73	490	1377.98	57756.43
EMEA	Italy	890	4233	9329.40	416184.25
	Norway	559	2842	7318.18	307463.70
	Spain	2186	14070	28042.54	1373565.85
	Sweden	337	2322	4943.51	241620.93
	Switzerland	258	1078	2713.09	117713.56
	UK	1027	5384	11882.70	519683.27
	Japan	362	1842	4290.26	188167.81
	Philippines	167	961	2144.99	94015.73
	Singapore	317	1568	3718.97	115498.73

# Uso de groupby en Pandas

- ¿Podemos realizar varios tipos de agregación en paralelo?

```
In [29]: df_ventas.groupby(['Territorio','País'])["Total Venta"].agg(["count","max","min","sum"])
```

Territorio	País	count	max	min	sum
APAC	Australia	184	16689.42	652.35	697925.81
	Singapore	36	7020.48	785.64	115498.73
	Austria	55	9240.00	640.05	210837.69
	Belgium	33	12551.44	881.40	125040.78
	Denmark	63	10468.90	1146.50	245637.15
	Finland	92	14529.06	891.03	407378.22
	France	313	20079.20	482.13	1182209.72
EMEA	Germany	62	8940.96	948.99	220472.09
	Ireland	16	8258.00	1056.40	57756.43
	Italy	113	11285.66	577.60	416184.25
	Norway	85	8844.12	1129.04	307463.70
	Spain	342	23775.60	683.80	1373565.85
	Sweden	57	13962.00	1467.48	241620.93
	Switzerland	31	6761.60	1205.04	117713.56
Japan	UK	144	23773.20	710.20	519683.27
	Japan	52	10758.00	553.95	188167.81
	Philippines	26	7483.98	1173.15	94015.73
	Singapore	43	10993.50	859.14	177165.28

```
In [27]: df_ventas.groupby("País")["Total Venta"].agg(["count","max","min","sum",lambda x:x.max()-x.min()])
```

País	count	max	min	sum	<lambda_0>
Australia	184	16689.42	652.35	697925.81	16037.07
Austria	55	9240.00	640.05	210837.69	8599.95
Belgium	33	12551.44	881.40	125040.78	11670.04
Canada	70	12464.00	1119.93	257771.53	11344.07
Denmark	63	10468.90	1146.50	245637.15	9322.40
Finland	92	14529.06	891.03	407378.22	13638.03
France	313	20079.20	482.13	1182209.72	19597.07
Germany	62	8940.96	948.99	220472.09	7991.97
Ireland	16	8258.00	1056.40	57756.43	7201.60
Italy	113	11285.66	577.60	416184.25	10708.06
Japan	52	10758.00	553.95	188167.81	10204.05
Norway	85	8844.12	1129.04	307463.70	7715.08
Philippines	26	7483.98	1173.15	94015.73	6310.83
Singapore	79	10993.50	785.64	292664.01	10207.86
Spain	342	23775.60	683.80	1373565.85	23091.80
Sweden	57	13962.00	1467.48	241620.93	12494.52
Switzerland	31	6761.60	1205.04	117713.56	5556.56
UK	144	23773.20	710.20	519683.27	23063.00
USA	1004	20133.20	0.00	3958291.13	20133.20

# Uso de groupby en Pandas

- EJERCICIO:** Crear el dataframe (df\_peatones) a partir del fichero con la información de tráfico peatonal (PEATONES\_2020\_mod.csv). Además
  - Calcular el número de peatones promedio por cada uno de los distritos.
  - Obtener el mínimo y el promedio de peatones en cada distrito en cada uno de los días ("FECHA").
  - Calcular cuál ha sido el tráfico de peatones máximo total considerando que hay que añadir un 10% de factor de crecimiento por transeúntes motorizados (usar función lambda)

```
In [33]: df_peatones.groupby("DISTRITO")["PEATONES"].agg("mean")
out[33]: DISTRITO
Arganzuela    651.036304
Centro         488.248612
Chamberí       269.302136
Moncloa-Aravaca 360.899778
Latina          71.679353
Name: PEATONES, dtype: float64
```

```
In [5]: df_peatones.groupby(["FECHA", "DISTRITO"])["PEATONES"].agg(["min", "mean"])
```

```
In [9]: df_peatones.groupby(["FECHA", "DISTRITO"])["PEATONES"].agg(["max",lambda x:x.max()*1.1])
out[9]:
```

		FECHA	DISTRITO	max	<lambda_0>
01/01/2020	Arganzuela	1577.0	1734.7		
	Centro	1434.0	1577.4		
	Arganzuela	1809.0	1989.9		
01/02/2020	Centro	1477.0	1624.7		
	Chamberí	1059.0	1164.9		
	Arganzuela	1659.0	1824.9		
31/05/2020	Centro	1487.0	1635.7		
	Chamberí	897.0	986.7		
	Moncloa-Aravaca	786.0	864.6		
Latina	122.0	134.2			

860 rows × 2 columns

# Concatenación de dataframes (union)

TABLA A

Fecha compra	Producto	Cantidad	Importe
15/05/2019	A	2	10
15/05/2019	B	1	10
15/05/2019	A	5	25
18/05/2019	C	1	12
17/06/2019	A	2	10
25/08/2019	F	3	21
25/08/2019	B	1	10
25/08/2019	G	2	18
25/08/2019	D	6	48
25/08/2019	C	1	12

TABLA B

Fecha compra	Producto	Cantidad	Importe
01/09/2019	B	2	20
01/09/2019	C	5	60
01/09/2019	A	1	5
01/09/2019	E	2	10
05/09/2019	F	3	21
05/09/2019	D	4	32

TABLA A + TABLA B

Fecha compra	Producto	Cantidad	Importe
15/05/2019	A	2	10
15/05/2019	B	1	10
15/05/2019	A	5	25
18/05/2019	C	1	12
17/06/2019	A	2	10
25/08/2019	F	3	21
25/08/2019	B	1	10
25/08/2019	G	2	18
25/08/2019	D	6	48
25/08/2019	C	1	12
01/09/2019	B	2	20
01/09/2019	C	5	60
01/09/2019	A	1	5
01/09/2019	E	2	10
05/09/2019	F	3	21
05/09/2019	D	4	32

Tabla A y Tabla B deben tener la misma estructura!

# Concatenación de dataframes (union)

- ¿Cómo podemos realizar la concatenación (unión) con Pandas?
- **Sintaxis concat:**
  - df\_concatenado = pd.concat("lista\_dataframes\_union", ignore\_index = True)

**df1:** Dataframe con ventas del día **06/01/2022** (formato dataframe mm/dd/aaaa)

ID_Pedido	Unidades	Precio Unitario	Total Venta	Fecha	Línea de Producto	Código de Producto	País	Territorio
1990	1	62	94.58	5863.96	01/06/2022	Classic Cars	S24_3856	Finland EMEA
1209	12	64	89.12	5703.68	01/06/2022	Motorcycles	S18_3782	Finland EMEA
413	6	92	88.45	8137.40	01/06/2022	Classic Cars	S12_4675	Finland EMEA
310	4	68	96.73	6577.64	01/06/2022	Classic Cars	S12_3380	Finland EMEA
180	3	66	85.39	5635.74	01/06/2022	Classic Cars	S12_1099	Finland EMEA
1388	13	56	58.18	3258.08	01/06/2022	Classic Cars	S18_4721	Finland EMEA
1160	11	48	100.00	8285.28	01/06/2022	Classic Cars	S18_3482	Finland EMEA
361	5	68	81.62	5550.16	01/06/2022	Classic Cars	S12_3990	Finland EMEA
2190	2	100	100.00	13153.00	01/06/2022	Motorcycles	S32_1374	Finland EMEA
1687	14	86	61.23	5265.78	01/06/2022	Motorcycles	S24_2360	Finland EMEA
2216	1	96	48.28	4634.88	01/06/2022	Motorcycles	S32_2206	France EMEA
1109	10	92	60.30	5547.60	01/06/2022	Classic Cars	S18_3278	Finland EMEA
1583	8	42	100.00	7191.24	01/06/2022	Motorcycles	S24_2000	Finland EMEA
2141	9	86	100.00	10308.82	01/06/2022	Classic Cars	S24_4620	Finland EMEA
1888	15	42	100.00	4895.52	01/06/2022	Classic Cars	S24_3371	Finland EMEA
619	7	44	100.00	7373.08	01/06/2022	Classic Cars	S18_1889	Finland EMEA

**df2:** Dataframe con ventas del día **07/01/2022** (formato dataframe mm/dd/aaaa)

ID_Pedido	Unidades	Precio Unitario	Total Venta	Fecha	Línea de Producto	Código de Producto	País	Territorio
468	1	60	87.06	5223.60	01/07/2022	Classic Cars	S18_1129	USA NaN
2346	3	44	100.00	6850.36	01/07/2022	Motorcycles	S32_4485	USA NaN
2451	2	88	100.00	9968.64	01/07/2022	Motorcycles	S50_4713	USA NaN

In [42]: df\_concatenado = pd.concat([df\_1, df\_2], ignore\_index=True)

ID_Pedido	Unidades	Precio Unitario	Total Venta	Fecha	Línea de Producto	Código de Producto	País	Territorio
0	1	62	94.58	5863.96	01/06/2022	Classic Cars	S24_3856	Finland EMEA
1	12	64	89.12	5703.68	01/06/2022	Motorcycles	S18_3782	Finland EMEA
2	6	92	88.45	8137.40	01/06/2022	Classic Cars	S12_4675	Finland EMEA
3	4	68	96.73	6577.64	01/06/2022	Classic Cars	S12_3380	Finland EMEA
4	3	66	85.39	5635.74	01/06/2022	Classic Cars	S12_1099	Finland EMEA
5	13	56	58.18	3258.08	01/06/2022	Classic Cars	S18_4721	Finland EMEA
6	11	48	100.00	8285.28	01/06/2022	Classic Cars	S18_3482	Finland EMEA
7	5	68	81.62	5550.16	01/06/2022	Classic Cars	S12_3990	Finland EMEA
8	2	100	100.00	13153.00	01/06/2022	Motorcycles	S32_1374	Finland EMEA
9	14	86	61.23	5265.78	01/06/2022	Motorcycles	S24_2360	Finland EMEA
10	1	96	48.28	4634.88	01/06/2022	Motorcycles	S32_2206	France EMEA
11	10	92	60.30	5547.60	01/06/2022	Classic Cars	S18_3278	Finland EMEA
12	8	42	100.00	7191.24	01/06/2022	Motorcycles	S24_2000	Finland EMEA
13	9	86	100.00	10308.82	01/06/2022	Classic Cars	S24_4620	Finland EMEA
14	15	42	100.00	4895.52	01/06/2022	Classic Cars	S24_3371	Finland EMEA
15	7	44	100.00	7373.08	01/06/2022	Classic Cars	S18_1889	Finland EMEA
16	1	60	87.06	5223.60	01/07/2022	Classic Cars	S18_1129	USA NaN
17	3	44	100.00	6850.36	01/07/2022	Motorcycles	S32_4485	USA NaN
18	2	88	100.00	9968.64	01/07/2022	Motorcycles	S50_4713	USA NaN

# Combinación de dataframes (merge)

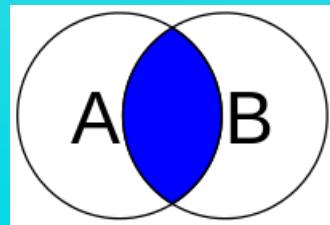
TABLA A      TABLA B

	X	Y
A	2	
A	3	
B	5	
C	3	

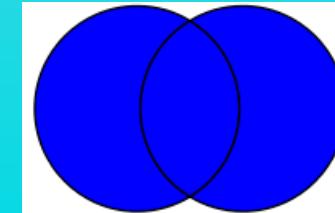
	X	z
A	7	
D	4	
B	3	
F	8	

INNER JOIN

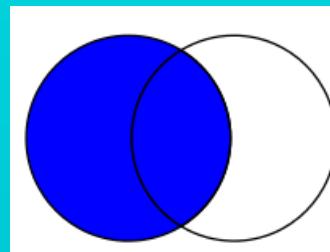


MODOS DE  
JUNTAR TABLAS

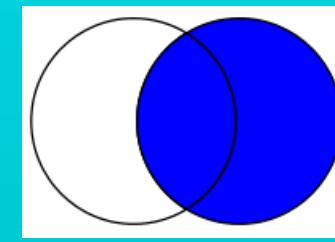
OUTER JOIN



LEFT OUTER  
JOIN



RIGHT OUTER  
JOIN



X	Y	Z
A	2	7
A	3	7
B	5	3

X	Y	Z
A	2	7
A	3	7
B	5	3
C	3	N/A

X	Y	Z
A	2	7
A	3	7
B	5	3
C	3	N/A
D	N/A	4
F	N/A	8

X	Y	Z
A	2	7
A	3	7
B	5	3
D	N/A	4
F	N/A	8

# Combinación de dataframes (merge)

- **Sintaxis Merge:**

- df\_combinado = df1.merge(df2, **on="columna\_clave"**, **how="método\_combinación"**)

Columna común en ambos  
dataframes para poder  
combinar

Outer, Join, Left, Right

df\_pais

	país	capital	superficie (Mkm2)
0	Brasil	Brasilia	8.5
1	México	Ciudad de México	1.9
2	España	Madrid	1.2
3	Perú	Lima	0.5
4	Colombia	Bogotá	1.1
5	Venezuela	Caracas	0.9

df\_poblacion

	país	población
0	Perú	32.6
1	Colombia	50.3
2	Brasil	211.4
3	México	127.8
4	España	47.0
5	EEUU	331.8
6	China	1393.0

```
In [13]: df_total_in = df_pais.merge(df_poblacion, on="país", how="inner")
```

	país	capital	superficie (Mkm2)	población
0	Brasil	Brasilia	8.5	211.4
1	México	Ciudad de México	1.9	127.8
2	España	Madrid	1.2	47.0
3	Perú	Lima	0.5	32.6
4	Colombia	Bogotá	1.1	50.3

```
In [16]: df_total_out = df_pais.merge(df_poblacion, on="país", how="outer")
```

	país	capital	superficie (Mkm2)	población
0	Brasil	Brasilia	8.5	211.4
1	México	Ciudad de México	1.9	127.8
2	España	Madrid	1.2	47.0
3	Perú	Lima	0.5	32.6
4	Colombia	Bogotá	1.1	50.3
5	Venezuela	Caracas	0.9	NaN
6	EEUU	NaN	NaN	331.8
7	China	NaN	NaN	1393.0

\*Venezuela no se encuentra en df\_poblacion

\*Venezuela no se encuentra en df\_poblacion

\*EEUU y China no están en df\_pais

# Combinación de dataframes (merge)

- **EJERCICIO:** Crear el dataframe (df\_peatones) a partir del fichero con la información de tráfico peatonal (PEATONES\_2020\_mod.csv). Además crear un dataframe adicional (df\_pob) con la población en cada uno de los distritos de Madrid.
  - Realizar una combinación de tipo inner de ambos dataframes creando el df\_combinado resultado de utilizar merge.
  - De cara a analizar cuál es el movimiento de personas entre los diferentes distritos de la ciudad, ¿cuál es el distrito que tiene mayor número de peatones en el momento máximo en comparación con su población? Siga los siguientes pasos:
    - Crear un KPI = PEATONES / Población\_distrito
    - Realizar un groupby para verificar el valor máximo del KPI en cada distrito

```
In [65]: df_combinado = df_peatones.merge(df_pob, on="DISTRITO", how="inner")
```

```
In [77]: df_combinado.groupby("DISTRITO")["Peatones/Pob"].agg("max")
```

```
Out[77]: DISTRITO
          Arganzuela      0.026809
          Centro           0.031786
          Chamberí         0.008413
          Latina           0.000920
          Moncloa-Aravaca  0.008811
          Name: Peatones/Pob, dtype: float64
```



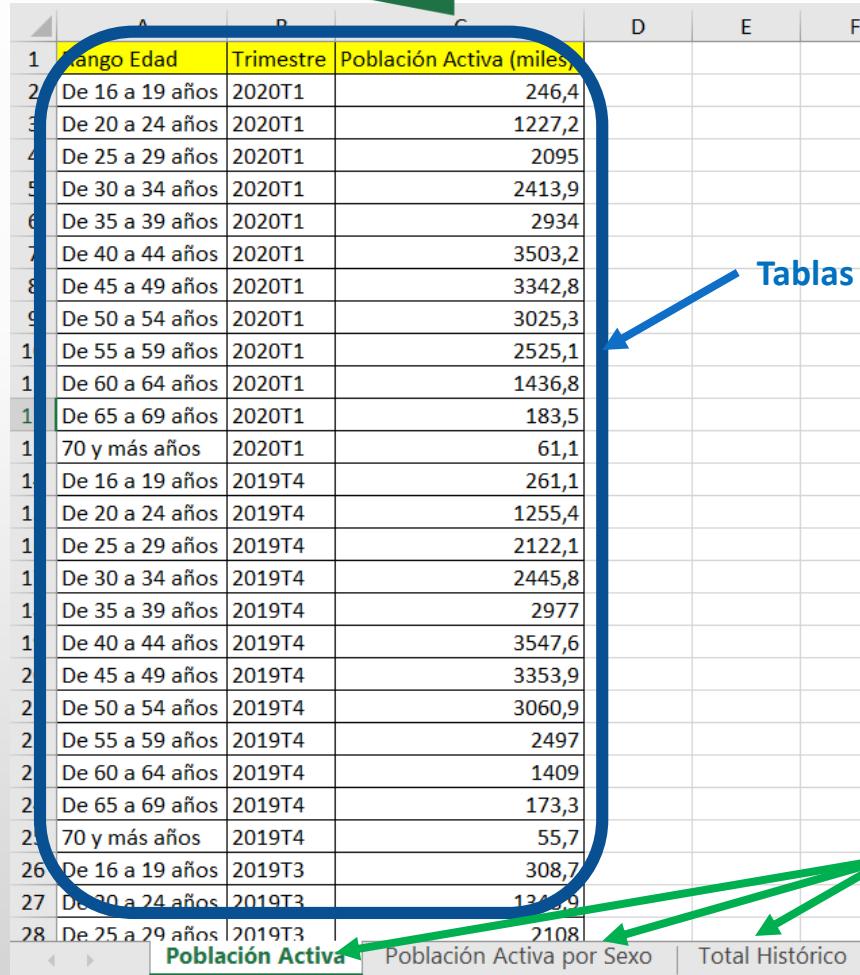
# BLOQUE 6:

## Importación y exportación de datos con Pandas

# Cómo importar datos desde un fichero Excel



**Tablas de datos**



Rango Edad	Trimestre	Población Activa (miles)
De 16 a 19 años	2020T1	246,4
De 20 a 24 años	2020T1	1227,2
De 25 a 29 años	2020T1	2095
De 30 a 34 años	2020T1	2413,9
De 35 a 39 años	2020T1	2934
De 40 a 44 años	2020T1	3503,2
De 45 a 49 años	2020T1	3342,8
De 50 a 54 años	2020T1	3025,3
De 55 a 59 años	2020T1	2525,1
De 60 a 64 años	2020T1	1436,8
De 65 a 69 años	2020T1	183,5
70 y más años	2020T1	61,1
De 16 a 19 años	2019T4	261,1
De 20 a 24 años	2019T4	1255,4
De 25 a 29 años	2019T4	2122,1
De 30 a 34 años	2019T4	2445,8
De 35 a 39 años	2019T4	2977
De 40 a 44 años	2019T4	3547,6
De 45 a 49 años	2019T4	3353,9
De 50 a 54 años	2019T4	3060,9
De 55 a 59 años	2019T4	2497
De 60 a 64 años	2019T4	1409
De 65 a 69 años	2019T4	173,3
70 y más años	2019T4	55,7
De 16 a 19 años	2019T3	308,7
De 20 a 24 años	2019T3	1345,9
De 25 a 29 años	2019T3	2108
<b>Población Activa</b>		
<b>Población Activa por Sexo</b>		
<b>Total Histórico</b>		

## Importar de Excel

```
df = pd.read_excel(r'file.xlsx', sheet_name = "nombre_pestaña")
```

```
In [2]: df = pd.read_excel(r'ruta_local', sheet_name="Población Activa")
```



Rango Edad	Trimestre	Población Activa (miles)
De 16 a 19 años	2020T1	246.4
De 20 a 24 años	2020T1	1227.2
De 25 a 29 años	2020T1	2095.0
De 30 a 34 años	2020T1	2413.9
De 35 a 39 años	2020T1	2934.0
De 40 a 44 años	2020T1	3503.2
De 45 a 49 años	2020T1	3342.8
De 50 a 54 años	2020T1	3025.3
De 55 a 59 años	2020T1	2525.1
De 60 a 64 años	2020T1	1436.8
De 65 a 69 años	2020T1	183.5
70 y más años	2020T1	61.1
De 16 a 19 años	2019T4	261.1
De 20 a 24 años	2019T4	1255.4
De 25 a 29 años	2019T4	2122.1
De 30 a 34 años	2019T4	2445.8
De 35 a 39 años	2019T4	2977
De 40 a 44 años	2019T4	3547.6
De 45 a 49 años	2019T4	3353.9
De 50 a 54 años	2019T4	3060.9
De 55 a 59 años	2019T4	2497
De 60 a 64 años	2019T4	1409
De 65 a 69 años	2019T4	173.3
70 y más años	2019T4	55.7
De 16 a 19 años	2019T3	308.7
De 20 a 24 años	2019T3	1345.9
De 25 a 29 años	2019T3	2108

Diferentes pestañas de datos

# Introducción a las BBDD relacionales / Modelos de datos



SGBD = Sistema de  
Gestión de Base de Datos



IP\_Servidor:  
**10.192.168.27**

BBDD → BBDD Relacional

Tabla de Ventas

ID_Producto	ID_Cliente	ID_Empresario	Fecha compra	Estado Pedido
1	C2	E1	05/06/2021	Enviado
2	C3	E1	05/06/2021	Entregado
5	C3	E3	07/06/2021	Entregado

## Modelo de datos

Tabla de Empleados

ID_Empresario	Nombre	Puesto	Fecha incorporación	País
E1	JSD	Comercial Nivel 1	05/04/2018	Perú
E2	PED	Comercial Nivel 2	06/05/2020	México
E3	TEZ	Gerente comercial	01/02/2015	México

Tabla de Productos

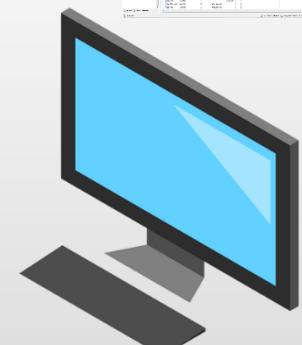
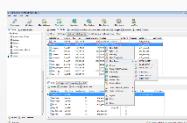
ID_Producto	Tipo Producto	Color	Precio
1	Pantalón	Blanco	40
2	Pantalón	Amarillo	28
3	Camiseta	Azul	15
4	Camiseta	Blanco	22
5	Gorra	Verde	10

Tabla de Clientes

ID_Cliente	Nombre	Dirección	País
C1	IPD	C/ Leonardo, 3	España
C2	BCB	Av. Libertad, 7	México
C3	JLT	C/ Augusto, 6	Venezuela

SQL = Structured  
Query Language

Aplicación Cliente  
SQL



# Cómo importar datos desde una BBDD SQL



IP\_Servidor:

**10.192.168.27**

Pandas (read\_sql)  
+ Sentencia SQL

Sintaxis básica SQL:

SELECT [columnas y agregaciones]  
FROM [tabla de la BBDD]  
GROUP BY [columna por la que agrupar]  
ORDER BY [columna por la que ordenar]



- ¿Cuántos datos debemos importar desde una BBDD SQL?

¡Solo los que sean de interés!

- ¿Cómo importamos datos en Python desde una BBDD SQL?



```
C:\Users\ivan_pinar>conda install pymysql
```



Nombre usuario:contraseña BBDD

Nombre\_BBDD

IP Servidor

```
In [7]: from sqlalchemy import create_engine  
import pymysql
```

```
In [9]: db_conexion = create_engine('mysql+pymysql://root:xxxxxx@127.0.0.1/databarhouse')
```

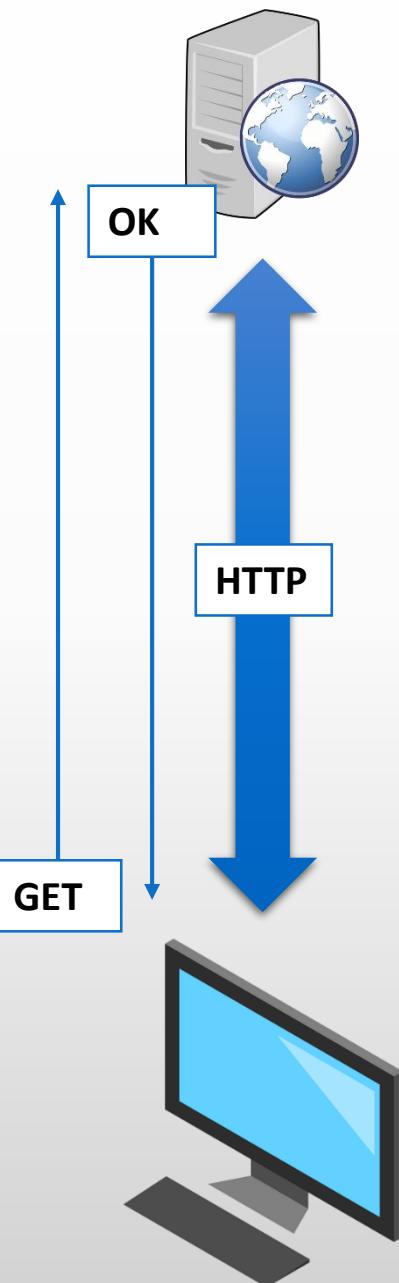
```
In [10]: df = pd.read_sql('SELECT * FROM ventas_acumulada', con=db_conexion)
```

	Order Number	Cantidad Pedido	Order Line Number	Ventas	Fecha Pedido	Estado	Ciudad	País	Precio Producto	Mes Pedido	Población
0	10100	49.0	1.0	1689	2020-06-01 00:00:00	Shipped	Nashua	United States	34.47	6	278357000
1	10100	50.0	2.0	3390	2020-06-01 00:00:00	Shipped	Nashua	United States	67.8	6	278357000
2	10100	30.0	3.0	5151	2020-06-01 00:00:00	Shipped	Nashua	United States	171.7	6	278357000
3	10100	22.0	4.0	1903	2020-06-01 00:00:00	Shipped	Nashua	United States	86.51	6	278357000
4	10101	26.0	1.0	3773	2020-09-01 00:00:00	Shipped	Frankfurt	Germany	145.13	9	82164700
...	...	...	...	...	...	...	...	...	...	...	...
44793	10416	94.0	6.0	8331	2022-10-05 00:00:00	Shipped	Reggio Emilia	Italy	88.63	10	57680000
44794	10416	74.0	8.0	3843	2022-10-05 00:00:00	Shipped	Reggio Emilia	Italy	51.93	10	57680000
44795	10416	78.0	10.0	5242	2022-10-05 00:00:00	Shipped	Reggio Emilia	Italy	67.2	10	57680000
44796	10416	86.0	12.0	5348	2022-10-05 00:00:00	Shipped	Reggio Emilia	Italy	62.19000000000005	10	57680000
44797	10416	48.0	14.0	8704	2022-10-05 00:00:00	Shipped	Reggio Emilia	Italy	181.34	10	57680000

44798 rows x 11 columns

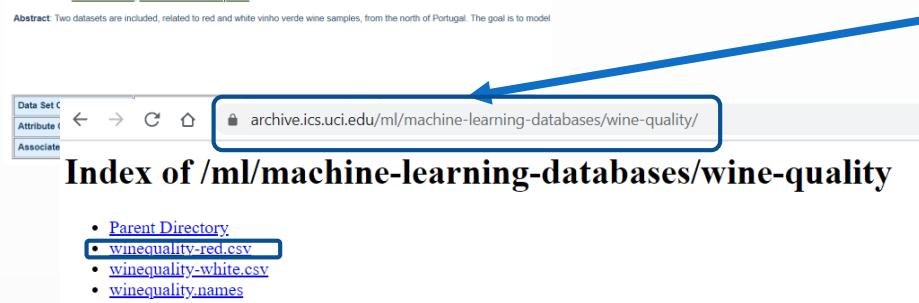
- Importación ODBC → Librería pyodbc

# Cómo importar datos desde una página Web



<https://archive.ics.uci.edu/ml/datasets.php>

URL: Universal Resource Locator



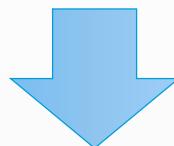
- ¿Cómo importamos un recurso web a partir de una URL?

```
In [1]: import pandas as pd
In [2]: url="https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv"
In [4]: df = pd.read_csv(url,delimiter=";")
In [5]: df
out[5]:
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  free sulfur dioxide  total sulfur dioxide  density  pH  sulphates  alcohol  quality
0            7.4           0.700       0.00          1.9     0.076             11.0            34.0    0.99780   3.51      0.56     9.4      5
1            7.8           0.880       0.00          2.6     0.098             25.0            67.0    0.99680   3.20      0.68     9.8      5
2            7.8           0.760       0.04          2.3     0.092             15.0            54.0    0.99700   3.26      0.65     9.8      5
3           11.2           0.280       0.56          1.9     0.075             17.0            60.0    0.99800   3.16      0.58     9.8      6
4            7.4           0.700       0.00          1.9     0.076             11.0            34.0    0.99780   3.51      0.56     9.4      5
```

# Cómo importar datos desde una página Web (Web scraping)

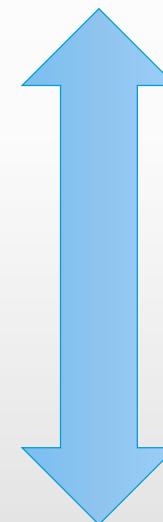
- ¿Qué ocurre si en lugar de tener un recurso en la web con los datos existe una tabla dentro HTML?

[https://es.wikipedia.org/wiki/Poblaci%C3%B3n\\_mundial](https://es.wikipedia.org/wiki/Poblaci%C3%B3n_mundial)



Población por continente [\[editar\]](#)

Continente	Densidad (hab./km <sup>2</sup> )	Superficie (km <sup>2</sup> )	Población (2019)	País más poblado (2019)	Ciudad más poblada (2019)
Asia	86,88	43.810.000	4.677.291.000	China (1.392.240.000)	Tokio (35.682.000)
Africa	32,7	30.370.000	1.110.020.000	Nigeria (191.205.000)	El Cairo (16.794.000)
Europa	70	10.180.000	801.000.000	Rusia (112.000.000 Europa)	Moscú (18.940.000)
América	23,5	42.330.000	1.094.215.000	Estados Unidos (330.028.000)	Ciudad de México (22.577.000) <sup>10</sup>
Oceanía	4,25	9.008.500	40.201.000	Australia (27.240.000)	Sídney (6.550.000)
Antártida	0,0003 (varía)	13.720.000	4.490 (no permanente, varía) <sup>11</sup>	N.D. <sup>[nota 1]</sup>	N.D.



- **SOLUCIÓN:** Web scraping con Pandas / BeautifulSoup

```
In [30]: import pandas as pd
import requests
```

```
In [31]: url = "https://es.wikipedia.org/wiki/Poblaci%C3%B3n_mundial"
```

```
In [32]: r = requests.get(url)
```

Mensaje GET HTTP a la URL

```
In [46]: df_lista = pd.read_html(r.text)
```

Parsea el texto HTML para identificar todas las tablas y guardarla en una lista de dataframes

```
In [47]: df = df_lista[0]
```

Continente	Densidad(hab./km <sup>2</sup> )	Superficie(km <sup>2</sup> )	Población(2019)	País más poblado(2019)	Ciudad más poblada(2019)
0 Asia	8688	43.810.000	4.677.291.000	China (1.392.240.000)	Tokio (35.682.000)
1 África	327	30.370.000	1.110.020.000	Nigeria (191.205.000)	El Cairo (16.794.000)
2 Europa	70	10.180.000	801.000.000	Rusia (112.000.000 Europa)	Moscú (18.940.000)
3 América	235	42.330.000	1.094.215.000	Estados Unidos (330.028.000)	Ciudad de México (22.577.000) <sup>10</sup>
4 Oceanía	425	9.008.500	40.201.000	Australia (27.240.000)	Sídney (6.550.000)
5 Antártida	0,0003(varia)	13.720.000	4.490(no permanente, varia) <sup>11</sup>	N.D.[nota 1]	N.D.

# Cómo importar datos desde una página Web (Web scraping)

- EJERCICIO:** Importar datos de la URL: [https://en.wikipedia.org/wiki/List\\_of\\_countries\\_by\\_average\\_wage](https://en.wikipedia.org/wiki/List_of_countries_by_average_wage) de tal manera que obtengamos la tabla de ingresos medios mensuales por país.

## UNECE statistics [\[edit\]](#)

Gross average monthly wage estimates for 2015 are national and international (OECD, EUROSTAT, CIS)

Gross average monthly wages cover total wages and worked, bonuses and gratuities paid by the employer

	Country	Average monthly wage
1	瑞士 Switzerland	\$7,466
2	卢森堡 Luxembourg	\$5,583
3	挪威 Norway	\$5,418
4	丹麦 Denmark	\$5,310
5	美国 United States	\$4,893
6	澳大利亚 Australia	\$4,700
7	爱尔兰 Ireland	\$4,379
8	荷兰 Netherlands	\$4,289
9	加拿大 Canada	\$4,134

## Ejercicio

```
In [49]: url_ej = "https://en.wikipedia.org/wiki/List_of_countries_by_average_wage"
```

```
In [51]: r = requests.get(url_ej)
```

```
In [52]: df_lista = pd.read_html(r.text)
```

```
In [68]: df=df_lista[15]
df
```

```
Out[68]:
```

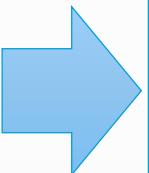
	Country	Average monthly wage
0	Switzerland	\$7,466
1	Luxembourg	\$5,583
2	Norway	\$5,418
3	Denmark	\$5,310
4	United States	\$4,893

# Cómo importar datos desde un fichero semi-estructurado JSON

- **¿Qué es el formato JSON?**

- **JSON (JavaScript Object Notation)** es un formato semiestructurado ligero útil para el intercambio de datos. Es fácil de analizar (parsear).

Elemento: valor {  
    propiedad\_1: valor  
    propiedad\_2: [objeto:  
        propiedad\_2a: valor  
        propiedad\_2b: valor] }



Departamento: “Ingeniería” {  
    Responsable: “Juan Segura”  
    Miembros: [  
        nombre: “José”, apellido: “Núñez”},  
        nombre: “María”, apellido: “Calvo”],

- **¿Dónde se utiliza este formato?**



- **API: Application Program Interface** → Conjunto de protocolos y rutinas que permite conectar 2 aplicaciones software (Ejemplo importar datos desde una Red Social)

# Cómo importar datos desde un fichero semi-estructurado JSON

- ¿Cómo importar datos en formato JSON en Python?

```
In [1]: import json
```

```
In [ ]: with open('ruta local', 'r') as json_file:  
    json_data = json.load(json_file)
```

```
In [156]: json_data  
Out[156]: {'head': {'counts': {'divisionCount': 13915, 'resultCount': 13915},  
            'dateRange': {'start': '1900-01-01T00:00:00.000Z',  
                         'end': '2099-01-01T00:00:00.000Z'},  
            'lang': 'mul'},  
          'results': [{'contextDate': '2020-05-27',  
                      'division': {'outcome': 'Carried',  
                                   'debate': {'uri': 'https://data.oireachtas.ie/akn/ie/debateRecord/dail/2020-05-27/debate/main',  
                                             'debateSection': 'dbsect_2',  
                                             'formats': {'pdf': None,  
                                                        'xml': {'uri': 'https://data.oireachtas.ie/akn/ie/debateRecord/dail/2020-05-27/debate/mul@/main.xml'}},  
                                             'showAs': 'An tOrd GnÃ³ - Order of Business'},  
                      'tellers': 'Tellers: TÃ¡g, Deputies Frankie Feighan and Peter Burke; NÃ\xadndl, Deputies Danny Healy-Rae &  
                                'voteId': 'vote_12',  
                                'datetime': '2020-05-27T12:00:00+01:00',  
                                'subject': {'uri': None},  
                                'showAs': 'Question put: "That the proposal for dealing with Thursday\'s business be agreed to."'},  
          ...]
```

```
In [5]: type(json_data)
```

```
Out[5]: dict
```

```
In [166]: json_data['results']  
Out[166]: [{'contextDate': '2020-05-27',  
            'division': {'outcome': 'Carried',  
                         'debate': {'uri': 'https://data.oireachtas.ie/akn/ie/debateRecord/dail/2020-05-27/debate/main',  
                                   'debateSection': 'dbsect_2',  
                                   'formats': {'pdf': None,  
                                              'xml': {'uri': 'https://data.oireachtas.ie/akn/ie/debateRecord/dail/2020-05-27/debate/mul@/main.xml'}},  
                                   'showAs': 'An tOrd GnÃ³ - Order of Business'},  
            'tellers': 'Tellers: TÃ¡g, Deputies Frankie Feighan and Peter Burke; NÃ\xadndl, Deputies Danny Healy-Rae &  
                                'voteId': 'vote_12',  
                                'datetime': '2020-05-27T12:00:00+01:00',  
                                'subject': {'uri': None},  
                                'showAs': 'Question put: "That the proposal for dealing with Thursday\'s business be agreed to."'},  
          ...]
```



Convertir a dataframe:  
json\_normalize

	contextDate	division.outcome	division.debate.uri	division.debate.debateSection	division.debate.formats.pdf
0	2020-05-27	Carried	https://data.oireachtas.ie/akn/ie/debateRecord...	dbsect_2	None https
1	2020-05-13	Lost	https://data.oireachtas.ie/akn/ie/debateRecord...	dbsect_2	None https
2	2020-05-12	Lost	https://data.oireachtas.ie/akn/ie/debateRecord...	dbsect_2	None https

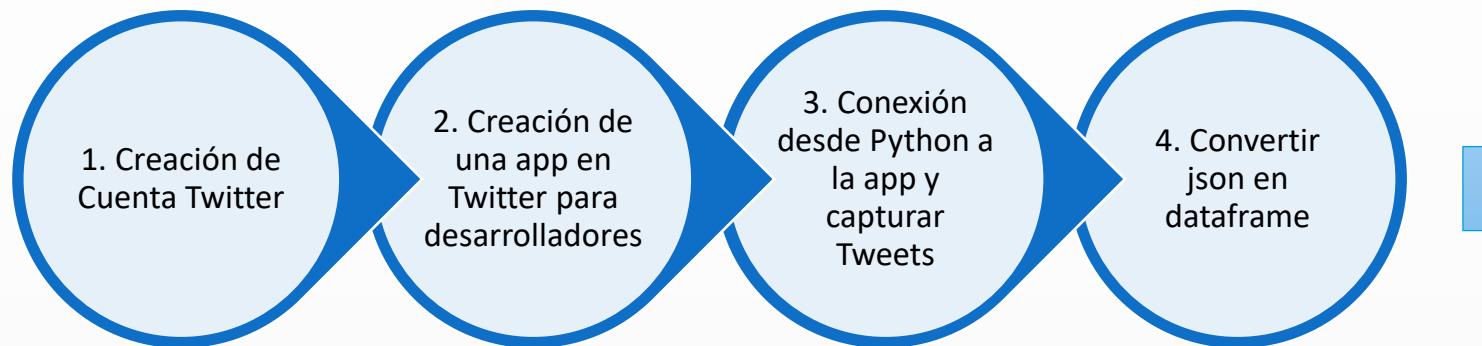


```
In [168]: from pandas.io.json import json_normalize  
import json
```

```
with open(r'C:\Users\ivan_pinar\Dropbox\CreaciÃ³n de MOCs\dail.json') as data_file:  
    d = json.load(data_file)  
  
df = json_normalize(d, 'results')  
df.head()
```

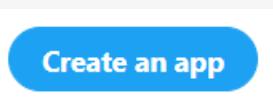
# Cómo importar datos desde Redes Sociales

- ¿Cómo conectar a Redes Sociales?  
Ejemplo Twitter

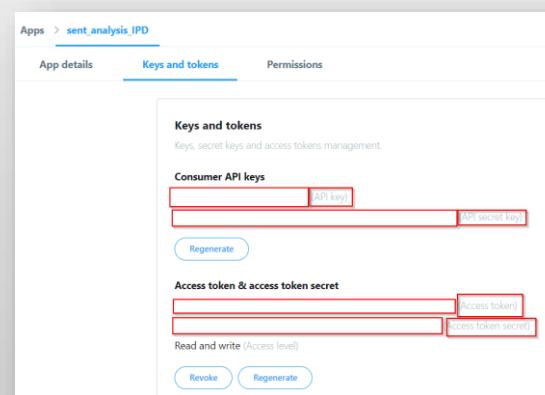


```
In [ ]: df = pd.read_json('ruta_local\tweets.json')
```

<https://apps.twitter.com/>



Website URL: [twitter.com](https://twitter.com)



```
In [94]: df.head()
```

Out[94]:

	created_at	id	id_str	text	truncated	entities	metadata
0	2020-06-01 11:24:46+00:00	1267416797227225090	1267416797227225088	RT @eldiarioes: Un juzgado de Madrid rechaza u...	False	{'hashtags': [], 'symbols': [], 'user_mentions': []}	{'iso_language_code': 'es', 'result_type': 're...', 're...
1	2020-06-01 11:24:45+00:00	1267416793959858177	1267416793959858176	Podcast Amsterdam 98 #0.1 "La Séptima" 20-May...	False	{'hashtags': [], 'symbols': [], 'user_mentions': []}	{'iso_language_code': 'es', 'result_type': 're...', 're...
2	2020-06-01 11:24:45+00:00	1267416793066409986	1267416793066409984	@Rita_Maestre @Monica_Garcia_G Claro Ritall Si...	True	{'hashtags': [], 'symbols': [], 'user_mentions': []}	{'iso_language_code': 'es', 'result_type': 're...', 're...
3	2020-06-01 11:24:45+00:00	1267416793011892225	1267416793011892224	Ya están "marcandose" noticias.\Según marca, ...	True	{'hashtags': [], 'symbols': [], 'user_mentions': []}	{'iso_language_code': 'es', 'result_type': 're...', 're...
4	2020-06-01 11:24:45+00:00	1267416792428883969	1267416792428883968	RT @Tonicanto1: Abalos sale a justificar a Irel...	False	{'hashtags': [], 'symbols': [], 'user_mentions': []}	{'iso_language_code': 'es', 'result_type': 're...', 're...

5 rows × 26 columns

# Cómo importar datos desde Cloud (AWS / Azure / Google Cloud)

- ¿Cuáles son las principales plataformas de Cloud Computing?



32,4%\*



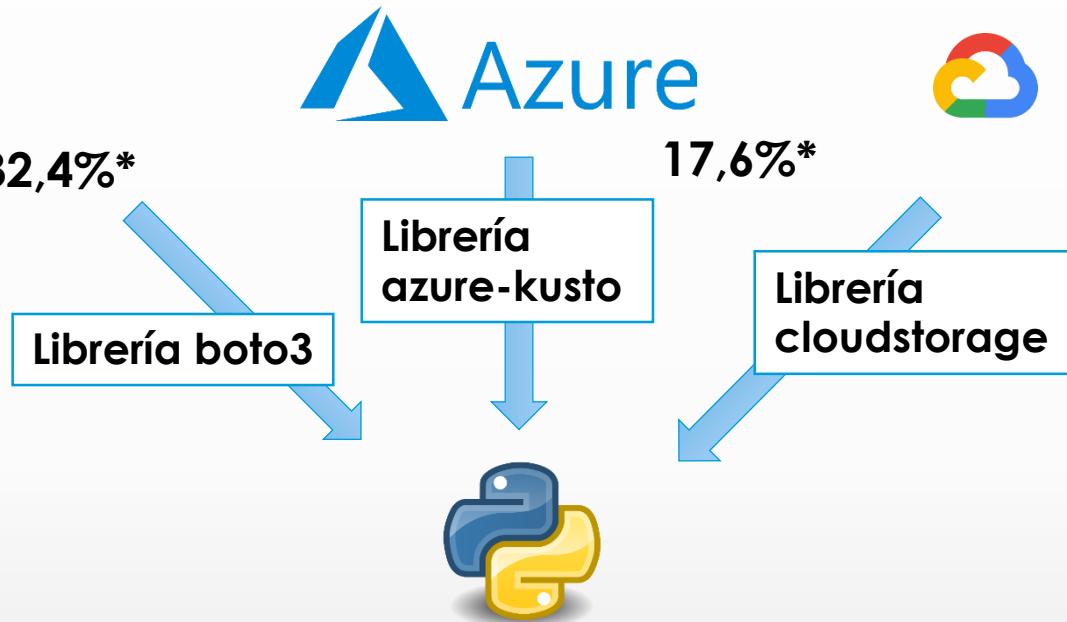
17,6%\*



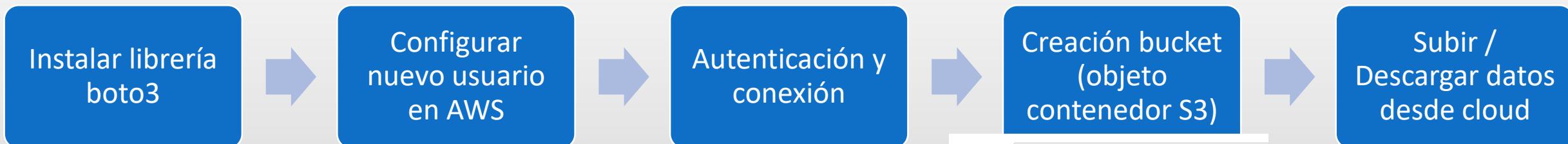
Google Cloud

6%\*

- ¿Cómo conectamos con las plataformas Cloud para importar datos en Python?



## Ejemplo AWS:



```
C:\Users\ivan_pinar>pip install boto3
```



ID

Fichero config & credentials en  
C/Users/nombre\_usuario/.aws/

```

In [ ]: import boto3
In [ ]: s3_resource = boto3.resource('s3')
In [ ]: s3_resource.create_bucket(Bucket='YOUR_BUCKET_NAME',
                                CreateBucketConfiguration={
                                    'LocationConstraint': 'eu-west-1'})
In [ ]: s3_resource.Object(first_bucket_name, first_file_name).download_file(
        f'/tmp/{first_file_name}')
  
```

# Exportación de datos a csv y Excel



## Exportar a CSV

`df.to_csv(r'ruta_local/nombre_fichero.csv')`

In [ ]: `df_pais.to_csv(r'ruta_local\Información_pais_.csv')`



***df\_pais***

	país	capital	superficie (Mkm2)
0	Brasil	Brasilia	8.5
1	México	Ciudad de México	1.9
2	España	Madrid	1.2
3	Perú	Lima	0.5
4	Colombia	Bogotá	1.1
5	Venezuela	Caracas	0.9



	país	capital	superficie (Mkm2)
1	0,Brasil	Brasilia	8.5
2	1,México	Ciudad de México	1.9
3	2,España	Madrid	1.2
4	3,Perú	Lima	0.5
5	4,Colombia	Bogotá	1.1
6	5,Venezuela	Caracas	0.9



## Exportar a Excel

`df.to_excel(r'ruta_local/nombre_fichero.xlsx', ,sheet_name = nombre_pestana)`

In [ ]: `df_pais.to_excel(r'ruta_local\Información_pais_.xlsx', sheet_name = "Info")`



A	B	C	D	E
1	país	capital	superficie (Mkm2)	
2	0	Brasil	8,5	
3	1	Ciudad de	1,9	
4	2	España	1,2	
5	3	Perú	0,5	
6	4	Colombia	1,1	
7	5	Venezuela	0,9	
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				

# Exportación de datos a csv y Excel

- **EJERCICIO:** Cargar el fichero CSV con la evolución del índice S&P500. A continuación realizar los siguientes pasos:
    - Crear una columna con la variación diaria porcentual del índice ((“Close” – “Open”)/”Open \* 100)
    - Exportar el resultado a un fichero Excel con nombre S&P500\_v2 en una pestaña denominada “Evolución SP500”.

```
In [7]: df_sp500 = pd.read_csv(r'C:\Users\ivan_pinar\Dropbox\Creación de MOCs\MOC Master Python Ar
```

```
In [13]: df_sp500["Variación día"] = (df_sp500["Close"]-df_sp500["Open"])/df_sp500["Open"]*100
```

```
In [16]: #Exportar a Excel  
df_sp500.to_excel(r'C:\Users\ivan pinar\Dropbox\Creación de MOCs\MOC Master Python Análisis de Datos\Datasets\6.9\S&P500 v2.xlsx')
```

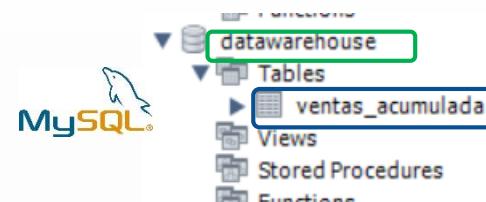
# Exportación de datos a BBDD SQL

- ¿Cómo exportar un dataframe a una tabla de la BBDD SQL?

`df.to_sql(name="nombre_tabla_en_BBDD", con = objeto_conexión)`

***df\_sp500***

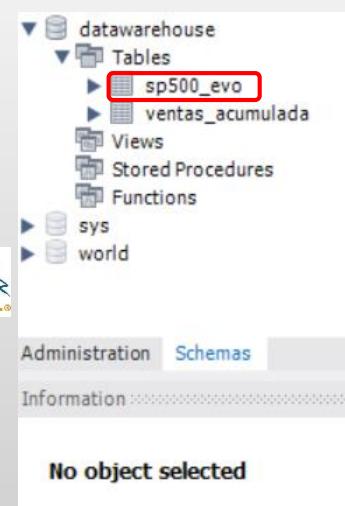
	Date	Open	High	Low	Close	Adj Close	Volume	Variación_dia
0	1927-12-30	17.660000	17.660000	17.660000	17.660000	17.660000	0	0.000000
1	1928-01-03	17.760000	17.760000	17.760000	17.760000	17.760000	0	0.000000
2	1928-01-04	17.719999	17.719999	17.719999	17.719999	17.719999	0	0.000000
3	1928-01-05	17.549999	17.549999	17.549999	17.549999	17.549999	0	0.000000
4	1928-01-06	17.660000	17.660000	17.660000	17.660000	17.660000	0	0.000000
...	...	...	...	...	...	...	...	...
23186	2020-04-23	2810.419922	2844.899902	2794.260010	2797.800049	2797.800049	5756520000	-0.449039
23187	2020-04-24	2812.639893	2842.709961	2791.760010	2836.739990	2836.739990	5374480000	0.856850
23188	2020-04-27	2854.649902	2887.719971	2852.889893	2878.479980	2878.479980	5194260000	0.834781
23189	2020-04-28	2909.959961	2921.149902	2860.709961	2863.389893	2863.389893	5672880000	-1.600368
23190	2020-04-29	2918.459961	2954.860107	2912.159912	2939.510010	2939.510010	6620140000	0.721272



```
In [23]: from sqlalchemy import create_engine
import pymysql
```

```
[24]: db_conexion = create_engine('mysql+pymysql://root:xxxxxx@127.0.0.1/datawarehouse')
```

```
In [43]: df_sp500.to_sql(name="sp500_evo", con=db_conexion)
```



Result Grid | Filter Rows: Export: Wrap Cell Content: Fetch rows:

index	Date	Open	High	Low	Close	Adj Close	Volume	Variación_dia
0	1927-12-30	17.66	17.66	17.66	17.66	17.66	0	0
1	1928-01-03	17.76	17.76	17.76	17.76	17.76	0	0
2	1928-01-04	17.719999	17.719999	17.719999	17.719999	17.719999	0	0
3	1928-01-05	17.549999	17.549999	17.549999	17.549999	17.549999	0	0
4	1928-01-06	17.66	17.66	17.66	17.66	17.66	0	0
5	1928-01-09	17.5	17.5	17.5	17.5	17.5	0	0
6	1928-01-10	17.370001	17.370001	17.370001	17.370001	17.370001	0	0
7	1928-01-11	17.35	17.35	17.35	17.35	17.35	0	0
8	1928-01-12	17.469999	17.469999	17.469999	17.469999	17.469999	0	0
9	1928-01-13	17.58	17.58	17.58	17.58	17.58	0	0
10	1928-01-16	17.290001	17.290001	17.290001	17.290001	17.290001	0	0
11	1928-01-17	17.299999	17.299999	17.299999	17.299999	17.299999	0	0

No object selected

# CASO PRÁCTICO 1

**Tabla Clientes**

ID Cliente
Nombre cliente
Segmento
País
Ciudad
Estado
Código Postal
Región

**Tabla Ventas**

ID Pedido
Fecha compra
Fecha envío
Modo envío
ID Cliente
ID Producto
Precio unitario
Cantidad

**Tabla Productos**

ID Producto
Categoría
Sub-categoría
Nombre Producto
Coste Producción