

2.1 Bibliotecas clave: Pandas y NumPy

June 13, 2025

Creado por:

Isabel Maniega

1 Pandas. Temario

1. Vista de los datos
2. Selección
3. Setting
4. Missing values
5. Operaciones
6. Unión de dataframe
7. Grouping
8. Reshaping
9. Time Series
10. Categoricals
11. Plotting

2 Pandas

Contiene dos tipos de estructuras:

- **Series**: una matriz etiquetada unidimensional que contiene datos de cualquier tipo como números enteros, cadenas, objetos Python, etc.
- **Dataframe**: una estructura de datos bidimensional que contiene datos como una matriz bidimensional o una tabla con filas y columnas.

```
[1]: # pip install pandas
```

```
[2]: from IPython import display
```

```
[3]: import pandas as pd
import numpy as np
```

2.1 Creación de un dataframe

`pd.DataFrame()`

- A partir de Series:

```
[4]: # Series:

s = pd.Series([1, 3, 5, np.nan, 6, 8])
s
```

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

```
[5]: # date_range(genera un rango de fecha a partir de un valor, marcando el número
     ↪ de datos a generar (periods)
     dates = pd.date_range("20130101", periods=6)
     dates
```

```
[5]: DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
                    '2013-01-05', '2013-01-06'],
                    dtype='datetime64[ns]', freq='D')
```

```
[6]: df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list("ABCD"))
     df
```

```
[6]:
```

	A	B	C	D
2013-01-01	0.135994	1.445202	-1.366478	1.245896
2013-01-02	-0.115987	-1.248233	0.968112	0.860303
2013-01-03	-0.311543	0.919419	0.827101	-0.991637
2013-01-04	0.691521	-0.473381	0.393770	-1.389059
2013-01-05	-0.778760	0.701268	-2.180317	-2.793037
2013-01-06	-1.028239	2.430918	0.076173	0.470310

```
[7]: # dtypes nos muestra de que tipo son los datos:

df.dtypes
```

```
[7]: A    float64
     B    float64
     C    float64
     D    float64
     dtype: object
```

- A partir de un diccionario:

```
[8]: notas = [10, 9, 5, 4, 8]

alumnos = ["Paula", "Fermin", "Pedro", "Luis", "Ana"]
```

```

evaluacion = {"Alumnos": alumnos, "Notas": notas}

df_eval = pd.DataFrame(evaluacion)
df_eval

```

```

[8]:   Alumnos  Notas
0    Paula    10
1   Fermin     9
2    Pedro     5
3     Luis     4
4     Ana      8

```

- A partir de listas:

```

[9]: df_evaluar = pd.DataFrame(alumnos, columns=["Alumnos"])
df_evaluar

```

```

[9]:   Alumnos
0    Paula
1   Fermin
2    Pedro
3     Luis
4     Ana

```

Otra forma...

```

[10]: df_evaluar = pd.DataFrame(list(zip(alumnos, notas)), columns=["Alumnos", "Notas"])
df_evaluar

```

```

[10]:   Alumnos  Notas
0    Paula    10
1   Fermin     9
2    Pedro     5
3     Luis     4
4     Ana      8

```

2.2 Vista de los datos

```

[11]: # Muestra las primeras filas del dataframe, por defecto las 5 primeras
df.head()

```

```

[11]:           A           B           C           D
2013-01-01  0.135994  1.445202 -1.366478  1.245896
2013-01-02 -0.115987 -1.248233  0.968112  0.860303
2013-01-03 -0.311543  0.919419  0.827101 -0.991637
2013-01-04  0.691521 -0.473381  0.393770 -1.389059

```

```
2013-01-05 -0.778760  0.701268 -2.180317 -2.793037
```

```
[12]: df.head(2)
```

```
[12]:
```

	A	B	C	D
2013-01-01	0.135994	1.445202	-1.366478	1.245896
2013-01-02	-0.115987	-1.248233	0.968112	0.860303

```
[13]: # Muestra las últimas filas de un dataframe, por defecto las 5 últimas:
```

```
df.tail()
```

```
[13]:
```

	A	B	C	D
2013-01-02	-0.115987	-1.248233	0.968112	0.860303
2013-01-03	-0.311543	0.919419	0.827101	-0.991637
2013-01-04	0.691521	-0.473381	0.393770	-1.389059
2013-01-05	-0.778760	0.701268	-2.180317	-2.793037
2013-01-06	-1.028239	2.430918	0.076173	0.470310

```
[14]: df.tail(2)
```

```
[14]:
```

	A	B	C	D
2013-01-05	-0.778760	0.701268	-2.180317	-2.793037
2013-01-06	-1.028239	2.430918	0.076173	0.470310

```
[15]: # Muestra el valor de la primera columna que suele ser un valor único (id), en
```

```
↪este ejemplo una fecha:
```

```
df.index
```

```
[15]: DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',  
                    '2013-01-05', '2013-01-06'],  
                  dtype='datetime64[ns]', freq='D')
```

```
[16]: # Muestra el nombre de las columnas:
```

```
df.columns
```

```
[16]: Index(['A', 'B', 'C', 'D'], dtype='object')
```

```
[17]: # Podemos convertir un dataframe en una matriz de numpy con:
```

```
df.to_numpy()
```

```
[17]: array([[ 0.1359942 ,  1.44520206, -1.3664783 ,  1.24589643],  
          [-0.11598682, -1.24823285,  0.96811244,  0.86030334],  
          [-0.31154284,  0.91941923,  0.82710052, -0.99163685],  
          [ 0.6915211 , -0.47338116,  0.3937703 , -1.38905907],
```

```
[-0.77875952, 0.701268, -2.18031699, -2.79303687],
[-1.02823858, 2.43091785, 0.07617329, 0.47030963]])
```

[18]: *# Para obtener los estadísticos más representativos usamos:*

```
df.describe()
```

```
[18]:
```

	A	B	C	D
count	6.000000	6.000000	6.000000	6.000000
mean	-0.234502	0.629199	-0.213606	-0.432871
std	0.623057	1.322264	1.275106	1.555840
min	-1.028239	-1.248233	-2.180317	-2.793037
25%	-0.661955	-0.179719	-1.005815	-1.289704
50%	-0.213765	0.810344	0.234972	-0.260664
75%	0.072999	1.313756	0.718768	0.762805
max	0.691521	2.430918	0.968112	1.245896

[19]: *# Podemos dar la vuelta a la tabla y poner lo que esta en filas en columnas y ↵
↪viceversa:*

```
df.T
```

```
[19]:
```

	2013-01-01	2013-01-02	2013-01-03	2013-01-04	2013-01-05	2013-01-06
A	0.135994	-0.115987	-0.311543	0.691521	-0.778760	-1.028239
B	1.445202	-1.248233	0.919419	-0.473381	0.701268	2.430918
C	-1.366478	0.968112	0.827101	0.393770	-2.180317	0.076173
D	1.245896	0.860303	-0.991637	-1.389059	-2.793037	0.470310

[20]: *# Colocar los valores según el índice:*

```
df.sort_index(axis=1, ascending=False)
```

```
[20]:
```

	D	C	B	A
2013-01-01	1.245896	-1.366478	1.445202	0.135994
2013-01-02	0.860303	0.968112	-1.248233	-0.115987
2013-01-03	-0.991637	0.827101	0.919419	-0.311543
2013-01-04	-1.389059	0.393770	-0.473381	0.691521
2013-01-05	-2.793037	-2.180317	0.701268	-0.778760
2013-01-06	0.470310	0.076173	2.430918	-1.028239

[21]: *# Ordenar los datos según una columna:*

```
df.sort_values(by="B")
```

```
[21]:
```

	A	B	C	D
2013-01-02	-0.115987	-1.248233	0.968112	0.860303
2013-01-04	0.691521	-0.473381	0.393770	-1.389059
2013-01-05	-0.778760	0.701268	-2.180317	-2.793037

```
2013-01-03 -0.311543  0.919419  0.827101 -0.991637
2013-01-01  0.135994  1.445202 -1.366478  1.245896
2013-01-06 -1.028239  2.430918  0.076173  0.470310
```

2.3 Selecccion

2.4 GetItem()

Selección de columna. Existen 3 formas de seleccionar una columna:

```
[22]: df['A']
```

```
[22]: 2013-01-01    0.135994
      2013-01-02   -0.115987
      2013-01-03   -0.311543
      2013-01-04    0.691521
      2013-01-05   -0.778760
      2013-01-06   -1.028239
      Freq: D, Name: A, dtype: float64
```

```
[23]: df.A
```

```
[23]: 2013-01-01    0.135994
      2013-01-02   -0.115987
      2013-01-03   -0.311543
      2013-01-04    0.691521
      2013-01-05   -0.778760
      2013-01-06   -1.028239
      Freq: D, Name: A, dtype: float64
```

```
[24]: df[['A']]
```

```
[24]:              A
      2013-01-01  0.135994
      2013-01-02 -0.115987
      2013-01-03 -0.311543
      2013-01-04  0.691521
      2013-01-05 -0.778760
      2013-01-06 -1.028239
```

Selección de filas mediante slicing(:)

```
[25]: df[0:2]
```

```
[25]:              A          B          C          D
      2013-01-01  0.135994  1.445202 -1.366478  1.245896
      2013-01-02 -0.115987 -1.248233  0.968112  0.860303
```

```
[26]: df["20130103":"20130105"]
```

```
[26]:
```

	A	B	C	D
2013-01-03	-0.311543	0.919419	0.827101	-0.991637
2013-01-04	0.691521	-0.473381	0.393770	-1.389059
2013-01-05	-0.778760	0.701268	-2.180317	-2.793037

Selección con la función loc[] y at[]

```
[27]: # Filas que coinciden con una etiqueta, selección de la primera fila:

df.loc[dates[0]]
```

```
[27]: A    0.135994
      B    1.445202
      C   -1.366478
      D    1.245896
      Name: 2013-01-01 00:00:00, dtype: float64
```

```
[28]: # Seleccionar todas las filas de una determinada columna:

df.loc[:, ['B', 'C']]
```

```
[28]:
```

	B	C
2013-01-01	1.445202	-1.366478
2013-01-02	-1.248233	0.968112
2013-01-03	0.919419	0.827101
2013-01-04	-0.473381	0.393770
2013-01-05	0.701268	-2.180317
2013-01-06	2.430918	0.076173

```
[29]: # Seleccionar por filas y columnas:

df.loc["20130103":"20130105", ['B', 'C']]
```

```
[29]:
```

	B	C
2013-01-03	0.919419	0.827101
2013-01-04	-0.473381	0.393770
2013-01-05	0.701268	-2.180317

```
[30]: # Seleccionar para un valor determinado -0.891699 (20130103, B):

df.loc[dates[2], 'B']
```

```
[30]: np.float64(0.9194192259843359)
```

```
[31]: df.at[dates[2], 'B']
```

```
[31]: np.float64(0.9194192259843359)
```

Selección por posición: método iloc[] y iat[]

```
[32]: # Selección de una fila en posición 3:
```

```
df.iloc[3]
```

```
[32]: A    0.691521  
      B   -0.473381  
      C    0.393770  
      D   -1.389059  
      Name: 2013-01-04 00:00:00, dtype: float64
```

```
[33]: # Selección de una fila y columna por slicing:
```

```
df.iloc[3:5, 1:3]
```

```
[33]:           B          C  
2013-01-04 -0.473381  0.393770  
2013-01-05  0.701268 -2.180317
```

```
[34]: # Selección por lista de posiciones:
```

```
# Filas: 1, 2, 4
```

```
# Columnas: 0(A), 2(C)
```

```
df.iloc[[1, 2, 4], [0, 2]]
```

```
[34]:           A          C  
2013-01-02 -0.115987  0.968112  
2013-01-03 -0.311543  0.827101  
2013-01-05 -0.778760 -2.180317
```

```
[35]: # Selección por filas o columnas:
```

```
df.iloc[1:3, :]
```

```
[35]:           A          B          C          D  
2013-01-02 -0.115987 -1.248233  0.968112  0.860303  
2013-01-03 -0.311543  0.919419  0.827101 -0.991637
```

```
[36]: df.iloc[:, 1:3]
```

```
[36]:           B          C  
2013-01-01  1.445202 -1.366478  
2013-01-02 -1.248233  0.968112  
2013-01-03  0.919419  0.827101  
2013-01-04 -0.473381  0.393770  
2013-01-05  0.701268 -2.180317  
2013-01-06  2.430918  0.076173
```



```
[37]: # Seleccionar un valor concreto por posición (2013-01-03, 'B'):
df.iloc[2, 1]
```

```
[37]: np.float64(0.9194192259843359)
```

```
[38]: df.iat[2, 1]
```

```
[38]: np.float64(0.9194192259843359)
```

2.5 Boolean indexing

```
[39]: # Selección por comparativa:
df[df['A'] >= 0.2]
```

```
[39]:
```

	A	B	C	D
2013-01-04	0.691521	-0.473381	0.39377	-1.389059

Método isin()

```
[40]: # Selección según una coincidencia (filtrado):

df2 = pd.DataFrame(["one", "one", "two", "three", "four", "three"],
                    columns=['E'])

df2[df2["E"].isin(["one", "four"])]
```

```
[40]:
```

	E
0	one
1	one
4	four

2.6 Setting (Modificación del dataframe)

```
[41]: # Añadir Valores nuevo

serie = pd.Series([1, 2, 3, 4, 5, 6], index=pd.date_range("20130101",
                periods=6))
serie
```

```
[41]:
```

2013-01-01	1
2013-01-02	2
2013-01-03	3
2013-01-04	4
2013-01-05	5
2013-01-06	6

Freq: D, dtype: int64

```
[42]: df['E'] = serie
df
```

```
[42]:
```

	A	B	C	D	E
2013-01-01	0.135994	1.445202	-1.366478	1.245896	1
2013-01-02	-0.115987	-1.248233	0.968112	0.860303	2
2013-01-03	-0.311543	0.919419	0.827101	-0.991637	3
2013-01-04	0.691521	-0.473381	0.393770	-1.389059	4
2013-01-05	-0.778760	0.701268	-2.180317	-2.793037	5
2013-01-06	-1.028239	2.430918	0.076173	0.470310	6

```
[43]: # Modificar valor por etiqueta
# Se modifica el primer valor de df por 0 en la columna A:

df.at[dates[0], "A"] = 0
df
```

```
[43]:
```

	A	B	C	D	E
2013-01-01	0.000000	1.445202	-1.366478	1.245896	1
2013-01-02	-0.115987	-1.248233	0.968112	0.860303	2
2013-01-03	-0.311543	0.919419	0.827101	-0.991637	3
2013-01-04	0.691521	-0.473381	0.393770	-1.389059	4
2013-01-05	-0.778760	0.701268	-2.180317	-2.793037	5
2013-01-06	-1.028239	2.430918	0.076173	0.470310	6

```
[44]: # Modificación de valor por posición
# Se modifica el primer valor de la columna B:

df.iat[0, 1] = 0
df
```

```
[44]:
```

	A	B	C	D	E
2013-01-01	0.000000	0.000000	-1.366478	1.245896	1
2013-01-02	-0.115987	-1.248233	0.968112	0.860303	2
2013-01-03	-0.311543	0.919419	0.827101	-0.991637	3
2013-01-04	0.691521	-0.473381	0.393770	-1.389059	4
2013-01-05	-0.778760	0.701268	-2.180317	-2.793037	5
2013-01-06	-1.028239	2.430918	0.076173	0.470310	6

```
[45]: # Modificación asignada por Numpy usando array:

df.loc[:, "D"] = np.array([5] * len(df))
df
```

```
[45]:
```

	A	B	C	D	E
2013-01-01	0.000000	0.000000	-1.366478	5.0	1
2013-01-02	-0.115987	-1.248233	0.968112	5.0	2
2013-01-03	-0.311543	0.919419	0.827101	5.0	3

2013-01-04	0.691521	-0.473381	0.393770	5.0	4
2013-01-05	-0.778760	0.701268	-2.180317	5.0	5
2013-01-06	-1.028239	2.430918	0.076173	5.0	6

```
[46]: # Modificar según una condición (where):

df2 = df.copy() # Realización de una copia del df

df2[df2 > 0.1] = -df2
df2
```

```
[46]:
```

	A	B	C	D	E
2013-01-01	0.000000	0.000000	-1.366478	-5.0	-1
2013-01-02	-0.115987	-1.248233	-0.968112	-5.0	-2
2013-01-03	-0.311543	-0.919419	-0.827101	-5.0	-3
2013-01-04	-0.691521	-0.473381	-0.393770	-5.0	-4
2013-01-05	-0.778760	-0.701268	-2.180317	-5.0	-5
2013-01-06	-1.028239	-2.430918	0.076173	-5.0	-6

2.7 Missing values

```
[47]: # Creamos una columna nueva con valores nulos:

df1 = df.reindex(index=dates[0:4], columns=list(df.columns))

df1.loc[dates[2]:dates[3], "E"] = np.nan
df1.at[dates[0], "D"] = np.nan

print(df1)
```

	A	B	C	D	E
2013-01-01	0.000000	0.000000	-1.366478	NaN	1.0
2013-01-02	-0.115987	-1.248233	0.968112	5.0	2.0
2013-01-03	-0.311543	0.919419	0.827101	5.0	NaN
2013-01-04	0.691521	-0.473381	0.393770	5.0	NaN

```
[48]: # Eliminamos los valores nulos con la función dropna(): eliminando cualquier
      ↪ fila que contenga valores nulos

df_1 = df1.dropna(how="any")
df_1
```

```
[48]:
```

	A	B	C	D	E
2013-01-02	-0.115987	-1.248233	0.968112	5.0	2.0

```
[49]: # Rellenar valores nulos:

df_1 = df1.fillna(value=5)
```

```
df_1
```

```
[49]:
```

	A	B	C	D	E
2013-01-01	0.000000	0.000000	-1.366478	5.0	1.0
2013-01-02	-0.115987	-1.248233	0.968112	5.0	2.0
2013-01-03	-0.311543	0.919419	0.827101	5.0	5.0
2013-01-04	0.691521	-0.473381	0.393770	5.0	5.0

```
[50]: # isna() nos muestra si en el df hay valores nulo o no, sustituyendo por un  
      ↪ booleano (True / False)
```

```
pd.isna(df1)
```

```
[50]:
```

	A	B	C	D	E
2013-01-01	False	False	False	True	False
2013-01-02	False	False	False	False	False
2013-01-03	False	False	False	False	True
2013-01-04	False	False	False	False	True

2.8 Operaciones

En estos casos no tiene en cuenta los valores nulos.

```
[51]: df = pd.DataFrame({"notas_1": [15, 16, 15, 17, 14, 14, 14, 10, 15, 25],  
                        "notas_2": [16, 21, 16, 16, 13, 15, 15, 19, 22, 15],  
                        "notas_3": [17, 22, 15, 22, 14, 15, 16, 15, 24, 16]})  
df.head()
```

```
[51]:
```

	notas_1	notas_2	notas_3
0	15	16	17
1	16	21	22
2	15	16	15
3	17	16	22
4	14	13	14

2.8.1 Tendencia Central

Media

Como calcular la media de las distintas notas:

```
[52]: media_1 = df["notas_1"].mean()  
media_1
```

```
[52]: np.float64(15.5)
```

```
[53]: media_2 = df["notas_2"].mean()  
media_2
```

```
[53]: np.float64(16.8)
```

```
[54]: media_3 = df["notas_3"].mean()  
media_3
```

```
[54]: np.float64(17.6)
```

Mediana

Como calcular la mediana de las distintas notas:

```
[55]: mediana_1 = df["notas_1"].median()  
mediana_1
```

```
[55]: np.float64(15.0)
```

```
[56]: mediana_2 = df["notas_2"].median()  
mediana_2
```

```
[56]: np.float64(16.0)
```

```
[57]: mediana_3 = df["notas_3"].median()  
mediana_3
```

```
[57]: np.float64(16.0)
```

Moda

Como calcular la moda de las distintas notas:

```
[58]: moda_1 = df["notas_1"].mode()  
moda_1
```

```
[58]: 0    14  
      1    15  
      Name: notas_1, dtype: int64
```

```
[59]: moda_2 = df["notas_2"].mode()  
moda_2
```

```
[59]: 0    15  
      1    16  
      Name: notas_2, dtype: int64
```

```
[60]: moda_3 = df["notas_3"].mode()  
moda_3
```

```
[60]: 0    15  
      Name: notas_3, dtype: int64
```

```
[61]: df.notas_3.value_counts()
```

```
[61]: notas_3
      15      3
      22      2
      16      2
      17      1
      14      1
      24      1
      Name: count, dtype: int64
```

Resultados Nota_1:

```
[62]: print(f"Media: {media_1}, Mediana: {mediana_1}, Moda: \n{moda_1}")
```

```
Media: 15.5, Mediana: 15.0, Moda:
0      14
1      15
      Name: notas_1, dtype: int64
```

Resultados Nota_2:

```
[63]: print(f"Media: {media_2}, Mediana: {mediana_2}, Moda: \n{moda_2}")
```

```
Media: 16.8, Mediana: 16.0, Moda:
0      15
1      16
      Name: notas_2, dtype: int64
```

Resultados Nota_2:

```
[64]: print(f"Media: {media_3}, Mediana: {mediana_3}, Moda: \n{moda_3}")
```

```
Media: 17.6, Mediana: 16.0, Moda:
0      15
      Name: notas_3, dtype: int64
```

Varianza

Se calcula la cuasi-varianza:

$$S^2 = \frac{\sum_{i=1}^n (x_i - X)^2}{n - 1}$$

```
[65]: var_1 = df["notas_1"].var()
      var_1
```

```
[65]: np.float64(14.5)
```

```
[66]: var_2 = df["notas_2"].var()
      var_2
```

```
[66]: np.float64(8.399999999999999)
```

```
[67]: var_3 = df["notas_3"].var()
      var_3
```

```
[67]: np.float64(13.155555555555557)
```

Si queremos calcular la varianza, utilizamos el argumento ddof=0. El denominador en la fórmula será entonces n-ddof=0:

```
[68]: var_1 = df["notas_1"].var(ddof=0)
      var_1
```

```
[68]: np.float64(13.05)
```

Desviación típica

En python, utilizamos el método .std() para calcular la cuasi-desviación típica. Para calcular la desviación típica, nuevamente utilizamos ddof=0.

$$S = \sqrt{S^2}$$

```
[69]: std_1 = df["notas_1"].std()
      std_1
```

```
[69]: np.float64(3.8078865529319543)
```

```
[70]: std_2 = df["notas_2"].std()
      std_2
```

```
[70]: np.float64(2.8982753492378874)
```

```
[71]: std_3 = df["notas_3"].std()
      std_3
```

```
[71]: np.float64(3.6270588023294517)
```

Si queremos calcular la varianza, utilizamos el argumento ddof=0. El denominador en la fórmula será entonces n-ddof=0:

```
[72]: std_1 = df["notas_1"].std(ddof=0)
      std_1
```

```
[72]: np.float64(3.6124783736376886)
```

Máximo y mínimo

```
[73]: max_1 = df["notas_1"].max()
      min_1 = df["notas_1"].min()
      print(max_1, min_1)
```

25 10

```
[74]: max_2 = df["notas_2"].max()
      min_2 = df["notas_2"].min()
      print(max_2, min_2)
```

22 13

```
[75]: max_3 = df["notas_3"].max()
      min_3 = df["notas_3"].min()
      print(max_3, min_3)
```

24 14

3 RESUMEN

Notas 1 Notas 2 Notas 3Media 15.5 16.8 17.6Mediana 15.0 16.0 16.0Moda 14/15 15/16 15.0std 3.807 2.90 3.63max 25 22 24min 10 13 14

```
[76]: df.describe()
```

```
[76]:
```

	notas_1	notas_2	notas_3
count	10.000000	10.000000	10.000000
mean	15.500000	16.800000	17.600000
std	3.807887	2.898275	3.627059
min	10.000000	13.000000	14.000000
25%	14.000000	15.000000	15.000000
50%	15.000000	16.000000	16.000000
75%	15.750000	18.250000	20.750000
max	25.000000	22.000000	24.000000

3.1 Union de dataframe

```
[77]: iris = pd.read_csv('./files/Iris.csv')
      iris = iris.drop(['Id'], axis=1)
      iris_setosa = iris[0:50]
      iris_setosa
```

```
[77]:
```

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

9	4.9	3.1	1.5	0.1	Iris-setosa
10	5.4	3.7	1.5	0.2	Iris-setosa
11	4.8	3.4	1.6	0.2	Iris-setosa
12	4.8	3.0	1.4	0.1	Iris-setosa
13	4.3	3.0	1.1	0.1	Iris-setosa
14	5.8	4.0	1.2	0.2	Iris-setosa
15	5.7	4.4	1.5	0.4	Iris-setosa
16	5.4	3.9	1.3	0.4	Iris-setosa
17	5.1	3.5	1.4	0.3	Iris-setosa
18	5.7	3.8	1.7	0.3	Iris-setosa
19	5.1	3.8	1.5	0.3	Iris-setosa
20	5.4	3.4	1.7	0.2	Iris-setosa
21	5.1	3.7	1.5	0.4	Iris-setosa
22	4.6	3.6	1.0	0.2	Iris-setosa
23	5.1	3.3	1.7	0.5	Iris-setosa
24	4.8	3.4	1.9	0.2	Iris-setosa
25	5.0	3.0	1.6	0.2	Iris-setosa
26	5.0	3.4	1.6	0.4	Iris-setosa
27	5.2	3.5	1.5	0.2	Iris-setosa
28	5.2	3.4	1.4	0.2	Iris-setosa
29	4.7	3.2	1.6	0.2	Iris-setosa
30	4.8	3.1	1.6	0.2	Iris-setosa
31	5.4	3.4	1.5	0.4	Iris-setosa
32	5.2	4.1	1.5	0.1	Iris-setosa
33	5.5	4.2	1.4	0.2	Iris-setosa
34	4.9	3.1	1.5	0.1	Iris-setosa
35	5.0	3.2	1.2	0.2	Iris-setosa
36	5.5	3.5	1.3	0.2	Iris-setosa
37	4.9	3.1	1.5	0.1	Iris-setosa
38	4.4	3.0	1.3	0.2	Iris-setosa
39	5.1	3.4	1.5	0.2	Iris-setosa
40	5.0	3.5	1.3	0.3	Iris-setosa
41	4.5	2.3	1.3	0.3	Iris-setosa
42	4.4	3.2	1.3	0.2	Iris-setosa
43	5.0	3.5	1.6	0.6	Iris-setosa
44	5.1	3.8	1.9	0.4	Iris-setosa
45	4.8	3.0	1.4	0.3	Iris-setosa
46	5.1	3.8	1.6	0.2	Iris-setosa
47	4.6	3.2	1.4	0.2	Iris-setosa
48	5.3	3.7	1.5	0.2	Iris-setosa
49	5.0	3.3	1.4	0.2	Iris-setosa

```
[78]: iris_virginica = iris[100:]
iris_virginica
```

```
[78]:      SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
100              6.3            3.3            6.0            2.5  Iris-virginica
```

101	5.8	2.7	5.1	1.9	Iris-virginica
102	7.1	3.0	5.9	2.1	Iris-virginica
103	6.3	2.9	5.6	1.8	Iris-virginica
104	6.5	3.0	5.8	2.2	Iris-virginica
105	7.6	3.0	6.6	2.1	Iris-virginica
106	4.9	2.5	4.5	1.7	Iris-virginica
107	7.3	2.9	6.3	1.8	Iris-virginica
108	6.7	2.5	5.8	1.8	Iris-virginica
109	7.2	3.6	6.1	2.5	Iris-virginica
110	6.5	3.2	5.1	2.0	Iris-virginica
111	6.4	2.7	5.3	1.9	Iris-virginica
112	6.8	3.0	5.5	2.1	Iris-virginica
113	5.7	2.5	5.0	2.0	Iris-virginica
114	5.8	2.8	5.1	2.4	Iris-virginica
115	6.4	3.2	5.3	2.3	Iris-virginica
116	6.5	3.0	5.5	1.8	Iris-virginica
117	7.7	3.8	6.7	2.2	Iris-virginica
118	7.7	2.6	6.9	2.3	Iris-virginica
119	6.0	2.2	5.0	1.5	Iris-virginica
120	6.9	3.2	5.7	2.3	Iris-virginica
121	5.6	2.8	4.9	2.0	Iris-virginica
122	7.7	2.8	6.7	2.0	Iris-virginica
123	6.3	2.7	4.9	1.8	Iris-virginica
124	6.7	3.3	5.7	2.1	Iris-virginica
125	7.2	3.2	6.0	1.8	Iris-virginica
126	6.2	2.8	4.8	1.8	Iris-virginica
127	6.1	3.0	4.9	1.8	Iris-virginica
128	6.4	2.8	5.6	2.1	Iris-virginica
129	7.2	3.0	5.8	1.6	Iris-virginica
130	7.4	2.8	6.1	1.9	Iris-virginica
131	7.9	3.8	6.4	2.0	Iris-virginica
132	6.4	2.8	5.6	2.2	Iris-virginica
133	6.3	2.8	5.1	1.5	Iris-virginica
134	6.1	2.6	5.6	1.4	Iris-virginica
135	7.7	3.0	6.1	2.3	Iris-virginica
136	6.3	3.4	5.6	2.4	Iris-virginica
137	6.4	3.1	5.5	1.8	Iris-virginica
138	6.0	3.0	4.8	1.8	Iris-virginica
139	6.9	3.1	5.4	2.1	Iris-virginica
140	6.7	3.1	5.6	2.4	Iris-virginica
141	6.9	3.1	5.1	2.3	Iris-virginica
142	5.8	2.7	5.1	1.9	Iris-virginica
143	6.8	3.2	5.9	2.3	Iris-virginica
144	6.7	3.3	5.7	2.5	Iris-virginica
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica

148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

```
[79]: iris_versicolor = pd.read_json('./files/iris_versicolor.json')
iris_versicolor
```

```
[79]:
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	7.0	3.2	4.7	1.4	Iris-versicolor
1	6.4	3.2	4.5	1.5	Iris-versicolor
2	6.9	3.1	4.9	1.5	Iris-versicolor
3	5.5	2.3	4.0	1.3	Iris-versicolor
4	6.5	2.8	4.6	1.5	Iris-versicolor
5	5.7	2.8	4.5	1.3	Iris-versicolor
6	6.3	3.3	4.7	1.6	Iris-versicolor
7	4.9	2.4	3.3	1.0	Iris-versicolor
8	6.6	2.9	4.6	1.3	Iris-versicolor
9	5.2	2.7	3.9	1.4	Iris-versicolor
10	5.0	2.0	3.5	1.0	Iris-versicolor
11	5.9	3.0	4.2	1.5	Iris-versicolor
12	6.0	2.2	4.0	1.0	Iris-versicolor
13	6.1	2.9	4.7	1.4	Iris-versicolor
14	5.6	2.9	3.6	1.3	Iris-versicolor
15	6.7	3.1	4.4	1.4	Iris-versicolor
16	5.6	3.0	4.5	1.5	Iris-versicolor
17	5.8	2.7	4.1	1.0	Iris-versicolor
18	6.2	2.2	4.5	1.5	Iris-versicolor
19	5.6	2.5	3.9	1.1	Iris-versicolor
20	5.9	3.2	4.8	1.8	Iris-versicolor
21	6.1	2.8	4.0	1.3	Iris-versicolor
22	6.3	2.5	4.9	1.5	Iris-versicolor
23	6.1	2.8	4.7	1.2	Iris-versicolor
24	6.4	2.9	4.3	1.3	Iris-versicolor
25	6.6	3.0	4.4	1.4	Iris-versicolor
26	6.8	2.8	4.8	1.4	Iris-versicolor
27	6.7	3.0	5.0	1.7	Iris-versicolor
28	6.0	2.9	4.5	1.5	Iris-versicolor
29	5.7	2.6	3.5	1.0	Iris-versicolor
30	5.5	2.4	3.8	1.1	Iris-versicolor
31	5.5	2.4	3.7	1.0	Iris-versicolor
32	5.8	2.7	3.9	1.2	Iris-versicolor
33	6.0	2.7	5.1	1.6	Iris-versicolor
34	5.4	3.0	4.5	1.5	Iris-versicolor
35	6.0	3.4	4.5	1.6	Iris-versicolor
36	6.7	3.1	4.7	1.5	Iris-versicolor
37	6.3	2.3	4.4	1.3	Iris-versicolor
38	5.6	3.0	4.1	1.3	Iris-versicolor
39	5.5	2.5	4.0	1.3	Iris-versicolor

40	5.5	2.6	4.4	1.2	Iris-versicolor
41	6.1	3.0	4.6	1.4	Iris-versicolor
42	5.8	2.6	4.0	1.2	Iris-versicolor
43	5.0	2.3	3.3	1.0	Iris-versicolor
44	5.6	2.7	4.2	1.3	Iris-versicolor
45	5.7	3.0	4.2	1.2	Iris-versicolor
46	5.7	2.9	4.2	1.3	Iris-versicolor
47	6.2	2.9	4.3	1.3	Iris-versicolor
48	5.1	2.5	3.0	1.1	Iris-versicolor
49	5.7	2.8	4.1	1.3	Iris-versicolor

3.2 concat()

[80]: *# Unión de varios dataframe por nombre de columna, los apendiza al final:*

```
dfs = [iris_setosa, iris_virginica, iris_versicolor]
iris_concat = pd.concat(dfs)
iris_concat
```

[80]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
..
45	5.7	3.0	4.2	1.2	Iris-versicolor
46	5.7	2.9	4.2	1.3	Iris-versicolor
47	6.2	2.9	4.3	1.3	Iris-versicolor
48	5.1	2.5	3.0	1.1	Iris-versicolor
49	5.7	2.8	4.1	1.3	Iris-versicolor

[150 rows x 5 columns]

[81]: `display.Image('./images/merging_concat_basic.png')`

[81]:

df1					Result				
	A	B	C	D		A	B	C	D
0	A0	B0	C0	D0	0	A0	B0	C0	D0
1	A1	B1	C1	D1	1	A1	B1	C1	D1
2	A2	B2	C2	D2	2	A2	B2	C2	D2
3	A3	B3	C3	D3	3	A3	B3	C3	D3
df2					4	A4	B4	C4	D4
	A	B	C	D	5	A5	B5	C5	D5
4	A4	B4	C4	D4	6	A6	B6	C6	D6
5	A5	B5	C5	D5	7	A7	B7	C7	D7
6	A6	B6	C6	D6	8	A8	B8	C8	D8
7	A7	B7	C7	D7	9	A9	B9	C9	D9
df3					10	A10	B10	C10	D10
	A	B	C	D	11	A11	B11	C11	D11
8	A8	B8	C8	D8					
9	A9	B9	C9	D9					
10	A10	B10	C10	D10					
11	A11	B11	C11	D11					

```
[82]: iris = pd.read_csv('./files/Iris.csv')
iris_medidas = iris.iloc[:, 0:4]
iris_medidas
```

```
[82]:
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm
0	1	5.1	3.5	1.4
1	2	4.9	3.0	1.4
2	3	4.7	3.2	1.3
3	4	4.6	3.1	1.5
4	5	5.0	3.6	1.4
..
145	146	6.7	3.0	5.2
146	147	6.3	2.5	5.0
147	148	6.5	3.0	5.2
148	149	6.2	3.4	5.4
149	150	5.9	3.0	5.1

[150 rows x 4 columns]

```
[83]: iris_especies = iris[['Species']]
iris_especies
```

```
[83]:      Species
0      Iris-setosa
1      Iris-setosa
2      Iris-setosa
3      Iris-setosa
4      Iris-setosa
..      ...
145    Iris-virginica
146    Iris-virginica
147    Iris-virginica
148    Iris-virginica
149    Iris-virginica
```

[150 rows x 1 columns]

```
[84]: # Apendizar una columna nueva usando concat:
# axis=1 elegimos el eje
# join='inner' elegimos el tipo de unión:

new_setosa = pd.concat([iris_medidas, iris_especies], axis=1, join='inner')
new_setosa
```

```
[84]:      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm      Species
0      1           5.1           3.5           1.4      Iris-setosa
1      2           4.9           3.0           1.4      Iris-setosa
2      3           4.7           3.2           1.3      Iris-setosa
3      4           4.6           3.1           1.5      Iris-setosa
4      5           5.0           3.6           1.4      Iris-setosa
..      ...           ...           ...           ...           ...
145  146           6.7           3.0           5.2    Iris-virginica
146  147           6.3           2.5           5.0    Iris-virginica
147  148           6.5           3.0           5.2    Iris-virginica
148  149           6.2           3.4           5.4    Iris-virginica
149  150           5.9           3.0           5.1    Iris-virginica
```

[150 rows x 5 columns]

```
[85]: display.Image('./images/merging_concat_mixed.png')
```

```
[85]:
```

df1					s1		Result					
	A	B	C	D		X		A	B	C	D	X
0	A0	B0	C0	D0	0	X0	0	A0	B0	C0	D0	X0
1	A1	B1	C1	D1	1	X1	1	A1	B1	C1	D1	X1
2	A2	B2	C2	D2	2	X2	2	A2	B2	C2	D2	X2
3	A3	B3	C3	D3	3	X3	3	A3	B3	C3	D3	X3

3.3 merge()

many-to-many: El método merge une dos dataframe por el Id de cada una de las filas

```
[86]: new_species = iris.loc[:, ['Id', 'Species']]
new_species
```

```
[86]:      Id      Species
0      1  Iris-setosa
1      2  Iris-setosa
2      3  Iris-setosa
3      4  Iris-setosa
4      5  Iris-setosa
..    ...      ...
145  146  Iris-virginica
146  147  Iris-virginica
147  148  Iris-virginica
148  149  Iris-virginica
149  150  Iris-virginica
```

[150 rows x 2 columns]

```
[87]: new_setosa = pd.merge(iris_medidas, new_species, on='Id')
new_setosa
```

```
[87]:      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm      Species
0      1           5.1           3.5           1.4  Iris-setosa
1      2           4.9           3.0           1.4  Iris-setosa
2      3           4.7           3.2           1.3  Iris-setosa
3      4           4.6           3.1           1.5  Iris-setosa
4      5           5.0           3.6           1.4  Iris-setosa
..    ...      ...      ...      ...      ...
145  146           6.7           3.0           5.2  Iris-virginica
146  147           6.3           2.5           5.0  Iris-virginica
147  148           6.5           3.0           5.2  Iris-virginica
148  149           6.2           3.4           5.4  Iris-virginica
```

149 150 5.9 3.0 5.1 Iris-virginica

[150 rows x 5 columns]

```
[88]: display.Image('./images/merging_merge_on_key.png')
```

[88]:

left				right				Result					
	key	A	B		key	C	D		key	A	B	C	D
0	K0	A0	B0	0	K0	C0	D0	0	K0	A0	B0	C0	D0
1	K1	A1	B1	1	K1	C1	D1	1	K1	A1	B1	C1	D1
2	K2	A2	B2	2	K2	C2	D2	2	K2	A2	B2	C2	D2
3	K3	A3	B3	3	K3	C3	D3	3	K3	A3	B3	C3	D3

Se puede añadir un parámetro que se llama **how**, donde se especifica el tipo de unión de los dataframes, para ello, nos basamos en la siguiente tabla para relacionarlos con los comandos SQL:

Merge method	SQL Join Name	Description
left	LEFT OUTER JOIN	Use keys from left frame only
right	RIGHT OUTER JOIN	Use keys from right frame only
outer	FULL OUTER JOIN	Use union of keys from both frames
inner	INNER JOIN	Use intersection of keys from both frames
cross	CROSS JOIN	Create the cartesian product of rows of both frames

```
[89]: new_setosa = pd.merge(iris_medidas, new_species, how='left', on='Id')
new_setosa
```

```
[89]:
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	Species
0	1	5.1	3.5	1.4	Iris-setosa
1	2	4.9	3.0	1.4	Iris-setosa
2	3	4.7	3.2	1.3	Iris-setosa
3	4	4.6	3.1	1.5	Iris-setosa
4	5	5.0	3.6	1.4	Iris-setosa
...
145	146	6.7	3.0	5.2	Iris-virginica
146	147	6.3	2.5	5.0	Iris-virginica
147	148	6.5	3.0	5.2	Iris-virginica
148	149	6.2	3.4	5.4	Iris-virginica
149	150	5.9	3.0	5.1	Iris-virginica

[150 rows x 5 columns]


```
[90]: new_setosa = pd.merge(iris_medidas, new_species, how='right', on='Id')
new_setosa
```

```
[90]:
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	Species
0	1	5.1	3.5	1.4	Iris-setosa
1	2	4.9	3.0	1.4	Iris-setosa
2	3	4.7	3.2	1.3	Iris-setosa
3	4	4.6	3.1	1.5	Iris-setosa
4	5	5.0	3.6	1.4	Iris-setosa
..
145	146	6.7	3.0	5.2	Iris-virginica
146	147	6.3	2.5	5.0	Iris-virginica
147	148	6.5	3.0	5.2	Iris-virginica
148	149	6.2	3.4	5.4	Iris-virginica
149	150	5.9	3.0	5.1	Iris-virginica

[150 rows x 5 columns]

```
[91]: new_setosa = pd.merge(iris_medidas, new_species, how='inner', on='Id')
new_setosa
```

```
[91]:
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	Species
0	1	5.1	3.5	1.4	Iris-setosa
1	2	4.9	3.0	1.4	Iris-setosa
2	3	4.7	3.2	1.3	Iris-setosa
3	4	4.6	3.1	1.5	Iris-setosa
4	5	5.0	3.6	1.4	Iris-setosa
..
145	146	6.7	3.0	5.2	Iris-virginica
146	147	6.3	2.5	5.0	Iris-virginica
147	148	6.5	3.0	5.2	Iris-virginica
148	149	6.2	3.4	5.4	Iris-virginica
149	150	5.9	3.0	5.1	Iris-virginica

[150 rows x 5 columns]

```
[92]: new_setosa = pd.merge(iris_medidas, new_species, how='outer', on='Id')
new_setosa
```

```
[92]:
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	Species
0	1	5.1	3.5	1.4	Iris-setosa
1	2	4.9	3.0	1.4	Iris-setosa
2	3	4.7	3.2	1.3	Iris-setosa
3	4	4.6	3.1	1.5	Iris-setosa
4	5	5.0	3.6	1.4	Iris-setosa
..
145	146	6.7	3.0	5.2	Iris-virginica

146	147	6.3	2.5	5.0	Iris-virginica
147	148	6.5	3.0	5.2	Iris-virginica
148	149	6.2	3.4	5.4	Iris-virginica
149	150	5.9	3.0	5.1	Iris-virginica

[150 rows x 5 columns]

```
[93]: new_setosa = pd.merge(iris_medidas, new_species, how='cross')
new_setosa
```

```
[93]:
```

	Id_x	SepalLengthCm	SepalWidthCm	PetalLengthCm	Id_y	Species
0	1	5.1	3.5	1.4	1	Iris-setosa
1	1	5.1	3.5	1.4	2	Iris-setosa
2	1	5.1	3.5	1.4	3	Iris-setosa
3	1	5.1	3.5	1.4	4	Iris-setosa
4	1	5.1	3.5	1.4	5	Iris-setosa
...
22495	150	5.9	3.0	5.1	146	Iris-virginica
22496	150	5.9	3.0	5.1	147	Iris-virginica
22497	150	5.9	3.0	5.1	148	Iris-virginica
22498	150	5.9	3.0	5.1	149	Iris-virginica
22499	150	5.9	3.0	5.1	150	Iris-virginica

[22500 rows x 6 columns]

```
[94]: # Si no existe la clave la duplica en el caso how=cross:
display.Image('./images/merging_merge_on_key_dup.png')
```

[94]:

left			right			Result			
							A_x	B	A_y
						0	1	2	4
						1	1	2	5
						2	1	2	6
						3	2	2	4
						4	2	2	5
						5	2	2	6

3.4 join()

```
[95]: iris_medidas
```

```
[95]:      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm
0      1             5.1             3.5             1.4
1      2             4.9             3.0             1.4
2      3             4.7             3.2             1.3
3      4             4.6             3.1             1.5
4      5             5.0             3.6             1.4
..    ...             ...             ...             ...
145   146             6.7             3.0             5.2
146   147             6.3             2.5             5.0
147   148             6.5             3.0             5.2
148   149             6.2             3.4             5.4
149   150             5.9             3.0             5.1
```

[150 rows x 4 columns]

```
[96]: iris_especies
```

```
[96]:      Species
0      Iris-setosa
1      Iris-setosa
2      Iris-setosa
3      Iris-setosa
4      Iris-setosa
..    ...
145   Iris-virginica
146   Iris-virginica
147   Iris-virginica
148   Iris-virginica
149   Iris-virginica
```

[150 rows x 1 columns]

```
[97]: iris_2 = iris_medidas.join(iris_especies)
iris_2
```

```
[97]:      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm      Species
0      1             5.1             3.5             1.4      Iris-setosa
1      2             4.9             3.0             1.4      Iris-setosa
2      3             4.7             3.2             1.3      Iris-setosa
3      4             4.6             3.1             1.5      Iris-setosa
4      5             5.0             3.6             1.4      Iris-setosa
..    ...             ...             ...             ...      ...
145   146             6.7             3.0             5.2   Iris-virginica
146   147             6.3             2.5             5.0   Iris-virginica
```

147	148	6.5	3.0	5.2	Iris-virginica
148	149	6.2	3.4	5.4	Iris-virginica
149	150	5.9	3.0	5.1	Iris-virginica

[150 rows x 5 columns]

También se le puede añadir los parámetros de `how` y `on`, igual que se hace con el método `merge()`

3.5 Grouping

Por “group by” nos referimos a un proceso que implica uno o más de los siguientes pasos:

- **Splitting** los datos en grupos según ciertos criterios
- **Applying** una función a cada grupo de forma independiente
- **Combining** los resultados en una estructura de datos

```
[98]: iris
```

```
[98]:
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	\
0	1	5.1	3.5	1.4	0.2	
1	2	4.9	3.0	1.4	0.2	
2	3	4.7	3.2	1.3	0.2	
3	4	4.6	3.1	1.5	0.2	
4	5	5.0	3.6	1.4	0.2	
..	
145	146	6.7	3.0	5.2	2.3	
146	147	6.3	2.5	5.0	1.9	
147	148	6.5	3.0	5.2	2.0	
148	149	6.2	3.4	5.4	2.3	
149	150	5.9	3.0	5.1	1.8	

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

[150 rows x 6 columns]

```
[99]: iris_sepal = iris.groupby('Species')[["SepalLengthCm", "SepalWidthCm"]].mean()
iris_sepal
```

```
[99]:
```

	SepalLengthCm	SepalWidthCm
Species		
Iris-setosa	5.006	3.418
Iris-versicolor	5.936	2.770
Iris-virginica	6.588	2.974

```
[100]: iris_petal = iris.groupby('Species')[["PetalLengthCm", "PetalWidthCm"]].mean()
iris_petal
```

```
[100]:
```

	PetalLengthCm	PetalWidthCm
Species		
Iris-setosa	1.464	0.244
Iris-versicolor	4.260	1.326
Iris-virginica	5.552	2.026

3.6 Reshaping

3.7 stack()

```
[101]: # Ponemos como columna de index la de especies, así aplicaremos los datos segun
↳ de que
# especie sean:

reiris = iris.set_index('Species', append=True)
reiris
```

```
[101]:
```

		Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	\
	Species					
0	Iris-setosa	1	5.1	3.5	1.4	
1	Iris-setosa	2	4.9	3.0	1.4	
2	Iris-setosa	3	4.7	3.2	1.3	
3	Iris-setosa	4	4.6	3.1	1.5	
4	Iris-setosa	5	5.0	3.6	1.4	
...	
145	Iris-virginica	146	6.7	3.0	5.2	
146	Iris-virginica	147	6.3	2.5	5.0	
147	Iris-virginica	148	6.5	3.0	5.2	
148	Iris-virginica	149	6.2	3.4	5.4	
149	Iris-virginica	150	5.9	3.0	5.1	

		PetalWidthCm
	Species	
0	Iris-setosa	0.2
1	Iris-setosa	0.2
2	Iris-setosa	0.2
3	Iris-setosa	0.2
4	Iris-setosa	0.2
...

```

145 Iris-virginica      2.3
146 Iris-virginica      1.9
147 Iris-virginica      2.0
148 Iris-virginica      2.3
149 Iris-virginica      1.8

```

[150 rows x 5 columns]

```
[102]: stack_iris = reiris.stack(future_stack=True)
stack_iris
```

```
[102]:   Species
0   Iris-setosa   Id      1.0
      SepalLengthCm  5.1
      SepalWidthCm   3.5
      PetalLengthCm  1.4
      PetalWidthCm   0.2
      ...
149 Iris-virginica Id     150.0
      SepalLengthCm  5.9
      SepalWidthCm   3.0
      PetalLengthCm  5.1
      PetalWidthCm   1.8

```

Length: 750, dtype: float64

Nos muestra los datos apilados según la especie y las longitudes de los pétalos y sépalos.

Para desapilar usaremos el método `unstack`.

```
[103]: unstack_iris = reiris.unstack()
unstack_iris
```

```
[103]:   Id      SepalLengthCm \
Species Iris-setosa Iris-versicolor Iris-virginica  Iris-setosa
0          1.0          NaN          NaN          5.1
1          2.0          NaN          NaN          4.9
2          3.0          NaN          NaN          4.7
3          4.0          NaN          NaN          4.6
4          5.0          NaN          NaN          5.0
..          ...          ...          ...          ...
145         NaN          NaN          146.0         NaN
146         NaN          NaN          147.0         NaN
147         NaN          NaN          148.0         NaN
148         NaN          NaN          149.0         NaN
149         NaN          NaN          150.0         NaN

      SepalWidthCm \
Species Iris-versicolor Iris-virginica  Iris-setosa Iris-versicolor
```

0	NaN	NaN	3.5	NaN
1	NaN	NaN	3.0	NaN
2	NaN	NaN	3.2	NaN
3	NaN	NaN	3.1	NaN
4	NaN	NaN	3.6	NaN
..
145	NaN	6.7	NaN	NaN
146	NaN	6.3	NaN	NaN
147	NaN	6.5	NaN	NaN
148	NaN	6.2	NaN	NaN
149	NaN	5.9	NaN	NaN

	PetalLengthCm				
Species	Iris-virginica	Iris-setosa	Iris-versicolor	Iris-virginica	\
0	NaN	1.4	NaN	NaN	
1	NaN	1.4	NaN	NaN	
2	NaN	1.3	NaN	NaN	
3	NaN	1.5	NaN	NaN	
4	NaN	1.4	NaN	NaN	
..	
145	3.0	NaN	NaN	5.2	
146	2.5	NaN	NaN	5.0	
147	3.0	NaN	NaN	5.2	
148	3.4	NaN	NaN	5.4	
149	3.0	NaN	NaN	5.1	

	PetalWidthCm		
Species	Iris-setosa	Iris-versicolor	Iris-virginica
0	0.2	NaN	NaN
1	0.2	NaN	NaN
2	0.2	NaN	NaN
3	0.2	NaN	NaN
4	0.2	NaN	NaN
..
145	NaN	NaN	2.3
146	NaN	NaN	1.9
147	NaN	NaN	2.0
148	NaN	NaN	2.3
149	NaN	NaN	1.8

[150 rows x 15 columns]

3.8 pivot_table()

```
[104]: # Agrupación de datos de especie por media:  
# Podemos añadir: df, values="D", index=["A", "B"], columns=["C"]
```

```
iris_pivot = pd.pivot_table(iris, index='Species')  
iris_pivot
```

```
[104]:
```

	Id	PetalLengthCm	PetalWidthCm	SepalLengthCm	\
Species					
Iris-setosa	25.5	1.464	0.244	5.006	
Iris-versicolor	75.5	4.260	1.326	5.936	
Iris-virginica	125.5	5.552	2.026	6.588	

	SepalWidthCm
Species	
Iris-setosa	3.418
Iris-versicolor	2.770
Iris-virginica	2.974

```
[105]: # Agrupación de datos de especie por media:  
  
iris_pivot2 = pd.pivot_table(iris, index='Species', aggfunc="sum")  
iris_pivot2
```

```
[105]:
```

	Id	PetalLengthCm	PetalWidthCm	SepalLengthCm	\
Species					
Iris-setosa	1275	73.2	12.2	250.3	
Iris-versicolor	3775	213.0	66.3	296.8	
Iris-virginica	6275	277.6	101.3	329.4	

	SepalWidthCm
Species	
Iris-setosa	170.9
Iris-versicolor	138.5
Iris-virginica	148.7

```
[106]: # el parametro values nos ayuda a seleccionar las columnas concretas:  
  
iris_pivot = pd.pivot_table(iris, values="PetalLengthCm", index='Species')  
iris_pivot
```

```
[106]:
```

	PetalLengthCm
Species	
Iris-setosa	1.464
Iris-versicolor	4.260
Iris-virginica	5.552

3.9 Time Series

```
[107]: # Generamos una serie temporal primero generamos los valores de la fecha de la
      ↪ que quieres partir, creando 15 días consecutivos:
      # Una vez creados ponemos valores aleatorios a esas fechas:

      rng = pd.date_range("6/1/2024 00:00", periods=15, freq="D")
      ts = pd.Series(np.random.randn(len(rng)), rng)
      ts
```

```
[107]: 2024-06-01    2.568833
      2024-06-02    0.302244
      2024-06-03   -0.502816
      2024-06-04   -0.626335
      2024-06-05   -1.537261
      2024-06-06    0.473356
      2024-06-07   -1.424944
      2024-06-08    0.845725
      2024-06-09    1.018898
      2024-06-10   -0.368384
      2024-06-11    0.606400
      2024-06-12    1.209049
      2024-06-13   -0.398228
      2024-06-14    0.516435
      2024-06-15    0.156343
      Freq: D, dtype: float64
```

3.10 tz_localize()

```
[108]: # añadimos la hora al dataframe creado:

      ts_utc = ts.tz_localize("UTC")
      ts_utc
```

```
[108]: 2024-06-01 00:00:00+00:00    2.568833
      2024-06-02 00:00:00+00:00    0.302244
      2024-06-03 00:00:00+00:00   -0.502816
      2024-06-04 00:00:00+00:00   -0.626335
      2024-06-05 00:00:00+00:00   -1.537261
      2024-06-06 00:00:00+00:00    0.473356
      2024-06-07 00:00:00+00:00   -1.424944
      2024-06-08 00:00:00+00:00    0.845725
      2024-06-09 00:00:00+00:00    1.018898
      2024-06-10 00:00:00+00:00   -0.368384
      2024-06-11 00:00:00+00:00    0.606400
      2024-06-12 00:00:00+00:00    1.209049
      2024-06-13 00:00:00+00:00   -0.398228
      2024-06-14 00:00:00+00:00    0.516435
```

```
2024-06-15 00:00:00+00:00    0.156343
Freq: D, dtype: float64
```

3.11 tz_convert()

```
[109]: # Ponemos la franja horaria a la cual nos encontramos:

ts_utc.tz_convert("Europe/Madrid")
```

```
[109]: 2024-06-01 02:00:00+02:00    2.568833
2024-06-02 02:00:00+02:00    0.302244
2024-06-03 02:00:00+02:00   -0.502816
2024-06-04 02:00:00+02:00   -0.626335
2024-06-05 02:00:00+02:00   -1.537261
2024-06-06 02:00:00+02:00    0.473356
2024-06-07 02:00:00+02:00   -1.424944
2024-06-08 02:00:00+02:00    0.845725
2024-06-09 02:00:00+02:00    1.018898
2024-06-10 02:00:00+02:00   -0.368384
2024-06-11 02:00:00+02:00    0.606400
2024-06-12 02:00:00+02:00    1.209049
2024-06-13 02:00:00+02:00   -0.398228
2024-06-14 02:00:00+02:00    0.516435
2024-06-15 02:00:00+02:00    0.156343
Freq: D, dtype: float64
```

3.12 offsets.BusinessDay()

Escogemos de ese periodo de tiempo los que sean laborables, ayuda de offset.BusinnesDay():

```
[110]: rng
```

```
[110]: DatetimeIndex(['2024-06-01', '2024-06-02', '2024-06-03', '2024-06-04',
                    '2024-06-05', '2024-06-06', '2024-06-07', '2024-06-08',
                    '2024-06-09', '2024-06-10', '2024-06-11', '2024-06-12',
                    '2024-06-13', '2024-06-14', '2024-06-15'],
                    dtype='datetime64[ns]', freq='D')
```

```
[111]: # se añade 5 como número de días a representar:
rng = rng + pd.offsets.BusinessDay(5)
```

```
[112]: ts = pd.Series(np.random.randn(len(rng)), rng).tz_localize("UTC")
ts
```

```
[112]: 2024-06-07 00:00:00+00:00    0.062354
2024-06-07 00:00:00+00:00    0.033048
2024-06-10 00:00:00+00:00   -0.227042
2024-06-11 00:00:00+00:00    1.189045
```

```

2024-06-12 00:00:00+00:00    -0.424436
2024-06-13 00:00:00+00:00    -1.268878
2024-06-14 00:00:00+00:00    -0.599337
2024-06-14 00:00:00+00:00    -0.876437
2024-06-14 00:00:00+00:00     0.885342
2024-06-17 00:00:00+00:00     0.131211
2024-06-18 00:00:00+00:00    -1.985825
2024-06-19 00:00:00+00:00    -0.689601
2024-06-20 00:00:00+00:00     0.591801
2024-06-21 00:00:00+00:00     1.163983
2024-06-21 00:00:00+00:00    -1.356650
dtype: float64

```

```
[113]: ts.tz_convert("Europe/Madrid")
```

```

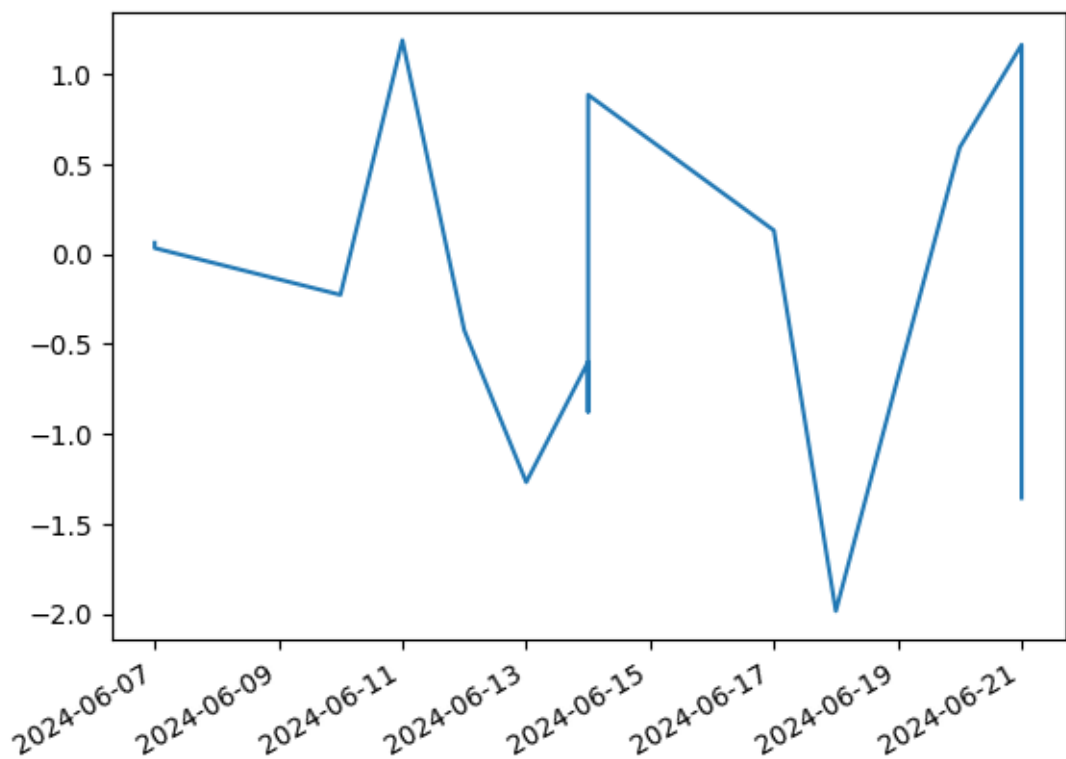
[113]: 2024-06-07 02:00:00+02:00     0.062354
2024-06-07 02:00:00+02:00     0.033048
2024-06-10 02:00:00+02:00    -0.227042
2024-06-11 02:00:00+02:00     1.189045
2024-06-12 02:00:00+02:00    -0.424436
2024-06-13 02:00:00+02:00    -1.268878
2024-06-14 02:00:00+02:00    -0.599337
2024-06-14 02:00:00+02:00    -0.876437
2024-06-14 02:00:00+02:00     0.885342
2024-06-17 02:00:00+02:00     0.131211
2024-06-18 02:00:00+02:00    -1.985825
2024-06-19 02:00:00+02:00    -0.689601
2024-06-20 02:00:00+02:00     0.591801
2024-06-21 02:00:00+02:00     1.163983
2024-06-21 02:00:00+02:00    -1.356650
dtype: float64

```

```
[114]: import matplotlib.pyplot as plt
```

```
[115]: ts.plot()
```

```
[115]: <Axes: >
```



3.13 Categoricals

```
[116]: iris
```

```
[116]:
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	\
0	1	5.1	3.5	1.4	0.2	
1	2	4.9	3.0	1.4	0.2	
2	3	4.7	3.2	1.3	0.2	
3	4	4.6	3.1	1.5	0.2	
4	5	5.0	3.6	1.4	0.2	
..	
145	146	6.7	3.0	5.2	2.3	
146	147	6.3	2.5	5.0	1.9	
147	148	6.5	3.0	5.2	2.0	
148	149	6.2	3.4	5.4	2.3	
149	150	5.9	3.0	5.1	1.8	

	Species
0	Iris-setosa
1	Iris-setosa
2	Iris-setosa
3	Iris-setosa

```

4      Iris-setosa
..      ""
145    Iris-virginica
146    Iris-virginica
147    Iris-virginica
148    Iris-virginica
149    Iris-virginica

[150 rows x 6 columns]

```

```
[117]: iris.dtypes
```

```

[117]: Id          int64
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species          object
dtype: object

```

```

[118]: # Convertimos la columna Species en categoricas:
iris["Species"] = iris["Species"].astype("category")
iris.dtypes

```

```

[118]: Id          int64
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species          category
dtype: object

```

3.14 rename_categories()

```

[119]: # Renombrar la columna especie con solo la especie que es:

new_categories = ["setosa", "versicolor", "virginica"]

iris["Species"] = iris["Species"].cat.rename_categories(new_categories)
iris

```

```

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

```

```

.. ...
145 146      6.7      3.0      5.2      2.3 virginica
146 147      6.3      2.5      5.0      1.9 virginica
147 148      6.5      3.0      5.2      2.0 virginica
148 149      6.2      3.4      5.4      2.3 virginica
149 150      5.9      3.0      5.1      1.8 virginica

```

[150 rows x 6 columns]

3.15 set_categories()

```

[120]: # Renombrar la columna sustituyendo por los valores por ejemplo,
# renombrar las viejas categorias ponemos rename=True:

new_categories = [0, 1, 2]

iris["spc"] = iris["Species"].cat.set_categories(new_categories, rename=True)
iris

```

```

[120]:      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Species \
0      1          5.1          3.5          1.4          0.2    setosa
1      2          4.9          3.0          1.4          0.2    setosa
2      3          4.7          3.2          1.3          0.2    setosa
3      4          4.6          3.1          1.5          0.2    setosa
4      5          5.0          3.6          1.4          0.2    setosa
.. ...
145 146      6.7      3.0      5.2      2.3 virginica
146 147      6.3      2.5      5.0      1.9 virginica
147 148      6.5      3.0      5.2      2.0 virginica
148 149      6.2      3.4      5.4      2.3 virginica
149 150      5.9      3.0      5.1      1.8 virginica

```

```

      spc
0      0
1      0
2      0
3      0
4      0
.. ..
145  2
146  2
147  2
148  2
149  2

```

[150 rows x 7 columns]

3.16 sort_values()

```
[121]: # Colocar las filas según los valores de una columna, en este caso ordenamos
        ↪ por la especie (spc):
```

```
iris.sort_values(by="spc", ascending=False)
```

```
[121]:
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species	\
149	150	5.9	3.0	5.1	1.8	virginica	
111	112	6.4	2.7	5.3	1.9	virginica	
122	123	7.7	2.8	6.7	2.0	virginica	
121	122	5.6	2.8	4.9	2.0	virginica	
120	121	6.9	3.2	5.7	2.3	virginica	
..	
31	32	5.4	3.4	1.5	0.4	setosa	
30	31	4.8	3.1	1.6	0.2	setosa	
29	30	4.7	3.2	1.6	0.2	setosa	
28	29	5.2	3.4	1.4	0.2	setosa	
0	1	5.1	3.5	1.4	0.2	setosa	

```
      spc
149    2
111    2
122    2
121    2
120    2
..    ..
31     0
30     0
29     0
28     0
0      0
```

```
[150 rows x 7 columns]
```

```
[122]: # Agrupamos para que nos muestre cuantos valores tenemos de cada uno, para ello
        ↪ usamos observed=False en groupby,
        # Incluyen categorías vacías si las hubiera:
```

```
iris.groupby("spc", observed=False).size()
```

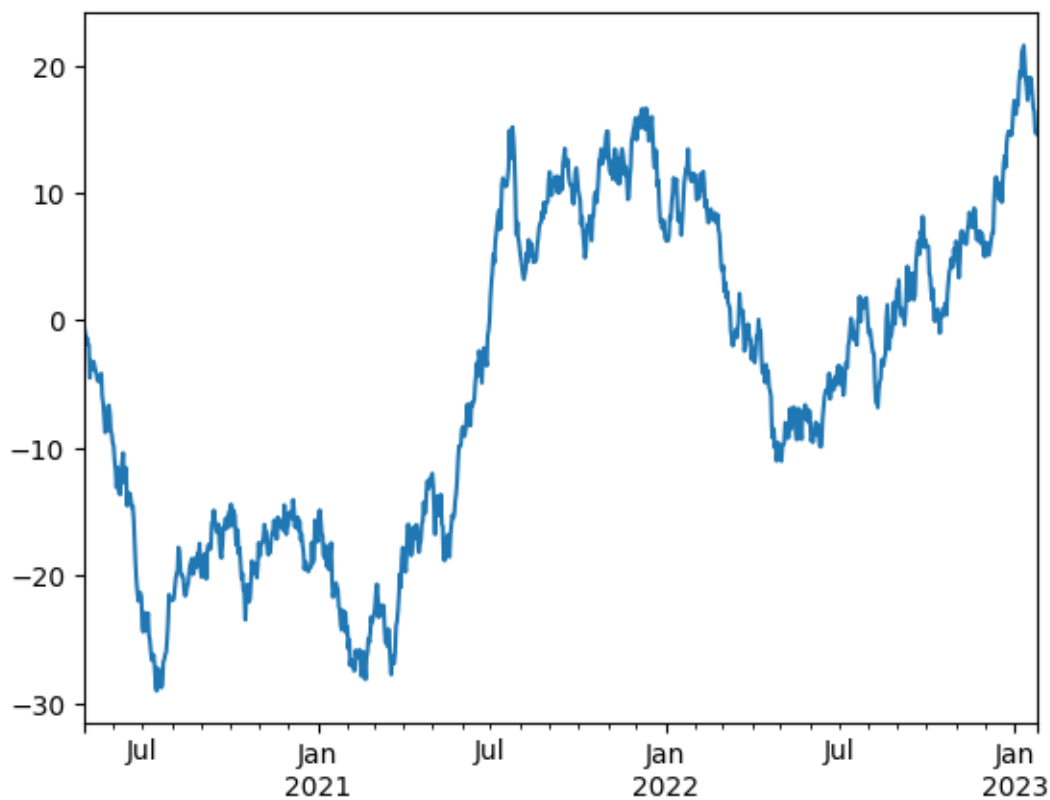
```
[122]: spc
0      50
1      50
2      50
dtype: int64
```

3.17 Plotting

Pandas usa de manera interna matplotlib, simplemente importando la librería y pasando el dataframe a `.plot()` te genera el gráfico:

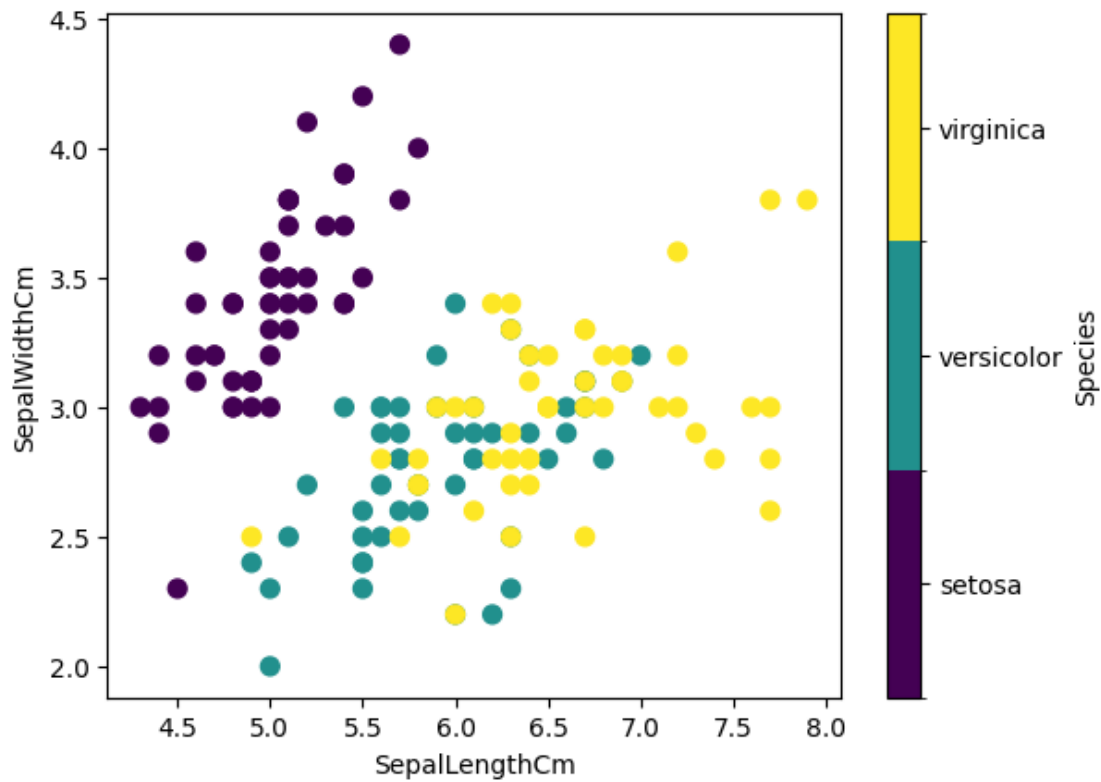
```
[123]: ts = pd.Series(np.random.randn(1000), index=pd.date_range("5/1/2020",  
    ↪ periods=1000))  
  
ts = ts.cumsum()  
  
ts.plot()
```

[123]: <Axes: >



```
[124]: # c: variable categorica  
# cmap: escala de color  
# s: tamaño de los puntos  
  
iris.plot.scatter(x='SepalLengthCm', y='SepalWidthCm', c='Species',  
    ↪ cmap="viridis", s=50)
```

[124]: <Axes: xlabel='SepalLengthCm', ylabel='SepalWidthCm'>



[125]: *# Usando matplotlib ampliando pandas:*

```
df = pd.DataFrame(
    np.random.randn(1000, 4),
    index=ts.index,
    columns=["A", "B", "C", "D"]
)

df = df.cumsum()

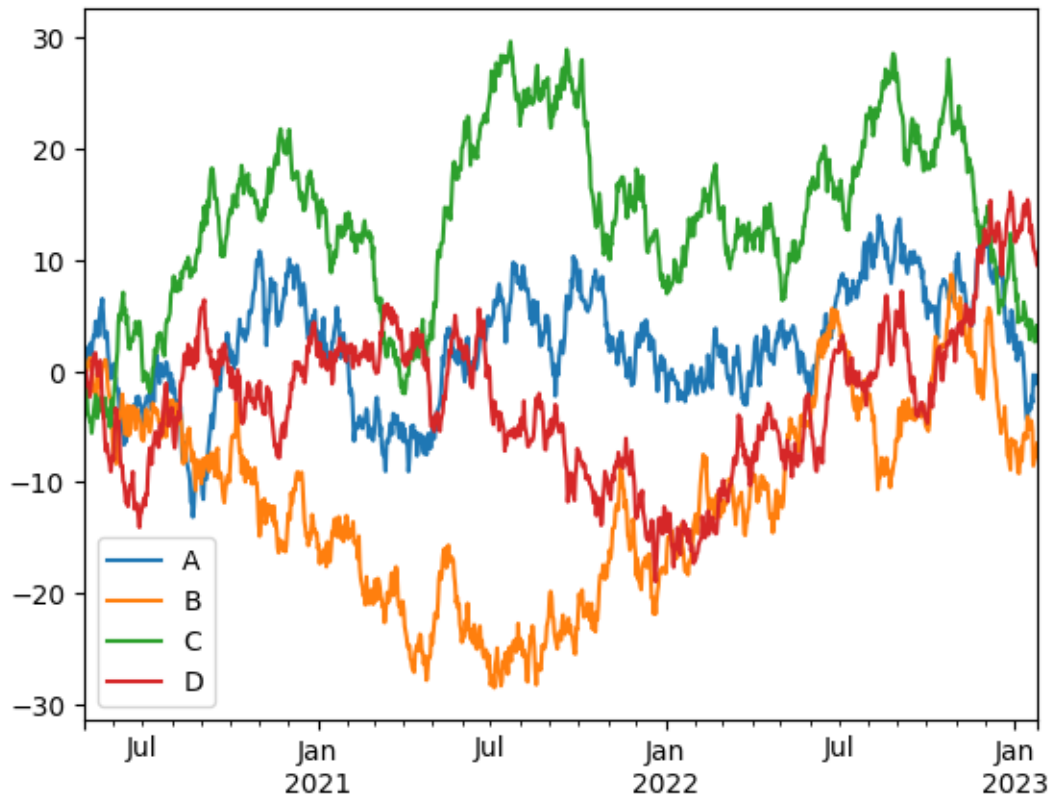
plt.figure()

df.plot()

plt.legend(loc='best')
```

[125]: <matplotlib.legend.Legend at 0x76cc3359ac60>

<Figure size 640x480 with 0 Axes>



4 Numpy

1. Método `array()`
2. Método `arange()`
3. [Matrices básicas en numpy](#)
4. Métodos `random()` / `indices()`
5. [Réplicas o copias con numpy](#)
6. [Leer un archivo csv con el método `loadtxt\(\)`](#)
7. [Modificación de matrices](#)
8. [Slicing](#)
9. [Comparacion entre Arrays](#)
10. [Operaciones](#)
11. [Matematical functions](#)

5 Numpy

```
[126]: # pip install numpy
```

```
[127]: import numpy as np
```

5.1 Método array()

Un array puede formarse apartir de otras estructuras de Python como son listas o tuplas:

```
[128]: e = np.array([
        [1, 2],
        [3, 4],
        [5, 6]
    ])
    e
```

```
[128]: array([[1, 2],
             [3, 4],
             [5, 6]])
```

```
[129]: print(e)
```

```
[[1 2]
 [3 4]
 [5 6]]
```

```
[130]: # Cuidado No confundir con las listas, en las matrices faltan las comas!!!
    listado = [[1, 3], [2, 6]]
    print(listado)
    print(type(e))
    print(type(listado))
```

```
[[1, 3], [2, 6]]
<class 'numpy.ndarray'>
<class 'list'>
```

```
[131]: len(e)
```

```
[131]: 3
```

```
[132]: e.shape
```

```
[132]: (3, 2)
```

```
[133]: e.size
```

```
[133]: 6
```

```
[134]: e[0]
```

```
[134]: array([1, 2])
```

```
[135]: for i in range(len(e)):
        # print(e[i])
        for j in range(len(e[i])):
```

```
print(e[i][j])
```

```
1  
2  
3  
4  
5  
6
```

```
[136]: a1d = np.array((1, 5, 6))  
a1d
```

```
[136]: array([1, 5, 6])
```

Se puede añadir otro atributo que es `dtype` indicando de cuantos bytes consta el array:

```
[137]: np.array([127, 128, 129], dtype=np.int8)
```

```
-----  
OverflowError                                Traceback (most recent call last)  
Cell In[137], line 1  
----> 1 np.array([127, 128, 129], dtype=np.int8)  
  
OverflowError: Python integer 128 out of bounds for int8
```

Representa enteros desde -128 a 127, arroja un error de fuera de rango.

Lo normal es que se formen arrays entre 32 o 64-bit de valores enteros o decimales:

```
[138]: a = np.array([2, 3, 4], dtype=np.uint32)  
b = np.array([5, 6, 7], dtype=np.uint32)  
c = a - b  
c
```

```
[138]: array([4294967293, 4294967293, 4294967293], dtype=uint32)
```

```
[139]: c_32 = a - b.astype(np.int32)  
c_32
```

```
[139]: array([-3, -3, -3])
```

El método `.astype()` convierte el array `b` en `int32`, en vez en `uint32`.

Podemos saber de que tipo de datos son mediante la función `issubdtype()`:

```
[140]: d = np.dtype(np.int64)  
  
# 1º Atributo es el array a testear y 2º Atributo el tipo que queremos comparar  
↪ (entero, decimal, etc):
```

```
print(np.issubdtype(d, np.integer))
print(np.issubdtype(d, np.floating))
```

True

False

Los tipos de datos pueden ser: booleanos (bool), enteros (int), enteros sin signo (uint), decimales (float) y complejos (complex).

También pueden ser: string numpy.str_ dtype (U character code), secuencia de bytes numpy.bytes_ (S character code), and arbitrary byte sequences, via numpy.void (V character code).

```
[141]: np.array(["hello", "world"], dtype="S7").tobytes()
```

```
[141]: b'hello\x00\x00world\x00\x00'
```

5.2 Método arange().

5.3 Numeros dentro de un rango:

Generación de números con numpy en un rango

```
[142]: a = np.arange(6)
a
```

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

```
[143]: type(a)
```

```
[143]: numpy.ndarray
```

Formas de imprimir la información

```
[144]: a
```

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

```
[145]: for i in a:
        print(i)
```

0

1

2

3

4

5

Longitud, forma, tamaño

```
[146]: a
```

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

```
[147]: len(a)
```

```
[147]: 6
```

```
[148]: a.shape
```

```
[148]: (6,)
```

```
[149]: a.size
```

```
[149]: 6
```

Media, mediana, desviación típica, máximos y mínimos

```
[150]: a
```

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

```
[151]: np.mean(a)
```

```
[151]: np.float64(2.5)
```

```
[152]: np.median(a)
```

```
[152]: np.float64(2.5)
```

```
[153]: np.std(a)
```

```
[153]: np.float64(1.707825127659933)
```

```
[154]: max(a)
```

```
[154]: np.int64(5)
```

```
[155]: min(a)
```

```
[155]: np.int64(0)
```

```
[156]: np.percentile(a, [25, 50, 75])
```

```
[156]: array([1.25, 2.5 , 3.75])
```

Comprobación de elementos en el array

```
[157]: a
```

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

```
[158]: 25 in a
```

```
[158]: False
```

```
[159]: 0 in a
```

```
[159]: True
```

```
[160]: 25 not in a
```

```
[160]: True
```

```
[161]: 0 not in a
```

```
[161]: False
```

Redefinir el tamaño

```
[162]: a
```

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

```
[163]: a1 = a.reshape(2, 3)
a1
```

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

Generar números en un intervalo

```
[164]: # sin especificar va de 1 en 1
b = np.arange(2,7) # 2, 3, 4, 5, 6
b
```

```
[164]: array([2, 3, 4, 5, 6])
```

Generar números en un intervalo con salto

```
[165]: c = np.arange(10, 40, 5)
c
```

```
[165]: array([10, 15, 20, 25, 30, 35])
```

```
[166]: d = np.arange(10, 41, 5)
d
```

```
[166]: array([10, 15, 20, 25, 30, 35, 40])
```

También tenemos el atributo `dtype` para definir de que tipo son los valores que forman el array:

```
[167]: # Definimos un array que empiece en 2 y acabe en 9 y sean decimales:
np.arange(2, 10, dtype=float)
```

```
[167]: array([2., 3., 4., 5., 6., 7., 8., 9.])
```

5.4 linspace()

```
[168]: # Recogemos una muestra de los datos, especificamos: min, max, y cada tantos
      ↪ recoja un valor
```

```
[169]: f = np.linspace(10, 20, 2) # de 10 a 20 con 2 elementos
f
```

```
[169]: array([10., 20.])
```

```
[170]: g = np.linspace(10, 20, 5) # de 10 a 20 muestra 5
g
```

```
[170]: array([10. , 12.5, 15. , 17.5, 20. ])
```

```
[171]: g1 = np.linspace(10, 20, 3) # de 10 a 20 muestra 3
g1
```

```
[171]: array([10., 15., 20.])
```

5.5 Matrices basicas en numpy

5.6 2D: Método eye(), diag() / vander()

5.6.1 Matriz Identidad: Diagonal principal llena de 1, resto 0

eye(n, m)

```
[172]: h = np.eye(3) # de 3 filas y 3 columnas --> matriz identidad
h
```

```
[172]: array([[1., 0., 0.],
          [0., 1., 0.],
          [0., 0., 1.]])
```

```
[173]: i = np.eye(5) # Matriz de 5 filas y 5 columnas
i
```

```
[173]: array([[1., 0., 0., 0., 0.],
          [0., 1., 0., 0., 0.],
          [0., 0., 1., 0., 0.],
          [0., 0., 0., 1., 0.],
          [0., 0., 0., 0., 1.]])
```



```
[174]: # n = filas, m = columnas, el resto que no son de la diagonal las rellena con 0:
np.eye(3, 5)
```

```
[174]: array([[1., 0., 0., 0., 0.],
           [0., 1., 0., 0., 0.],
           [0., 0., 1., 0., 0.]])

diag()
```

```
[175]: # Los elementos estan en la diagonal principal:

a2D = np.diag([1, 2, 3])
a2D
```

```
[175]: array([[1, 0, 0],
           [0, 2, 0],
           [0, 0, 3]])
```

```
[176]: # El segundo parámetro es agregar un fila y columna de 0:

np.diag([1, 2, 3], 1)
```

```
[176]: array([[0, 1, 0, 0],
           [0, 0, 2, 0],
           [0, 0, 0, 3],
           [0, 0, 0, 0]])

vander(x, n)
```

```
[177]: # x = array 1d, la lista o tupla de valores, n = al número de columnas:
np.vander([1, 2, 3, 4], 2)
```

```
[177]: array([[1, 1],
           [2, 1],
           [3, 1],
           [4, 1]])
```

```
[178]: # Se crea una matriz decreciente de los valores 1, 2, 3, 4, que contiene 4
      ↪ columnas:
# así, la primera columna decrece 64, 27, 8, 1
# segunda columna: 16, 9, 4, 1.

np.vander((1, 2, 3, 4), 4)
```

```
[178]: array([[ 1,  1,  1,  1],
           [ 8,  4,  2,  1],
           [27,  9,  3,  1],
           [64, 16,  4,  1]])
```

5.6.2 Matriz identidad multiplicada por un valor

```
[179]: j = 5 * i  
j
```

```
[179]: array([[5., 0., 0., 0., 0.],  
            [0., 5., 0., 0., 0.],  
            [0., 0., 5., 0., 0.],  
            [0., 0., 0., 5., 0.],  
            [0., 0., 0., 0., 5.]])
```

5.7 Métodos zeros() / ones()

5.7.1 Matriz de todo 1

```
[180]: k = np.ones((3, 4)) # Matriz de 3 filas por 4 columnas --> valores 1  
k
```

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

```
[181]: # se puede añadir un tercer parámetro que es el numero de arrays:  
  
np.ones((2, 3, 2))
```

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

5.7.2 Matriz de todo 0

```
[182]: l = np.zeros((3, 4)) # Matriz de 0s --> 3 filas por 4 columnas  
l
```

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

```
[183]: l2 = np.zeros((6, 2))  
l2
```

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

```

[0., 0.],
[0., 0.],
[0., 0.]]

```

```
[184]: np.zeros((2, 3, 2)) # Idem: a ones()
```

```
[184]: array([[[0., 0.],
               [0., 0.],
               [0., 0.]],

              [[0., 0.],
               [0., 0.],
               [0., 0.]])
```

5.8 Metodos random() / indices()

random() genera valores pseudoaleatorios entre 0 y 1:

```
[185]: from numpy.random import default_rng

# 42: corresponde a seed
# array de 2 filas x 3 columnas

default_rng(42).random((2,3))
```

```
[185]: array([[0.77395605, 0.43887844, 0.85859792],
              [0.69736803, 0.09417735, 0.97562235]])
```

```
[186]: default_rng(42).random((2,3,2)) # idem a ones()
```

```
[186]: array([[[0.77395605, 0.43887844],
               [0.85859792, 0.69736803],
               [0.09417735, 0.97562235]],

              [[0.7611397 , 0.78606431],
               [0.12811363, 0.45038594],
               [0.37079802, 0.92676499]])
```

indices(): genera una matriz de un conjunto de matrices:

```
[187]: # Matriz de 3 filas por 3 columnas:

np.indices((3,3))
```

```
[187]: array([[[0, 0, 0],
               [1, 1, 1],
               [2, 2, 2]],

              [[0, 1, 2],
```

```
[0, 1, 2],  
[0, 1, 2]])
```

6 Replicas o copias con numpy

```
[188]: a = np.array([1, 2, 3, 4, 5, 6])  
b = a[:2]  
b += 1  
print('a =', a, '; b =', b)
```

```
a = [2 3 3 4 5 6] ; b = [2 3]
```

El cambio realizado a b afecta en a en este caso es una réplica de a.

Ahora veamos que ocurre si usamos `numpy.copy()`

```
[189]: a = np.array([1, 2, 3, 4])  
b = a[:2].copy()  
b += 1  
print('a = ', a, 'b = ', b)
```

```
a = [1 2 3 4] b = [2 3]
```

En este caso, a no se ve afectado por los cambios de b, ya que b es una copia de a.

```
[190]: A = np.ones((2, 2))  
print('A: \n', A)  
B = np.eye(2, 2)  
print('B: \n', B)  
C = np.zeros((2, 2))  
print('C: \n', C)  
D = np.diag((-3, -4))  
print('D: \n', D)  
a4d = np.block([A, B], [C, D])  
print('4D: \n', a4d)
```

```
A:  
[[1. 1.]  
 [1. 1.]]  
B:  
[[1. 0.]  
 [0. 1.]]  
C:  
[[0. 0.]  
 [0. 0.]]  
D:  
[[-3  0]  
 [ 0 -4]]  
4D:
```

```
[[ 1.  1.  1.  0.]
 [ 1.  1.  0.  1.]
 [ 0.  0. -3.  0.]
 [ 0.  0.  0. -4.]]
```

`np.block`: crea la matriz resultante de: `[[A, B], [C, D]]`

6.1 Leer un archivo csv con el metodo `loadtxt()`

[191]: *# Poner nombre del archivo, seleccionar el delimitador que en este ejemplo es ↵
↵(,) y escapar la cabecera del documento (skiprows):*

```
np.loadtxt('./files/simple.csv', delimiter = ',', skiprows = 1)
```

```
[191]: array([[0., 0.],
             [1., 1.],
             [2., 4.],
             [3., 9.]])
```

6.2 Modificación de matrices

6.2.1 Transpuesta de una matriz: `transpose()` & `.T`

Intercambio de filas por columnas

```
[192]: m = np.array([[1, 2, 3],
                    [4, 5, 6]])
m
```

```
[192]: array([[1, 2, 3],
             [4, 5, 6]])
```

```
[193]: # Opción 1
m.transpose()
```

```
[193]: array([[1, 4],
             [2, 5],
             [3, 6]])
```

```
[194]: # Opción 2
m.T
```

```
[194]: array([[1, 4],
             [2, 5],
             [3, 6]])
```

6.2.2 Logic functions: Metodos all() & any()

```
[195]: n = np.array([[1, 2, 3],  
                    [4, 5, 6]])  
n
```

```
[195]: array([[1, 2, 3],  
            [4, 5, 6]])
```

```
[196]: # ALL --> ¿Todos los elementos son mayores de 0? --> True/False  
np.all(n>0)
```

```
[196]: np.True_
```

```
[197]: np.all(n>2)
```

```
[197]: np.False_
```

```
[198]: # ANY --> ¿Algún elemento son mayores de 2?  
np.any(n>2)
```

```
[198]: np.True_
```

Si queremos declarar un array con valores nulos usaremos: `np.nan` y lo comprobaremos mediante la función `np.isnan()`

```
[199]: x = np.array([[1., 2.], [np.nan, 3.], [np.nan, np.nan]])  
x
```

```
[199]: array([[ 1.,  2.],  
            [nan,  3.],  
            [nan, nan]])
```

```
[200]: # isnan nos muestra el array resultante con salida de True si es un valor nulo  
      ↪ False si no es un valor nulo:  
  
np.isnan(x)
```

```
[200]: array([[False, False],  
            [ True, False],  
            [ True,  True]])
```

6.2.3 Función ravel()

```
[201]: # Pone en una sola dimensión una matriz
```

```
[202]: p = np.array([[1, 2, 3],  
                    [4, 5, 6]])  
p
```

```
[202]: array([[1, 2, 3],  
            [4, 5, 6]])
```

```
[203]: # np.ravel(matriz a modificar)  
np.ravel(p)
```

```
[203]: array([1, 2, 3, 4, 5, 6])
```

```
[204]: p1 = np.array([[1, 2, 3],  
                    [4, 5, 6],  
                    [7, 8, 9]])  
  
p1
```

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

```
[205]: np.ravel(p1)
```

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

6.2.4 flatten()

```
[206]: # Es una copia del array pero en 1 sola dimensión
```

```
[207]: matriz = np.array([[1, 2, 3],  
                        [4, 5, 6],  
                        [7, 8, 9]])  
  
matriz
```

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

```
[208]: # nombre matriz + flatten()  
matriz.flatten()
```

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

```
[209]: m = matriz.flatten()
```

```
[210]: m.shape
```

```
[210]: (9,)
```

6.2.5 roll()

```
[211]: # np.roll(array, desplazamiento, eje)
# Desplaza los elementos de manera circular a través de una dimensión
```

```
[212]: b = np.array([[1, 2, 3, 4],
                    [5, 6, 7, 8],
                    [9, 10, 11, 12]])
b
```

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

```
[213]: # Desplazamiento= 1 y eje horizontal
np.roll(b, 1, axis=0)
```

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

```
[214]: # Desplazamiento = 1 y eje vertical
np.roll(b, 1, axis=1)
```

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

```
[215]: # Desplazamiento= -1 y eje horizontal
np.roll(b, -1, axis=0)
```

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

```
[216]: # Desplazamiento = -1 y eje vertical
np.roll(b, -1, axis=1)
```

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

6.2.6 logspace()

```
[217]: # Array de elementos logarítmicos espaciados
```

```
[218]: # np.logspace(10^inicio, 10^fin, divisiones(elementos))
# como en linspace se incluye los extremos (inicios-->fin)
```



```
c = np.logspace(0, 1, 3)
c
```

```
[218]: array([ 1.          ,  3.16227766, 10.          ])
```

```
[219]: #  $10^0 = 1$  ;  $10^1 = 10$  ; 3 divisiones(elementos)
```

$$10^0 = 1 \text{ y } 10^1 = 10 \text{ 3 divisiones(elementos)}$$

6.3 Slicing

6.3.1 Acceso a un elemento de un array:

```
[220]: matriz = np.array([
        [10, 20],
        [30, 40]
    ])
matriz
```

```
[220]: array([[10, 20],
             [30, 40]])
```

```
[221]: matriz[0][0] # fila 0 columna 0
```

```
[221]: np.int64(10)
```

```
[222]: matriz[0][1] # fila 0 columna 1
```

```
[222]: np.int64(20)
```

Otro ejemplo...

```
[223]: q = np.array([[1, 2, 3],
                    [4, 5, 6],
                    [7, 8, 9]])
q
```

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

```
[224]: # Opción 1
q[2][1] # --> fila 2 y columna 1 (listas 0, 1, 2)
```

```
[224]: np.int64(8)
```

```
[225]: q[0][2]
```

```
[225]: np.int64(3)
```

```
[226]: # Opción 2
q[2, 1]
```

```
[226]: np.int64(8)
```

```
[227]: # dos primeras filas (: --> todas)
q[:2]
```

```
[227]: array([[1, 2, 3],
            [4, 5, 6]])
```

```
[228]: q[2:]
```

```
[228]: array([[7, 8, 9]])
```

```
[229]: # Filtrar por columnas
q[:,0]
```

```
[229]: array([[1],
            [4],
            [7]])
```

```
[230]: # Filtrar por columnas
q[:,0,1]
```

```
[230]: array([[1, 2],
            [4, 5],
            [7, 8]])
```

```
[231]: # También sigue como las listas [start:stop:step]

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
x[1:12:2]
```

```
[231]: array([1, 3, 5, 7, 9])
```

6.3.2 Array de 5 x 5

```
[232]: a = np.array([
            [1, 2, 3, 4, 5],
            [6, 7, 8, 9, 10],
            [11, 12, 13, 14, 15],
            [16, 17, 18, 19, 20],
            [21, 22, 23, 24, 25]
        ])
a
```

```
[232]: array([[ 1,  2,  3,  4,  5],
             [ 6,  7,  8,  9, 10],
             [11, 12, 13, 14, 15],
             [16, 17, 18, 19, 20],
             [21, 22, 23, 24, 25]])
```

6.3.3 Imprimir desde la 3ª columna hasta el final

```
[233]: a # mostrar la información de la matriz
```

```
[233]: array([[ 1,  2,  3,  4,  5],
             [ 6,  7,  8,  9, 10],
             [11, 12, 13, 14, 15],
             [16, 17, 18, 19, 20],
             [21, 22, 23, 24, 25]])
```

```
[234]: # ojo, empezamos contando 0...(0-1-2) hasta la columna 2 (la tercera)
# : antes del igual indica todas las filas
# todas las filas, las columnas de 0 hasta 2 (2 no incluida)
a[:, :2]
```

```
[234]: array([[ 1,  2],
             [ 6,  7],
             [11, 12],
             [16, 17],
             [21, 22]])
```

```
[235]: # todas las columnas de las 2 primeras filas
a[:2]
```

```
[235]: array([[ 1,  2,  3,  4,  5],
             [ 6,  7,  8,  9, 10]])
```

```
[236]: a[:2, :]
```

```
[236]: array([[ 1,  2,  3,  4,  5],
             [ 6,  7,  8,  9, 10]])
```

```
[237]: a[:, 1:2]
```

```
[237]: array([[ 2],
             [ 7],
             [12],
             [17],
             [22]])
```

```
[238]: # NOTA: esta parte será importante para el tema de visualización de los datos
      ↪ en dataframe,
```

```
# ver el tema de df.loc o df.iloc
```

Type...

```
[239]: type(a[:,2:])
```

```
[239]: numpy.ndarray
```

6.3.4 Imprimo desde la primera columna hasta la 2ª (incluida)

```
[240]: a
```

```
[240]: array([[ 1,  2,  3,  4,  5],
           [ 6,  7,  8,  9, 10],
           [11, 12, 13, 14, 15],
           [16, 17, 18, 19, 20],
           [21, 22, 23, 24, 25]])
```

```
[241]: # Opción 1
```

```
a[:, :2]
```

```
[241]: array([[ 1,  2],
           [ 6,  7],
           [11, 12],
           [16, 17],
           [21, 22]])
```

```
[242]: # Opción 2
```

```
a[:, 0:2]
```

```
[242]: array([[ 1,  2],
           [ 6,  7],
           [11, 12],
           [16, 17],
           [21, 22]])
```

6.3.5 Imprimo las pares

```
[243]: a
```

```
[243]: array([[ 1,  2,  3,  4,  5],
           [ 6,  7,  8,  9, 10],
           [11, 12, 13, 14, 15],
           [16, 17, 18, 19, 20],
           [21, 22, 23, 24, 25]])
```

```
[244]: # ":" antes de la coma equivale a todas las filas
# inicio:final:incremento (si añades un segundo ":" es poner el incremento)
# en el final si no ponemos nada es el final
```

```
[245]: a[:, 1::2]
```

```
[245]: array([[ 2,  4],
           [ 7,  9],
           [12, 14],
           [17, 19],
           [22, 24]])
```

```
[246]: a[:, 1::3]
```

```
[246]: array([[ 2,  5],
           [ 7, 10],
           [12, 15],
           [17, 20],
           [22, 25]])
```

6.3.6 Imprimir las impares

```
[247]: a
```

```
[247]: array([[ 1,  2,  3,  4,  5],
           [ 6,  7,  8,  9, 10],
           [11, 12, 13, 14, 15],
           [16, 17, 18, 19, 20],
           [21, 22, 23, 24, 25]])
```

```
[248]: a[:, 0::2]
```

```
[248]: array([[ 1,  3,  5],
           [ 6,  8, 10],
           [11, 13, 15],
           [16, 18, 20],
           [21, 23, 25]])
```

```
[249]: a[:, 0:2:2]
```

```
[249]: array([[ 1],
           [ 6],
           [11],
           [16],
           [21]])
```

```
[250]: a[:, 0:3:2]
```

```
[250]: array([[ 1,  3],
              [ 6,  8],
              [11, 13],
              [16, 18],
              [21, 23]])
```

6.4 Comparacion entre Arrays

```
[251]: # Creamos los arrays
```

```
[252]: s = np.array([
          [1, 2, 3],
          [4, 5, 6]
        ])
s
```

```
[252]: array([[1, 2, 3],
              [4, 5, 6]])
```

```
[253]: t = np.array([
          [100, 200, 3],
          [400, 5, 6]
        ])
t
```

```
[253]: array([[100, 200,  3],
              [400,   5,  6]])
```

Los comparo

`np.where(condicion, si es cierto, si es falso)`

```
[254]: np.where(s==t, "True", "False")
```

```
[254]: array([[ 'False', 'False', 'True'],
              [ 'False', 'True', 'True']], dtype='<U5')
```

```
[255]: np.where(s==t, "Si", "No")
```

```
[255]: array([[ 'No', 'No', 'Si'],
              [ 'No', 'Si', 'Si']], dtype='<U2')
```

```
[256]: np.where(s==t, 1, 0)
```

```
[256]: array([[0, 0, 1],
              [0, 1, 1]])
```

6.5 Concatenación de arrays

Crear los arrays

```
[257]: y = np.array([
        [1, 2],
        [3, 4]
    ])
y
```

```
[257]: array([[1, 2],
             [3, 4]])
```

```
[258]: z = np.array([
        [5, 6]
    ])
z
```

Concatenación por filas

```
[259]: np.concatenate((y,z), axis=0)
```

```
[259]: array([[1, 2],
             [3, 4],
             [5, 6]])
```

Concatenación por columnas

```
[260]: z1 = z.transpose()
z1
```

```
[260]: array([[5],
             [6]])
```

```
[261]: np.concatenate((y,z1), axis=1)
```

```
[261]: array([[1, 2, 5],
             [3, 4, 6]])
```

6.6 Operaciones

```
[262]: # Potencias
```

```
[263]: r = np.array([1, 2, 3, 4])
r
```

```
[263]: array([1, 2, 3, 4])
```

```
[264]: # Método 1
```

```
[265]: r**2 # 11, 22, 33, 44
```

```
[265]: array([ 1,  4,  9, 16])
```

```
[266]: # Método 2
```

```
[267]: pow(r, 2)
```

```
[267]: array([ 1,  4,  9, 16])
```

6.6.1 Producto escalar y producto vectorial de 2 vectores

```
[268]: w = np.array([1, 2, 3])  
w
```

```
[268]: array([1, 2, 3])
```

```
[269]: x = np.array([2, 5, -4])  
x
```

```
[269]: array([ 2,  5, -4])
```

Producto escalar:

```
[270]: # w * x = ((1*2) + (2*5) + (3*-4))
```

```
[271]: # np.dot(matriz1, matriz2)  
np.dot(w,x)
```

```
[271]: np.int64(0)
```

Producto Vectorial:

```
[272]: ## Producto Vectorial  
  
# i j k  
# 1 2 3  
# 2 5 -4  
  
# y se opera:  
# -8i+5K+6j -(-4k-4j+15i) = -23i+10j+1k --> (-23, 10, 1)
```

```
[273]: np.cross(w, x)
```

```
[273]: array([-23,  10,   1])
```


6.6.2 Matriz con “matrix”

```
[274]: # 4 filas 4 columnas
u = np.matrix([
    [4, -3, 11, 1],
    [5, 9, 7, 2],
    [2, 3, 4, 1],
    [5, 3, -5, -9]
])
u
```

```
[274]: matrix([[ 4, -3, 11,  1],
               [ 5,  9,  7,  2],
               [ 2,  3,  4,  1],
               [ 5,  3, -5, -9]])
```

```
[275]: # 1 fila y 4 columnas
v = np.matrix([4, 9, 1, 3])
v
```

```
[275]: matrix([[4, 9, 1, 3]])
```

Suma

```
[276]: u + v
```

```
[276]: matrix([[ 8,  6, 12,  4],
               [ 9, 18,  8,  5],
               [ 6, 12,  5,  4],
               [ 9, 12, -4, -6]])
```

Resta

```
[277]: u - v
```

```
[277]: matrix([[ 0, -12, 10, -2],
               [ 1,  0,  6, -1],
               [-2, -6,  3, -2],
               [ 1, -6, -6, -12]])
```

Producto

```
[278]: u * v
```

```
-----
ValueError
Cell In[278], line 1
----> 1 u * v
```

Traceback (most recent call last)

```
File ~/Documentos/Python_DS/env/lib/python3.12/site-packages/numpy/matrixlib/
↳ defmatrix.py:224, in matrix.__mul__(self, other)
    221 def __mul__(self, other):
    222     if isinstance(other, (N.ndarray, list, tuple)) :
    223         # This promotes 1-D vectors to row vectors
--> 224         return N.dot(self, asmatrix(other))
    225     if isscalar(other) or not hasattr(other, '__rmul__') :
    226         return N.dot(self, other)
```

ValueError: shapes (4,4) and (1,4) not aligned: 4 (dim 1) != 1 (dim 0)

[279]: *# ValueError --> es necesario realizar la transpuesta para este caso, ya que*
↳ las dimensiones No son las adecuadas

Opción 1:

```
[280]: u*v.transpose()
```

```
[280]: matrix([[ 3],
               [114],
               [ 42],
               [ 15]])
```

Opción 2:

```
[281]: u*v.T
```

```
[281]: matrix([[ 3],
               [114],
               [ 42],
               [ 15]])
```

Opción 3:

```
[282]: np.dot(u, v.T)
```

```
[282]: matrix([[ 3],
               [114],
               [ 42],
               [ 15]])
```

Traza de una matriz

(suma de los elementos de la diagonal principal)

```
[283]: u -v
```

```
[283]: matrix([[ 0, -12, 10, -2],
               [ 1,  0,  6, -1],
               [-2, -6,  3, -2],
```

```
[ 1, -6, -6, -12]])
```

```
[284]: type(u-v)
```

```
[284]: numpy.matrix
```

```
[285]: np.trace(u-v) # 0 + 0 + 3 + (-12) = -9 (suma de los elementos de la diagonal principal)
```

```
[285]: np.int64(-9)
```

6.7 Mathematical functions

6.7.1 Trigonometric functions

Functions	Description
<code>sin(x, /[, out, where, casting, order, ...])</code>	Trigonometric sine, element-wise.
<code>cos(x, /[, out, where, casting, order, ...])</code>	Cosine element-wise.
<code>tan(x, /[, out, where, casting, order, ...])</code>	Compute tangent element-wise.
<code>arcsin(x, /[, out, where, casting, order, ...])</code>	Inverse sine, element-wise.
<code>asin(x, /[, out, where, casting, order, ...])</code>	Inverse sine, element-wise.
<code>arccos(x, /[, out, where, casting, order, ...])</code>	Trigonometric inverse cosine, element-wise.
<code>acos(x, /[, out, where, casting, order, ...])</code>	Trigonometric inverse cosine, element-wise.
<code>arctan(x, /[, out, where, casting, order, ...])</code>	Trigonometric inverse tangent, element-wise.
<code>atan(x, /[, out, where, casting, order, ...])</code>	Trigonometric inverse tangent, element-wise.
<code>hypot(x1, x2, /[, out, where, casting, ...])</code>	Given the “legs” of a right triangle, return its hypotenuse.
<code>degrees(x, /[, out, where, casting, order, ...])</code>	Convert angles from radians to degrees.
<code>radians(x, /[, out, where, casting, order, ...])</code>	Convert angles from degrees to radians.

6.7.2 Rounding

Functions	Description
<code>round(a[, decimals, out])</code>	Evenly round to the given number of decimals.
<code>around(a[, decimals, out])</code>	Round an array to the given number of decimals.
<code>rint(x, /[, out, where, casting, order, ...])</code>	Round elements of the array to the nearest integer.
<code>fix(x[, out])</code>	Round to nearest integer towards zero.
<code>floor(x, /[, out, where, casting, order, ...])</code>	Return the floor of the input, element-wise.
<code>ceil(x, /[, out, where, casting, order, ...])</code>	Return the ceiling of the input, element-wise.
<code>trunc(x, /[, out, where, casting, order, ...])</code>	Return the truncated value of the input, element-wise.

6.7.3 Sums, products, differences

Functions	Description
<code>prod(a[, axis, dtype, out, keepdims, ...])</code>	Return the product of array elements over a given axis.
<code>sum(a[, axis, dtype, out, keepdims, ...])</code>	Sum of array elements over a given axis.
<code>nanprod(a[, axis, dtype, out, keepdims, ...])</code>	Return the product of array elements over a given axis treating Not a Numbers (NaNs) as ones.
<code>nansum(a[, axis, dtype, out, keepdims, ...])</code>	Return the sum of array elements over a given axis treating Not a Numbers (NaNs) as zero.
<code>cumprod(a[, axis, dtype, out])</code>	Return the cumulative product of elements along a given axis.
<code>cumsum(a[, axis, dtype, out])</code>	Return the cumulative sum of the elements along a given axis.
<code>gradient(f, *varargs[, axis, edge_order])</code>	Return the gradient of an N-dimensional array.
<code>cross(a, b[, axisa, axisb, axisc, axis])</code>	Return the cross product of two (arrays of) vectors.

6.7.4 Arithmetic operations

Functions	Description
<code>add(x1, x2, /[, out, where, casting, order, ...])</code>	Add arguments element-wise.
<code>reciprocal(x, /[, out, where, casting, ...])</code>	Return the reciprocal of the argument, element-wise.
<code>positive(x, /[, out, where, casting, order, ...])</code>	Numerical positive, element-wise.
<code>negative(x, /[, out, where, casting, order, ...])</code>	Numerical negative, element-wise.
<code>multiply(x1, x2, /[, out, where, casting, ...])</code>	Multiply arguments element-wise.
<code>divide(x1, x2, /[, out, where, casting, ...])</code>	Divide arguments element-wise.
<code>power(x1, x2, /[, out, where, casting, ...])</code>	First array elements raised to powers from second array, element-wise.
<code>pow(x1, x2, /[, out, where, casting, order, ...])</code>	First array elements raised to powers from second array, element-wise.
<code>subtract(x1, x2, /[, out, where, casting, ...])</code>	Subtract arguments, element-wise.
<code>true_divide(x1, x2, /[, out, where, ...])</code>	Divide arguments element-wise.
<code>floor_divide(x1, x2, /[, out, where, ...])</code>	Return the largest integer smaller or equal to the division of the inputs.
<code>float_power(x1, x2, /[, out, where, ...])</code>	First array elements raised to powers from second array, element-wise.
<code>fmod(x1, x2, /[, out, where, casting, ...])</code>	Returns the element-wise remainder of division.

Functions	Description
<code>mod(x1, x2, /[, out, where, casting, order, ...])</code>	Returns the element-wise remainder of division.

6.7.5 Extrema finding

Functions	Description
<code>maximum(x1, x2, /[, out, where, casting, ...])</code>	Element-wise maximum of array elements.
<code>max(a[, axis, out, keepdims, initial, where])</code>	Return the maximum of an array or maximum along an axis.
<code>minimum(x1, x2, /[, out, where, casting, ...])</code>	Element-wise minimum of array elements.
<code>min(a[, axis, out, keepdims, initial, where])</code>	Return the minimum of an array or minimum along an axis.

Creado por:

Isabel Maniega