Creado por:

Isabel Maniega

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.

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

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

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

```
/tmp/ipykernel_7159/2162656668.py:1: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major rele
ase of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and b
etter interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/
54466

  import pandas as pd
```

In [4]:
```
lista = [1, 2, 5, 9, None, 47, 20]
lista
```

Out[4]:
```
[1, 2, 5, 9, None, 47, 20]
```

In [5]:
```
# Series:

s = pd.Series(lista)
s
```

Out[5]:
```
0     1.0
1     2.0
2     5.0
3     9.0
4     NaN
5    47.0
6    20.0
dtype: float64
```

In [6]:
```
# Series:

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

Out[6]:
```
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

In [7]:
```
# date_range(genera un rango de fecha apartir de un valor, marcando el nú
dates = pd.date_range("20130101", periods=6)
dates
```

Out[7]:
```
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')
```

In [8]:
```
df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list("ABCD"
df
```

Out[8]:

	A	B	C	D
2013-01-01	-0.711362	0.825269	-0.532520	-2.232001
2013-01-02	1.530446	-1.099274	-0.275519	-0.607721
2013-01-03	-2.434541	-1.111805	2.101658	1.161692
2013-01-04	-0.323204	-0.416781	0.613086	-0.251889
2013-01-05	-1.008649	-1.469483	0.297839	-0.798676
2013-01-06	0.035385	-1.806611	0.660986	-0.421329

In [9]:
```
# dtypes nos muestra de que tipo son los datos:

df.dtypes
```

Out[9]:
```
A    float64
B    float64
C    float64
D    float64
dtype: object
```

Vista de los datos

In [10]:
```
# Muestra las primeras filas del dataframe, por defecto las 5 primeras
df.head()
```

Out[10]:

	A	B	C	D
2013-01-01	-0.711362	0.825269	-0.532520	-2.232001
2013-01-02	1.530446	-1.099274	-0.275519	-0.607721
2013-01-03	-2.434541	-1.111805	2.101658	1.161692
2013-01-04	-0.323204	-0.416781	0.613086	-0.251889
2013-01-05	-1.008649	-1.469483	0.297839	-0.798676

In [11]:
```
df.head(2)
```

Out[11]:

	A	B	C	D
2013-01-01	-0.711362	0.825269	-0.532520	-2.232001
2013-01-02	1.530446	-1.099274	-0.275519	-0.607721

In [12]:
```
# Muestra las últimas filas de un dataframe, por defecto las 5 últimas:

df.tail()
```

Out[12]:

	A	B	C	D
2013-01-02	1.530446	-1.099274	-0.275519	-0.607721
2013-01-03	-2.434541	-1.111805	2.101658	1.161692
2013-01-04	-0.323204	-0.416781	0.613086	-0.251889
2013-01-05	-1.008649	-1.469483	0.297839	-0.798676
2013-01-06	0.035385	-1.806611	0.660986	-0.421329

In [13]:
```
df.tail(2)
```

Out[13]:

	A	B	C	D
2013-01-05	-1.008649	-1.469483	0.297839	-0.798676
2013-01-06	0.035385	-1.806611	0.660986	-0.421329

In [14]:
```
# Muestra el valor de la primera columna que suele ser un valor único (id

df.index
```

Out[14]:
```
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')
```

In [15]:
```
# Muestra el nombre de las columnas:

df.columns
```

Out[15]:
```
Index(['A', 'B', 'C', 'D'], dtype='object')
```

In [16]:
```
# Podemos convertir un dataframe en una matriz de numpy con:

df.to_numpy()
```

Out[16]:
```
array([[-0.71136247,  0.82526885, -0.53251966, -2.23200059],
       [ 1.530446  , -1.09927375, -0.27551884, -0.60772076],
       [-2.43454111, -1.1118055 ,  2.10165828,  1.16169221],
       [-0.32320382, -0.4167814 ,  0.61308623, -0.25188882],
       [-1.00864866, -1.46948307,  0.2978386 , -0.79867647],
       [ 0.03538463, -1.80661127,  0.66098596, -0.42132873]])
```

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

print(df.to_numpy())
```

```
[[-0.71136247  0.82526885 -0.53251966 -2.23200059]
 [ 1.530446   -1.09927375 -0.27551884 -0.60772076]
 [-2.43454111 -1.1118055   2.10165828  1.16169221]
 [-0.32320382 -0.4167814   0.61308623 -0.25188882]
 [-1.00864866 -1.46948307  0.2978386  -0.79867647]
 [ 0.03538463 -1.80661127  0.66098596 -0.42132873]]
```

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

df.describe()
```

Out[18]:

	A	B	C	D
count	6.000000	6.000000	6.000000	6.000000
mean	-0.485321	-0.846448	0.477588	-0.524987
std	1.302702	0.940603	0.928375	1.088656
min	-2.434541	-1.806611	-0.532520	-2.232001
25%	-0.934327	-1.380064	-0.132179	-0.750938
50%	-0.517283	-1.105540	0.455462	-0.514525
75%	-0.054262	-0.587404	0.649011	-0.294249
max	1.530446	0.825269	2.101658	1.161692

In [19]:
```
# Podemos dar la vuelta a la tabla y poner lo que esta en filas en column

df.T
```

Out[19]:

	2013-01-01	2013-01-02	2013-01-03	2013-01-04	2013-01-05	2013-01-06
A	-0.711362	1.530446	-2.434541	-0.323204	-1.008649	0.035385
B	0.825269	-1.099274	-1.111805	-0.416781	-1.469483	-1.806611
C	-0.532520	-0.275519	2.101658	0.613086	0.297839	0.660986
D	-2.232001	-0.607721	1.161692	-0.251889	-0.798676	-0.421329

In [20]:

```
# Colocar los valores según el indice:

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

Out[20]:

	D	C	B	A
2013-01-01	-2.232001	-0.532520	0.825269	-0.711362
2013-01-02	-0.607721	-0.275519	-1.099274	1.530446
2013-01-03	1.161692	2.101658	-1.111805	-2.434541
2013-01-04	-0.251889	0.613086	-0.416781	-0.323204
2013-01-05	-0.798676	0.297839	-1.469483	-1.008649
2013-01-06	-0.421329	0.660986	-1.806611	0.035385

In [21]:

```
# Ordenar los datos según una columna:

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

Out[21]:

	A	B	C	D
2013-01-06	0.035385	-1.806611	0.660986	-0.421329
2013-01-05	-1.008649	-1.469483	0.297839	-0.798676
2013-01-03	-2.434541	-1.111805	2.101658	1.161692
2013-01-02	1.530446	-1.099274	-0.275519	-0.607721
2013-01-04	-0.323204	-0.416781	0.613086	-0.251889
2013-01-01	-0.711362	0.825269	-0.532520	-2.232001

Seleccion

`GetItem()`

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

In [22]:

```
df['A']
```

Out[22]:
```
2013-01-01   -0.711362
2013-01-02    1.530446
2013-01-03   -2.434541
2013-01-04   -0.323204
2013-01-05   -1.008649
2013-01-06    0.035385
Freq: D, Name: A, dtype: float64
```

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

Out[23]:
```
2013-01-01   -0.711362
2013-01-02    1.530446
2013-01-03   -2.434541
2013-01-04   -0.323204
2013-01-05   -1.008649
2013-01-06    0.035385
Freq: D, Name: A, dtype: float64
```

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

Out[24]:

	A
2013-01-01	-0.711362
2013-01-02	1.530446
2013-01-03	-2.434541
2013-01-04	-0.323204
2013-01-05	-1.008649
2013-01-06	0.035385

Selección de filas mediante slicing(:)

In [25]:
```
df
```

Out[25]:

	A	B	C	D
2013-01-01	-0.711362	0.825269	-0.532520	-2.232001
2013-01-02	1.530446	-1.099274	-0.275519	-0.607721
2013-01-03	-2.434541	-1.111805	2.101658	1.161692
2013-01-04	-0.323204	-0.416781	0.613086	-0.251889
2013-01-05	-1.008649	-1.469483	0.297839	-0.798676
2013-01-06	0.035385	-1.806611	0.660986	-0.421329

In [26]:
```
df[0:2]
```

Out[26]:

	A	B	C	D
2013-01-01	-0.711362	0.825269	-0.532520	-2.232001
2013-01-02	1.530446	-1.099274	-0.275519	-0.607721

In [27]:
```
df["20130103":"20130105"]
```

Out[27]:

	A	B	C	D
2013-01-03	-2.434541	-1.111805	2.101658	1.161692
2013-01-04	-0.323204	-0.416781	0.613086	-0.251889
2013-01-05	-1.008649	-1.469483	0.297839	-0.798676

Selección con la función `loc[]` y `at[]`

In [28]:

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

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

Out[28]:

```
A   -0.711362
B    0.825269
C   -0.532520
D   -2.232001
Name: 2013-01-01 00:00:00, dtype: float64
```

In [29]:

```
# Seleccionar todas las filas de una determinada columna:

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

Out[29]:

	B	C
2013-01-01	0.825269	-0.532520
2013-01-02	-1.099274	-0.275519
2013-01-03	-1.111805	2.101658
2013-01-04	-0.416781	0.613086
2013-01-05	-1.469483	0.297839
2013-01-06	-1.806611	0.660986

In [30]:

```
# Seleccionar por filas y columnas:

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

Out[30]:

	B	C
2013-01-03	-1.111805	2.101658
2013-01-04	-0.416781	0.613086
2013-01-05	-1.469483	0.297839

In [31]:

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

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

Out[31]:

```
-1.1118054961800448
```

In [32]:

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

Out[32]:

```
-1.1118054961800448
```

Selección por posicion: método `iloc[]` y `iat[]`

In [33]:
```
# Selección de una fila en posición 3:

df.iloc[3]
```

Out[33]:
```
A   -0.323204
B   -0.416781
C    0.613086
D   -0.251889
Name: 2013-01-04 00:00:00, dtype: float64
```

In [34]:
```
# Selección de una fila y columna por slicing:

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

Out[34]:

	B	C
2013-01-04	-0.416781	0.613086
2013-01-05	-1.469483	0.297839

In [43]:
```
# Selección por lista de posiciones:
# Filas: 1, 2, 4
# Columnas: 0(A), 2(C)

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

Out[43]:

	A	C
2013-01-02	1.530446	-0.275519
2013-01-03	-2.434541	2.101658
2013-01-05	-1.008649	0.297839

In [36]:
```
# Selección por filas o columnas:

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

Out[36]:

	A	B	C	D
2013-01-02	1.530446	-1.099274	-0.275519	-0.607721
2013-01-03	-2.434541	-1.111805	2.101658	1.161692

In [37]:
```
df.iloc[:, 1:3]
```

Out[37]:

	B	C
2013-01-01	0.825269	-0.532520
2013-01-02	-1.099274	-0.275519
2013-01-03	-1.111805	2.101658
2013-01-04	-0.416781	0.613086
2013-01-05	-1.469483	0.297839
2013-01-06	-1.806611	0.660986

In [38]:
```
# Seleccionar un valor concreto por posición (2013-01-03, 'B'):

df.iloc[2, 1]
```

Out[38]: -1.1118054961800448

In [39]:
```
df.iat[2, 1]
```

Out[39]: -1.1118054961800448

Boolean indexing

In [40]:
```
# Selección por comparativa:

df[df['A'] >= 0.2]
```

Out[40]:

	A	B	C	D
2013-01-02	1.530446	-1.099274	-0.275519	-0.607721

In [41]:
```
df[df > 0]
```

Out[41]:

	A	B	C	D
2013-01-01	NaN	0.825269	NaN	NaN
2013-01-02	1.530446	NaN	NaN	NaN
2013-01-03	NaN	NaN	2.101658	1.161692
2013-01-04	NaN	NaN	0.613086	NaN
2013-01-05	NaN	NaN	0.297839	NaN
2013-01-06	0.035385	NaN	0.660986	NaN

Método `isin()`

In [42]:
```
# Selección según una coincidencia (filtrado):

df2 = pd.DataFrame(["one", "one", "two", "three", "four", "three"], colum

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

Out[42]:

	E
0	one
1	one
4	four

Setting (Modificacion del dataframe)

In [44]:

```
# Añadir Valores nuevo

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

Out[44]:

```
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
```

In [45]:

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

Out[45]:

	A	B	C	D	E
2013-01-01	-0.711362	0.825269	-0.532520	-2.232001	1
2013-01-02	1.530446	-1.099274	-0.275519	-0.607721	2
2013-01-03	-2.434541	-1.111805	2.101658	1.161692	3
2013-01-04	-0.323204	-0.416781	0.613086	-0.251889	4
2013-01-05	-1.008649	-1.469483	0.297839	-0.798676	5
2013-01-06	0.035385	-1.806611	0.660986	-0.421329	6

In [46]:

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

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

Out[46]:

	A	B	C	D	E
2013-01-01	0.000000	0.825269	-0.532520	-2.232001	1
2013-01-02	1.530446	-1.099274	-0.275519	-0.607721	2
2013-01-03	-2.434541	-1.111805	2.101658	1.161692	3
2013-01-04	-0.323204	-0.416781	0.613086	-0.251889	4
2013-01-05	-1.008649	-1.469483	0.297839	-0.798676	5
2013-01-06	0.035385	-1.806611	0.660986	-0.421329	6

In [47]:
```
# Modificación de valor por posición
# Se modifica el primer valor de la columna B:

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

Out[47]:

	A	B	C	D	E
2013-01-01	0.000000	0.000000	-0.532520	-2.232001	1
2013-01-02	1.530446	-1.099274	-0.275519	-0.607721	2
2013-01-03	-2.434541	-1.111805	2.101658	1.161692	3
2013-01-04	-0.323204	-0.416781	0.613086	-0.251889	4
2013-01-05	-1.008649	-1.469483	0.297839	-0.798676	5
2013-01-06	0.035385	-1.806611	0.660986	-0.421329	6

In [48]:
```
# Modificación asignada por Numpy usando array:

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

Out[48]:

	A	B	C	D	E
2013-01-01	0.000000	0.000000	-0.532520	5.0	1
2013-01-02	1.530446	-1.099274	-0.275519	5.0	2
2013-01-03	-2.434541	-1.111805	2.101658	5.0	3
2013-01-04	-0.323204	-0.416781	0.613086	5.0	4
2013-01-05	-1.008649	-1.469483	0.297839	5.0	5
2013-01-06	0.035385	-1.806611	0.660986	5.0	6

In [49]:
```
# Modificar según una condición (where):

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

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

Out[49]:

	A	B	C	D	E
2013-01-01	0.000000	0.000000	-0.532520	-5.0	-1
2013-01-02	-1.530446	-1.099274	-0.275519	-5.0	-2
2013-01-03	-2.434541	-1.111805	-2.101658	-5.0	-3
2013-01-04	-0.323204	-0.416781	-0.613086	-5.0	-4
2013-01-05	-1.008649	-1.469483	-0.297839	-5.0	-5
2013-01-06	0.035385	-1.806611	-0.660986	-5.0	-6

Missing values

In [50]:
```
# 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 -0.532520  NaN  1.0
2013-01-02  1.530446 -1.099274 -0.275519  5.0  2.0
2013-01-03 -2.434541 -1.111805  2.101658  5.0  NaN
2013-01-04 -0.323204 -0.416781  0.613086  5.0  NaN
```

In [51]:
```
# Eliminamos los valores nulos con la función dropna(): eliminando cualqu

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

Out[51]:

	A	B	C	D	E
2013-01-02	1.530446	-1.099274	-0.275519	5.0	2.0

In [52]:
```
# Rellenar valores nulos:

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

Out[52]:

	A	B	C	D	E
2013-01-01	0.000000	0.000000	-0.532520	5.0	1.0
2013-01-02	1.530446	-1.099274	-0.275519	5.0	2.0
2013-01-03	-2.434541	-1.111805	2.101658	5.0	5.0
2013-01-04	-0.323204	-0.416781	0.613086	5.0	5.0

In [53]:
```
# isna() nos muestra si en el df hay valores nulo o no, sustituyendo por

pd.isna(df1)
```

Out[53]:

	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

In [54]:
```
# isnull() nos muestra si en el df hay valores nulo o no, sustituyendo po

df1.isnull().sum()
```

Out[54]:
```
A    0
B    0
C    0
D    1
E    2
dtype: int64
```

Operaciones

En estos casos no tiene en cuenta los valores nulos.

In [55]:
```
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()
```

Out[55]:

	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

Tendencia Central

Media

Como calcular la media de las distintas notas:

In [56]:
```
media_1 = df["notas_1"].mean()
media_1
```

Out[56]: 15.5

In [57]:
```
media_2 = df["notas_2"].mean()
media_2
```

Out[57]: 16.8

In [58]:
```
media_3 = df["notas_3"].mean()
media_3
```

Out[58]: 17.6

Mediana

Como calcular la mediana de las distintas notas:

In [59]:
```
mediana_1 = df["notas_1"].median()
mediana_1
```

Out[59]: 15.0

In [60]:
```
mediana_2 = df["notas_2"].median()
mediana_2
```

Out[60]: 16.0

In [61]:
```
mediana_3 = df["notas_3"].median()
mediana_3
```

Out[61]: 16.0

Moda

Como calcular la moda de las distintas notas:

In [62]:
```
moda_1 = df["notas_1"].mode()
moda_1
```

Out[62]:
```
0    14
1    15
Name: notas_1, dtype: int64
```

In [63]:
```
moda_2 = df["notas_2"].mode()
moda_2
```

Out[63]:
```
0    15
1    16
Name: notas_2, dtype: int64
```

In [64]:
```
moda_3 = df["notas_3"].mode()
moda_3
```

Out[64]:
```
0    15
Name: notas_3, dtype: int64
```

In [65]:
```
df.notas_3.value_counts()
```

Out[65]:
```
notas_3
15    3
22    2
16    2
17    1
14    1
24    1
Name: count, dtype: int64
```

Resultados Nota_1:

In [66]:
```
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:

In [67]:
```
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_3:

In [68]:
```
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:

$$ S^2= \frac{∑^n_{i=1}(x_i−X)^2}{n−1} $$

In [69]:
```
var_1 = df["notas_1"].var()
var_1
```

Out[69]: 14.5

In [70]:
```
var_2 = df["notas_2"].var()
var_2
```

Out[70]: 8.399999999999999

In [71]:
```
var_3 = df["notas_3"].var()
var_3
```

Out[71]: 13.155555555555557

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

In [72]:
```
var_1 = df["notas_1"].var(ddof=0)
var_1
```

Out[72]: 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. $$ S^=\sqrt{S^2} $$

In [73]:
```
std_1 = df["notas_1"].std()
std_1
```

Out[73]: 3.8078865529319543

In [74]:
```
std_2 = df["notas_2"].std()
std_2
```

Out[74]: 2.8982753492378874

In [75]:
```
std_3 = df["notas_3"].std()
std_3
```

Out[75]: 3.6270588023294517

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

In [76]:
```
std_1 = df["notas_1"].std(ddof=0)
std_1
```

Out[76]: 3.6124783736376886

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 3.63

In [77]:
```
df.describe()
```

Out[77]:

	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

Union de dataframe

In [78]:
```
iris = pd.read_csv('Iris.csv')
iris = iris.drop(['Id'], axis=1)
iris_setosa = iris[0:50]
iris_setosa
```

```
The history saving thread hit an unexpected error (OperationalError('attem
pt to write a readonly database')).History will not be written to the data
base.
```

Out[78]:

	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

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
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

In [79]:
```
iris_virginica = iris[100:]
iris_virginica
```

Out[79]:

	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

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
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

In [80]:

```
iris_versicolor = pd.read_json('iris_versicolor.json')
iris_versicolor
```

Out[80]:

	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

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
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

`concat()`

In [81]:
```
# 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
```

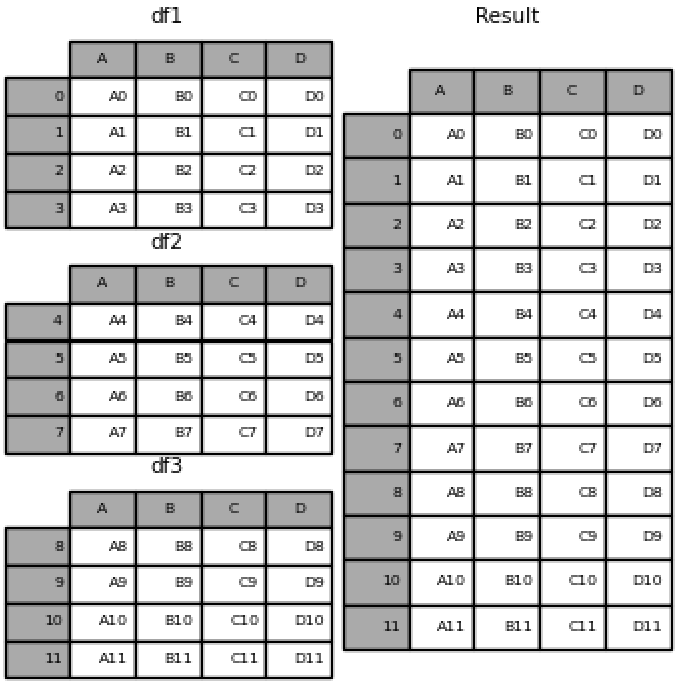
Out[81]:

	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 × 5 columns

In [82]:
```
display.Image('./images/merging_concat_basic.png')
```

Out[82]:



In [85]:
```
iris = pd.read_csv('Iris.csv')
iris_medidas = iris.iloc[:, 0:5]
iris_medidas
```

Out[85]:

	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

150 rows × 5 columns

In [86]:
```
iris_especies = iris[['Species']]
iris_especies
```

Out[86]:

	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 × 1 columns

In [87]:

```
# 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
```

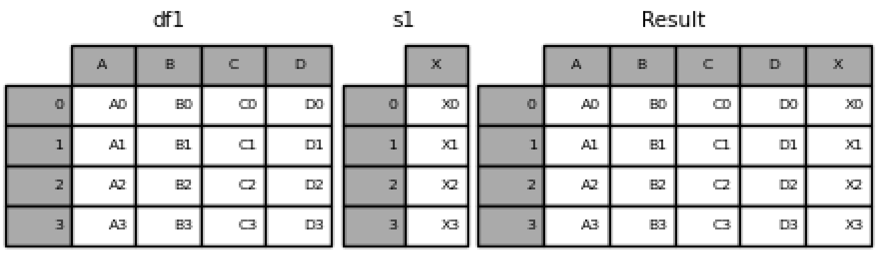
Out[87]:

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

150 rows × 6 columns

In [88]:
```python
display.Image('./images/merging_concat_mixed.png')
```

Out[88]:



`merge()`

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

In [89]:
```python
new_species = iris.loc[:, ['Id', 'Species']]
new_species
```

Out[89]:

	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 × 2 columns

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

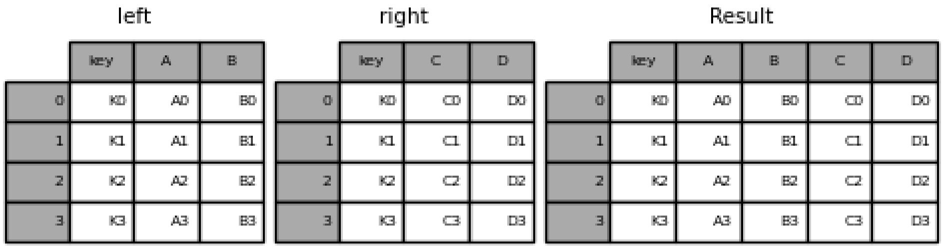
Out[90]:

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

150 rows × 6 columns

In [91]:
```
display.Image('./images/merging_merge_on_key.png')
```

Out[91]:



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

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

Out[92]:

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

150 rows × 6 columns

In [93]:

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

Out[93]:

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

150 rows × 6 columns

In [94]:

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

Out[94]:

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

150 rows × 6 columns

In [95]:

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

Out[95]:

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

150 rows × 6 columns

In [96]:

```
#

new_setosa = pd.merge(iris_medidas, new_species, how='cross')
new_setosa
```

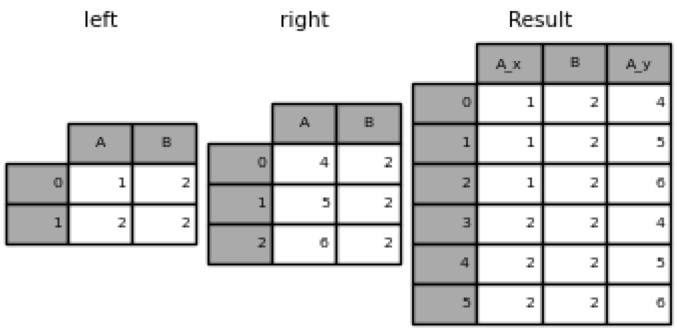
Out[96]:

	Id_x	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Id_y	Sp
0	1	5.1	3.5	1.4	0.2	1	s
1	1	5.1	3.5	1.4	0.2	2	s
2	1	5.1	3.5	1.4	0.2	3	s
3	1	5.1	3.5	1.4	0.2	4	s
4	1	5.1	3.5	1.4	0.2	5	s
...	...	...	...	...	...	...	
22495	150	5.9	3.0	5.1	1.8	146	vir
22496	150	5.9	3.0	5.1	1.8	147	vir
22497	150	5.9	3.0	5.1	1.8	148	vir
22498	150	5.9	3.0	5.1	1.8	149	vir
22499	150	5.9	3.0	5.1	1.8	150	vir

22500 rows × 7 columns

In [98]:
```python
# Si no existe la clave la duplica en el caso how=cross:

display.Image('./images/merging_merge_on_key_dup.png')
```

Out[98]:



left

	A	B
0	1	2
1	2	2

right

	A	B
0	4	2
1	5	2
2	6	2

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

`join()`

In [99]:
```python
iris_medidas
```

Out[99]:

	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

150 rows × 5 columns

In [100…

```
iris_especies
```

Out[100]:

	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 × 1 columns

In [101…

```
iris_2 = iris_medidas.join(iris_especies)
iris_2
```

Out[101]:

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

150 rows × 6 columns

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

Grouping

By “group by” we are referring to a process involving one or more of the following steps:

- **Splitting** the data into groups based on some criteria
- **Applying** a function to each group independently
- **Combining** the results into a data structure

In [102…
```
iris
```

Out[102]:

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

150 rows × 6 columns

In [103…
```
iris_sepal = iris.groupby('Species')[["SepalLengthCm",\
                                      "SepalWidthCm"]].mean()
iris_sepal
```

Out[103]:

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

In [105…
```
iris_petal = iris.groupby('Species')[["PetalLengthCm",\
                                      "PetalWidthCm"]].mean()
iris_petal
```

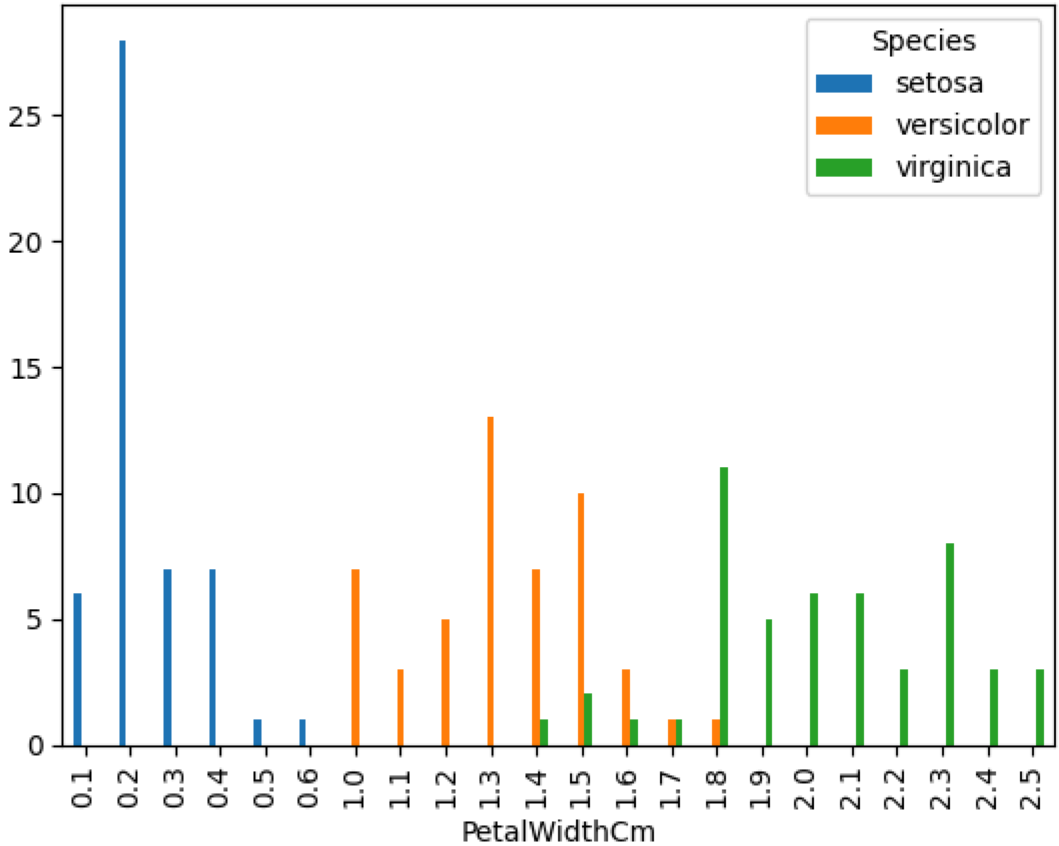
Out[105]:

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

Crosstab

In [146…
```
pd.crosstab(iris.PetalWidthCm, iris.Species).plot(kind='bar')
```

Out[146]: <Axes: xlabel='PetalWidthCm'>



In [300…
```
pd.crosstab(index=iris["Species"],  columns="count")
```

Out[300]:

col_0	count
Species	
setosa	50
versicolor	50
virginica	50

Reshaping

`stack()`

In [106…
```
# Ponemos como columna de index la de especies, asi aplicaremos los datos
# especie sean:

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

Out[106]:

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

150 rows × 5 columns

In [107…

```
stack_iris = reiris.stack(future_stack=True)
stack_iris
```

Out[107]:

```
     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`.

In [108…

```
unstack_iris = reiris.unstack()
unstack_iris
```

Out[108]:

	Id			SepalLengthCm			Se
Species	Iris-setosa	Iris-versicolor	Iris-virginica	Iris-setosa	Iris-versicolor	Iris-virginica	Iris-setosa	Iris versicolo
0	1.0	NaN	NaN	5.1	NaN	NaN	3.5	Nal
1	2.0	NaN	NaN	4.9	NaN	NaN	3.0	Nal
2	3.0	NaN	NaN	4.7	NaN	NaN	3.2	Nal
3	4.0	NaN	NaN	4.6	NaN	NaN	3.1	Nal
4	5.0	NaN	NaN	5.0	NaN	NaN	3.6	Nal
...	...	...	...	...	...	...	...	.
145	NaN	NaN	146.0	NaN	NaN	6.7	NaN	Nal
146	NaN	NaN	147.0	NaN	NaN	6.3	NaN	Nal
147	NaN	NaN	148.0	NaN	NaN	6.5	NaN	Nal
148	NaN	NaN	149.0	NaN	NaN	6.2	NaN	Nal
149	NaN	NaN	150.0	NaN	NaN	5.9	NaN	Nal

150 rows × 15 columns

`pivot_table()`

In [109…

```
# 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
```

Out[109]:

	Id	PetalLengthCm	PetalWidthCm	SepalLengthCm	SepalWidthCm
Species					
Iris-setosa	25.5	1.464	0.244	5.006	3.418
Iris-versicolor	75.5	4.260	1.326	5.936	2.770
Iris-virginica	125.5	5.552	2.026	6.588	2.974

In [110…

```
# Agrupación de datos de especie por media:

iris_pivot2 = pd.pivot_table(iris, index='Species', aggfunc="sum")
iris_pivot2
```

Out[110]:

	Id	PetalLengthCm	PetalWidthCm	SepalLengthCm	SepalWidthCm
Species					
Iris-setosa	1275	73.2	12.2	250.3	170.9
Iris-versicolor	3775	213.0	66.3	296.8	138.5
Iris-virginica	6275	277.6	101.3	329.4	148.7

In [111…

```
# el parametro values nos ayuda a seleccionar las columnas concretas:


iris_pivot = pd.pivot_table(iris, values="PetalLengthCm", index='Species'
iris_pivot
```

Out[111]:

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

Time Series

In [112…

```
# Generamos una serie temporal primero generamos los valores de la fecha
# 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
```

Out[112]:

```
2024-06-01    0.346906
2024-06-02    0.288451
2024-06-03    0.031042
2024-06-04    1.270237
2024-06-05    1.052303
2024-06-06    0.919944
2024-06-07   -0.463178
2024-06-08    0.521568
2024-06-09   -1.929001
2024-06-10   -1.912312
2024-06-11   -0.139794
2024-06-12    0.263261
2024-06-13   -1.394229
2024-06-14    1.274259
2024-06-15    0.174525
Freq: D, dtype: float64
```

`tz_localize()`

In [113…
```
# añadimos la hora al dataframe creado:

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

Out[113]:
```
2024-06-01 00:00:00+00:00    0.346906
2024-06-02 00:00:00+00:00    0.288451
2024-06-03 00:00:00+00:00    0.031042
2024-06-04 00:00:00+00:00    1.270237
2024-06-05 00:00:00+00:00    1.052303
2024-06-06 00:00:00+00:00    0.919944
2024-06-07 00:00:00+00:00   -0.463178
2024-06-08 00:00:00+00:00    0.521568
2024-06-09 00:00:00+00:00   -1.929001
2024-06-10 00:00:00+00:00   -1.912312
2024-06-11 00:00:00+00:00   -0.139794
2024-06-12 00:00:00+00:00    0.263261
2024-06-13 00:00:00+00:00   -1.394229
2024-06-14 00:00:00+00:00    1.274259
2024-06-15 00:00:00+00:00    0.174525
Freq: D, dtype: float64
```

`tz_convert()`

In [114…
```
# Ponemos la franja horaria a la cual nos encontramos:

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

Out[114]:
```
2024-06-01 02:00:00+02:00    0.346906
2024-06-02 02:00:00+02:00    0.288451
2024-06-03 02:00:00+02:00    0.031042
2024-06-04 02:00:00+02:00    1.270237
2024-06-05 02:00:00+02:00    1.052303
2024-06-06 02:00:00+02:00    0.919944
2024-06-07 02:00:00+02:00   -0.463178
2024-06-08 02:00:00+02:00    0.521568
2024-06-09 02:00:00+02:00   -1.929001
2024-06-10 02:00:00+02:00   -1.912312
2024-06-11 02:00:00+02:00   -0.139794
2024-06-12 02:00:00+02:00    0.263261
2024-06-13 02:00:00+02:00   -1.394229
2024-06-14 02:00:00+02:00    1.274259
2024-06-15 02:00:00+02:00    0.174525
Freq: D, dtype: float64
```

`offsets.BusinessDay()`

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

In [115…
```
rng
```

Out[115]:
```
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')
```

In [116…
```
# se añade 5 como número de días a representar:
rng = rng + pd.offsets.BusinessDay(5)
rng
```

Out[116]:
```
DatetimeIndex(['2024-06-07', '2024-06-07', '2024-06-10', '2024-06-11',
               '2024-06-12', '2024-06-13', '2024-06-14', '2024-06-14',
               '2024-06-14', '2024-06-17', '2024-06-18', '2024-06-19',
               '2024-06-20', '2024-06-21', '2024-06-21'],
              dtype='datetime64[ns]', freq=None)
```

In [117…
```
ts = pd.Series(np.random.randn(len(rng)), rng).tz_localize("UTC")
ts
```

Out[117]:
```
2024-06-07 00:00:00+00:00   -0.495060
2024-06-07 00:00:00+00:00    0.788531
2024-06-10 00:00:00+00:00   -1.938738
2024-06-11 00:00:00+00:00   -1.249571
2024-06-12 00:00:00+00:00    1.566901
2024-06-13 00:00:00+00:00   -0.111340
2024-06-14 00:00:00+00:00    1.495749
2024-06-14 00:00:00+00:00    0.437409
2024-06-14 00:00:00+00:00    0.052301
2024-06-17 00:00:00+00:00    0.477631
2024-06-18 00:00:00+00:00    0.497986
2024-06-19 00:00:00+00:00    0.068289
2024-06-20 00:00:00+00:00    0.649090
2024-06-21 00:00:00+00:00   -0.799040
2024-06-21 00:00:00+00:00   -0.434084
dtype: float64
```

In [118…
```
ts.tz_convert("Europe/Madrid")
```

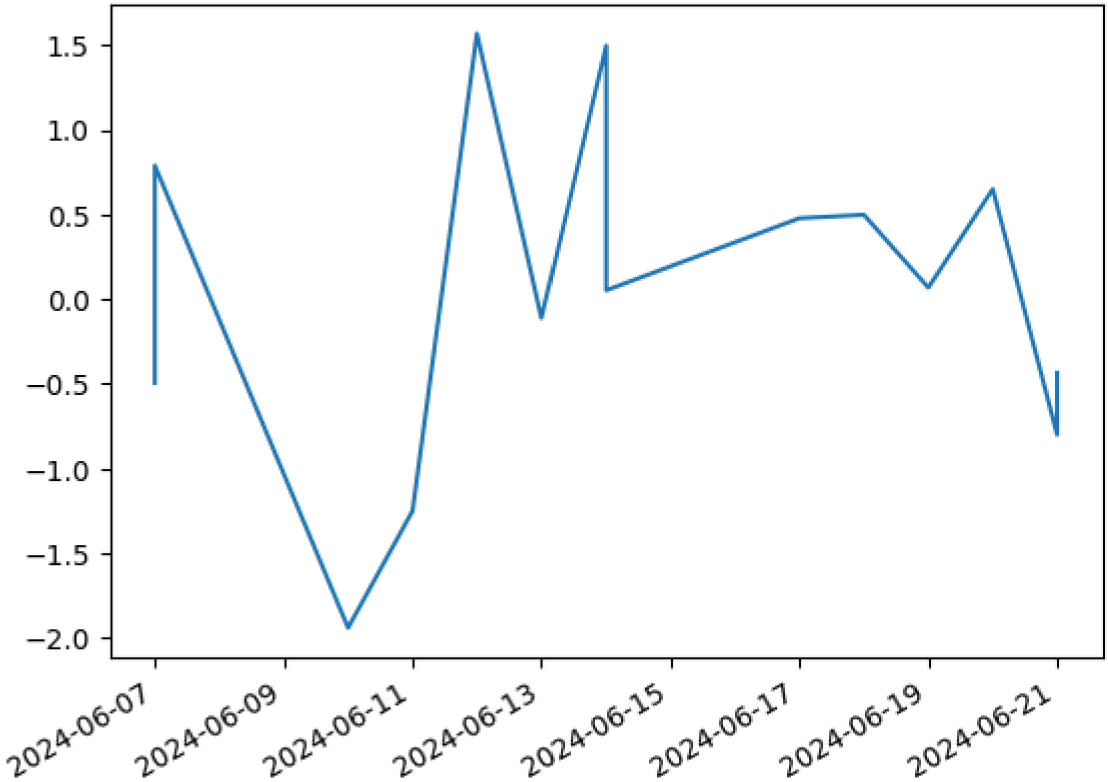
Out[118]:
```
2024-06-07 02:00:00+02:00   -0.495060
2024-06-07 02:00:00+02:00    0.788531
2024-06-10 02:00:00+02:00   -1.938738
2024-06-11 02:00:00+02:00   -1.249571
2024-06-12 02:00:00+02:00    1.566901
2024-06-13 02:00:00+02:00   -0.111340
2024-06-14 02:00:00+02:00    1.495749
2024-06-14 02:00:00+02:00    0.437409
2024-06-14 02:00:00+02:00    0.052301
2024-06-17 02:00:00+02:00    0.477631
2024-06-18 02:00:00+02:00    0.497986
2024-06-19 02:00:00+02:00    0.068289
2024-06-20 02:00:00+02:00    0.649090
2024-06-21 02:00:00+02:00   -0.799040
2024-06-21 02:00:00+02:00   -0.434084
dtype: float64
```

In [120…
```
# pip install matplotlib
```

In [121…
```
import matplotlib.pyplot as plt
```

In [122…
```
ts.plot()
```

Out[122]: <Axes: >



Categoricals

In [123…
```
iris
```

Out[123]:

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

150 rows × 6 columns

In [124…

```
iris.dtypes
```

Out[124]:

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

In [125…

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

Out[125]:

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

`rename_categories()`

In [126…

```
# 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
```

Out[126]:

	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 × 6 columns

`set_categories()`

In [127…

```
# 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
```

Out[127]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species	s
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 × 7 columns

`sort_values()`

In [129…

```
# Colocar las filas según los valores de una columna, en este caso ordena

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

Out[129]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species	s
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	

150 rows × 7 columns

In [130…

```
# Agrupamos para que nos muestre cuantos valores tenemos de cada uno, par
# Incluyen categorias vacias si las hubiera:
```

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

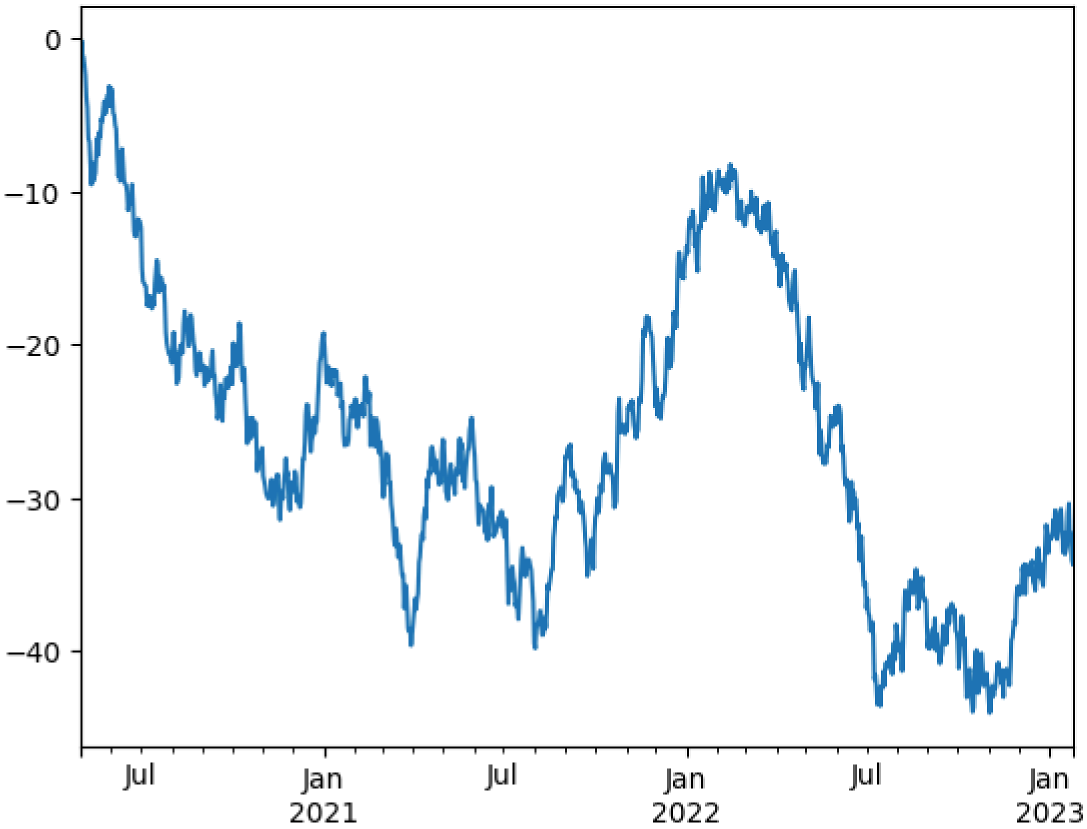
Out[130]:
```
spc
0    50
1    50
2    50
dtype: int64
```

Plotting

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

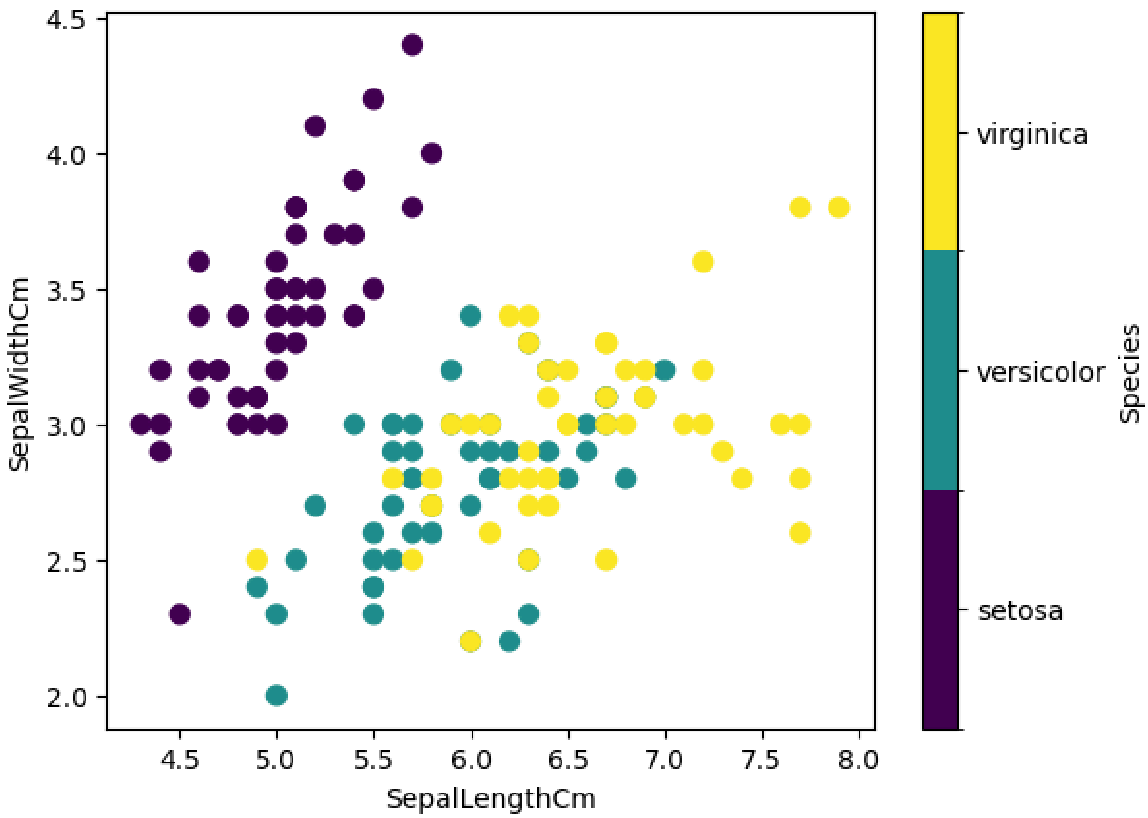
In [131…
```
ts = pd.Series(np.random.randn(1000), index=pd.date_range("5/1/2020",
                                                          periods=1000))

ts = ts.cumsum()

ts.plot()
```

Out[131]: <Axes: >



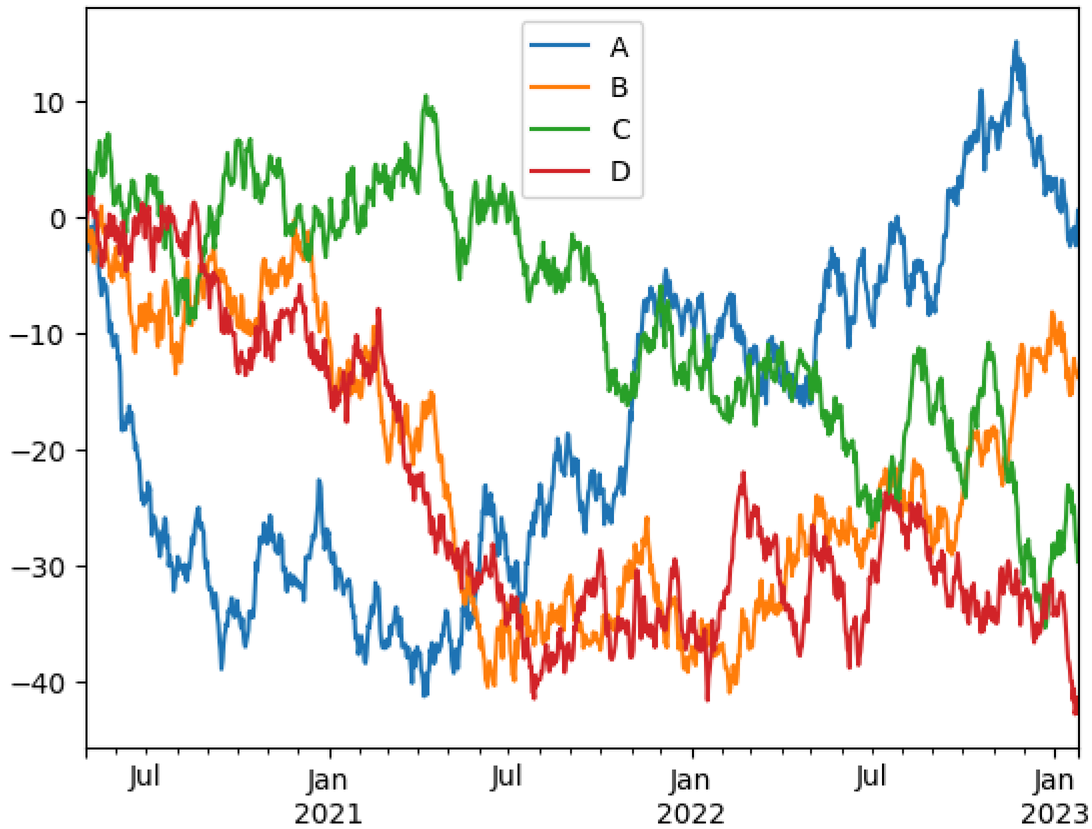
In [134…
```
# c: variable categorica
# cmap: escala de color
# s: tamaño de los puntos

iris.plot.scatter(x='SepalLengthCm', y='SepalWidthCm', c='Species', cmap=
```

Out[134]: <Axes: xlabel='SepalLengthCm', ylabel='SepalWidthCm'>



In [135…
```
# 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')
```

Out[135]:
```
<matplotlib.legend.Legend at 0x7feec83fb610>
<Figure size 640x480 with 0 Axes>
```



Numpy

In [155…
```
# pip install numpy
```

In [156…
```
import numpy as np
```

Método `array()`

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

In [157…
```
e = np.array([
    [1, 2],
    [3, 4],
```

```
    [5, 6]
])
e
```

Out[157]:
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

In [158…
```
len(e)
```

Out[158]:
```
3
```

In [159…
```
e.shape
```

Out[159]:
```
(3, 2)
```

In [160…
```
e.size
```

Out[160]:
```
6
```

In [164…
```
e[0]
```

Out[164]:
```
array([1, 2])
```

In [165…
```
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
```

In [166…
```
a1d = np.array((1, 5, 6))
a1d
```

Out[166]:
```
array([1, 5, 6])
```

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

In [167…
```
np.array([127, 128, 129], dtype=np.int8)
```

```
/tmp/ipykernel_7159/2745257341.py:1: DeprecationWarning: NumPy will stop a
llowing conversion of out-of-bound Python integers to integer arrays.  The
conversion of 128 to int8 will fail in the future.
For the old behavior, usually:
    np.array(value).astype(dtype)
will give the desired result (the cast overflows).
  np.array([127, 128, 129], dtype=np.int8)
/tmp/ipykernel_7159/2745257341.py:1: DeprecationWarning: NumPy will stop a
llowing conversion of out-of-bound Python integers to integer arrays.  The
conversion of 129 to int8 will fail in the future.
For the old behavior, usually:
    np.array(value).astype(dtype)
will give the desired result (the cast overflows).
  np.array([127, 128, 129], dtype=np.int8)
```

Out[167]:
```
array([ 127, -128, -127], dtype=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:

In [168…
```
a = np.array([2, 3, 4], dtype=np.uint32)
print(a)
b = np.array([5, 6, 7], dtype=np.uint32)
print(b)
c = a - b
print(c)
```

```
[2 3 4]
[5 6 7]
[4294967293 4294967293 4294967293]
```

In [169…
```
c_32 = a - b.astype(np.int32)
c_32
```

Out[169]:
```
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()`:

In [170…
```
d = np.dtype(np.int64)
print(d)

# 1º Atributo es el array a testear y 2º Atributo el tipo que queremos co

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

```
int64
True
False
```

Los tipos de datos pueden ser: boleanos (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).

In [171…
```
np.array(["hello", "world"], dtype="S7").tobytes()
```

Out[171]:
```
b'hello\x00\x00world\x00\x00'
```

Método `arange()`.

Numeros dentro de un rango:

Generación de números con numpy en un rango

In [172…
```
a = np.arange(6)
a
```

Out[172]:
```
array([0, 1, 2, 3, 4, 5])
```

In [173…
```
type(a)
```

Out[173]:
```
numpy.ndarray
```

Formas de imprimir la información

In [174…
```
print(a)
```

```
[0 1 2 3 4 5]
```

In [175…
```
for i in a:
    print(i)
```

```
0
1
2
3
4
5
```

Longitud, forma, tamaño

In [176…
```
a
```

Out[176]:
```
array([0, 1, 2, 3, 4, 5])
```

In [177…
```
len(a)
```

Out[177]:
```
6
```

In [178…
```
a.shape
```

Out[178]:
```
(6,)
```

In [179…
```
a.size
```

Out[179]:
```
6
```

Máximos y mínimos

In [180…
```
a
```
Out[180]:
```
array([0, 1, 2, 3, 4, 5])
```

In [181…
```
max(a)
```
Out[181]:
```
5
```

In [182…
```
min(a)
```
Out[182]:
```
0
```

Comprobación de elementos en el array

In [183…
```
a
```
Out[183]:
```
array([0, 1, 2, 3, 4, 5])
```

In [184…
```
25 in a
```
Out[184]:
```
False
```

In [185…
```
0 in a
```
Out[185]:
```
True
```

In [186…
```
25 not in a
```
Out[186]:
```
True
```

In [187…
```
0 not in a
```
Out[187]:
```
False
```

Redefinir el tamaño

In [188…
```
a
```
Out[188]:
```
array([0, 1, 2, 3, 4, 5])
```

In [189…
```
a1 = a.reshape(2, 3)
a1
```
Out[189]:
```
array([[0, 1, 2],
       [3, 4, 5]])
```

Generar números en un intervalo

In [190…
```
# sin especificar va de 1 en 1
b = np.arange(2,7) # 2, 3, 4, 5, 6
b
```
Out[190]:
```
array([2, 3, 4, 5, 6])
```

Generar números en un intervalo con salto

In [191…
```
c = np.arange(10, 40, 5)
c
```

Out[191]:
```
array([10, 15, 20, 25, 30, 35])
```

In [192…
```
d = np.arange(10, 41, 5)
d
```

Out[192]:
```
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:

In [193…
```
# Definimos un array que empice en 2 y acabe en 9 y sean decimales:

np.arange(2, 10, dtype=float)
```

Out[193]:
```
array([2., 3., 4., 5., 6., 7., 8., 9.])
```

`linspace()`

In [ ]:
```
# Recogemos una muestra de los datos, especificamos: min, max, y cada tan
```

In [194…
```
f = np.linspace(10, 20, 2) # de 10 a 20 con 2 elementos
f
```

Out[194]:
```
array([10., 20.])
```

In [195…
```
g = np.linspace(10, 20, 5) # de 10 a 20 muestra 5
g
```

Out[195]:
```
array([10. , 12.5, 15. , 17.5, 20. ])
```

In [196…
```
g1 = np.linspace(10, 20, 3) # de 10 a 20 muestra 3
g1
```

Out[196]:
```
array([10., 15., 20.])
```

Matrices basicas en numpy

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

Matriz Identidad: Diagonal principal llena de 1, resto 0

`eye(n, m)`

In [197…
```
h = np.eye(3) # de 3 filas y 3 columnas --> matriz identidad
```

```
h
```

Out[197]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [198…
```
i = np.eye(5) # Matriz de 5 filas y 5 columnas
i
```

Out[198]:
```
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.]])
```

In [199…
```
# n = filas, m = columnas, el resto que no son de la diagonal las rellena

np.eye(3, 5)
```

Out[199]:
```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.]])
```

`diag()`

In [200…
```
# Los elementos estan en la diagonal principal:

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

Out[200]:
```
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

In [201…
```
# El segundo parámetro es agregar un fila y columna de 0:

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

Out[201]:
```
array([[0, 1, 0, 0],
       [0, 0, 2, 0],
       [0, 0, 0, 3],
       [0, 0, 0, 0]])
```

`vander(x, n)`

In [202…
```
# x = array 1d, la lista o tupla de valores, n = al número de columnas:
np.vander([1, 2, 3, 4], 2)
```

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

In [203…
```
# Se crea una matriz decreciente de los valores 1, 2, 3, 4, que contiene
# así, la primera columna decrece 64, 27, 8, 1
# segunda columna: 16, 9, 4, 1.

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

Out[203]:
```
array([[ 1,  1,  1,  1],
       [ 8,  4,  2,  1],
       [27,  9,  3,  1],
       [64, 16,  4,  1]])
```

Matriz identidad multiplicada por un valor

In [204…
```
i
```

Out[204]:
```
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.]])
```

In [205…
```
j = 5 * i
j
```

Out[205]:
```
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.]])
```

Métodos `zeros()` / `ones()`

Matriz de todo 1

In [206…
```
k = np.ones((3, 4)) # Matriz de 3 filas por 4 columnas --> valores 1
k
```

Out[206]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

In [207…
```
# se puede añadir un tercer parámetro que es el numero de arrays:

np.ones((2, 3, 2))
```

Out[207]:
```
array([[[1., 1.],
        [1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.],
        [1., 1.]]])
```

Matriz de todo 0

In [208…
```
l = np.zeros((3, 4)) # Matriz de 0s --> 3 filas por 4 columnas
l
```

Out[208]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

In [209…
```
l2 = np.zeros((6, 2))
l2
```

Out[209]:
```
array([[0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.]])
```

In [210…
```
np.zeros((2, 3, 2))  # Idem: a ones()
```

Out[210]:
```
array([[[0., 0.],
        [0., 0.],
        [0., 0.]],

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

Metodos `random()` / `indices()`

`random()` genera valores pseudoaletarios entre 0 y 1:

In [211…
```
from numpy.random import default_rng

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

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

Out[211]:
```
array([[0.77395605, 0.43887844, 0.85859792],
       [0.69736803, 0.09417735, 0.97562235]])
```

In [212…
```
default_rng(42).random((2,3,2))  # idem a ones()
```

Out[212]:
```
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:

In [213…
```
# Matriz de 3 filas por 3 columnas:

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

Out[213]:
```
array([[[0, 0, 0],
        [1, 1, 1],
        [2, 2, 2]],

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

Replicas o copias con numpy

In [214…
```
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, pero b sólo escoge valortes de de 0 has 2 no incluido

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

In [215…
```
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.

In [216…
```
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]]

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

In [217…
```
# Poner nombre del archivo, seleccionar el delimitador que en este ejempl

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

Out[217]:
```
array([[0., 0.],
       [1., 1.],
       [2., 4.],
       [3., 9.]])
```

Modificacion de matrices

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

Intercambio de filas por columnas

In [218…
```
m = np.array([[1, 2, 3],
              [4, 5, 6]])
m
```

Out[218]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [219…
```
# Opción 1
m.transpose()
```

Out[219]:
```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

In [220…
```
# Opción 2
m.T
```

Out[220]:
```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

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

In [221…
```
n = np.array([[1, 2, 3],
              [4, 5, 6]])
n
```

Out[221]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [222…
```
# ALL --> ¿Todos los elementos son mayores de 0? --> True/False
np.all(n>0)
```

Out[222]:
```
True
```

In [223…
```
np.all(n>2)
```

Out[223]: False

In [224…
```
# ANY --> ¿Algún elemento son mayores de 2?
np.any(n>2)
```

Out[224]: True

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

In [225…
```
x = np.array([[1., 2.], [np.nan, 3.], [np.nan, np.nan]])
x
```

Out[225]:
```
array([[ 1.,  2.],
       [nan,  3.],
       [nan, nan]])
```

In [226…
```
# isnan nos muestra el array resultante con salida de True si es un valor

np.isnan(x)
```

Out[226]:
```
array([[False, False],
       [ True, False],
       [ True,  True]])
```

Función `ravel()`

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

In [227…
```
p = np.array([[1, 2, 3],
              [4, 5, 6]])
p
```

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

In [228…
```
# np.ravel(matriz a modificar)
np.ravel(p)
```

Out[228]:
```
array([1, 2, 3, 4, 5, 6])
```

In [229…
```
p1 = np.array([[1, 2, 3],
               [4, 5, 6],
               [7, 8, 9]])
p1
```

Out[229]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [230…
```
np.ravel(p1)
```

Out[230]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

`flatten()`

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

In [231…
```
matriz = np.array([[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9]])
matriz
```

Out[231]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [232…
```
# nombre matriz + flatten()
matriz.flatten()
```

Out[232]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [233…
```
m = matriz.flatten()
```

In [234…
```
m.shape
```

Out[234]:
```
(9,)
```

`roll()`

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

In [235…
```
b = np.array([[1, 2, 3, 4],
              [5, 6, 7, 8],
              [9, 10, 11, 12]])
b
```

Out[235]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [236…
```
# Desplazamiento= 1 y eje horizontal
np.roll(b, 1, axis=0)
```

Out[236]:
```
array([[ 9, 10, 11, 12],
       [ 1,  2,  3,  4],
       [ 5,  6,  7,  8]])
```

In [237…
```
# Desplazamiento = 1 y eje vertical
np.roll(b, 1, axis=1)
```

Out[237]:
```
array([[ 4,  1,  2,  3],
       [ 8,  5,  6,  7],
       [12,  9, 10, 11]])
```

In [238…
```
# Desplazamiento= -1 y eje horizontal
np.roll(b, -1, axis=0)
```

Out[238]:
```
array([[ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [ 1,  2,  3,  4]])
```

In [239...
```
# Desplazamiento = -1 y eje vertical
np.roll(b, -1, axis=1)
```

Out[239]:
```
array([[ 2,  3,  4,  1],
       [ 6,  7,  8,  5],
       [10, 11, 12,  9]])
```

`logspace()`

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

In [240...
```
# np.logspace(10^inicio, 10^fin, divisiones(elementos))
# como en linspace se incluye los extremos (inicios-->fin)
c = np.logspace(0, 1, 3)
c
```

Out[240]:
```
array([ 1.        ,  3.16227766, 10.        ])
```

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

$$ 10^0 = 1\,\,\,\,y\,\,\,\, 10^1=1\,\,\,\,3 divisiones(elementos) $$

Slicing

Acceso a un elemento de un array:

In [241...
```
matriz = np.array([
    [10, 20],
    [30, 40]
])
matriz
```

Out[241]:
```
array([[10, 20],
       [30, 40]])
```

In [242...
```
matriz[0][0] # fila 0 columna 0
```

Out[242]:
```
10
```

In [243...
```
matriz[0][1] # fila 0 columna 1
```

Out[243]:
```
20
```

Otro ejemplo...

In [244...
```
q = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])
q
```

Out[244]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [245…
```
# Opción 1
q[2][1] # --> fila 2 y columna 1 (listas 0, 1, 2)
```
Out[245]:
```
8
```
In [246…
```
q[0][2]
```
Out[246]:
```
3
```
In [247…
```
# Opción 2
q[2, 1]
```
Out[247]:
```
8
```
In [248…
```
# dos primeras filas (: --> todas)
q[:2]
```
Out[248]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```
In [249…
```
q[2:]
```
Out[249]:
```
array([[7, 8, 9]])
```
In [250…
```
# Filtrar por columnas
q[:,[0]]
```
Out[250]:
```
array([[1],
       [4],
       [7]])
```
In [251…
```
# Filtrar por columnas
q[:,[0,1]]
```
Out[251]:
```
array([[1, 2],
       [4, 5],
       [7, 8]])
```
In [252…
```
# 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]
```
Out[252]:
```
array([1, 3, 5, 7, 9])
```

Array de 5 x 5

In [253…
```
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
```

Out[253]:
```
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]])
```

Imprimir desde la 3ª columna hasta el final

In [254…
```
print(a) # mostrar la información de la matriz
```

```
[[ 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]]
```

In [255…
```
# 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 incluída)
a[:, :2]
```

Out[255]:
```
array([[ 1,  2],
       [ 6,  7],
       [11, 12],
       [16, 17],
       [21, 22]])
```

In [256…
```
# todas las columnas de las 2 primeras filas
a[:2]
```

Out[256]:
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
```

In [257…
```
a[:2, :]
```

Out[257]:
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
```

In [258…
```
a[:, 1:2]
```

Out[258]:
```
array([[ 2],
       [ 7],
       [12],
       [17],
       [22]])
```

In [ ]:
```
# NOTA: esta parte será importante para el tema de visualización de los d
# ver el tema de df.loc o df.iloc
```

Type...

In [259…
```
type(a[:,2:])
```

Out[259]:
```
numpy.ndarray
```

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

In [260…
```
a
```

Out[260]:
```
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]])
```

In [261…
```
# Opción 1

a[:, :2]
```

Out[261]:
```
array([[ 1,  2],
       [ 6,  7],
       [11, 12],
       [16, 17],
       [21, 22]])
```

In [262…
```
# Opción 2

a[:, 0:2]
```

Out[262]:
```
array([[ 1,  2],
       [ 6,  7],
       [11, 12],
       [16, 17],
       [21, 22]])
```

Imprimo las pares

In [263…
```
a
```

Out[263]:
```
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]])
```

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

In [264…
```
a[:, 1::2]
```

Out[264]:
```
array([[ 2,  4],
       [ 7,  9],
       [12, 14],
       [17, 19],
       [22, 24]])
```

In [265…
```
a[:, 1::3]
```

Out[265]:
```
array([[ 2,  5],
       [ 7, 10],
       [12, 15],
       [17, 20],
       [22, 25]])
```

Imprimir las impares

In [266…
```
a
```

Out[266]:
```
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]])
```

In [267…
```
a[:, 0::2]
```

Out[267]:
```
array([[ 1,  3,  5],
       [ 6,  8, 10],
       [11, 13, 15],
       [16, 18, 20],
       [21, 23, 25]])
```

In [268…
```
a[:, 0:2:2]
```

Out[268]:
```
array([[ 1],
       [ 6],
       [11],
       [16],
       [21]])
```

In [269…
```
a[:, 0:3:2]
```

Out[269]:
```
array([[ 1,  3],
       [ 6,  8],
       [11, 13],
       [16, 18],
       [21, 23]])
```

Comparacion entre Arrays

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

In [270…
```
s = np.array([
    [1, 2, 3],
    [4, 5, 6]
])
s
```

Out[270]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [271…
```
t = np.array([
    [100, 200, 3],
    [400, 5, 6]
])
t
```

Out[271]:
```
array([[100, 200,   3],
       [400,   5,   6]])
```

Los comparo

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

In [272…
```
np.where(s==t, "True", "False")
```

Out[272]:
```
array([['False', 'False', 'True'],
       ['False', 'True', 'True']], dtype='<U5')
```

In [273…
```
np.where(s==t, "Si", "No")
```

Out[273]:
```
array([['No', 'No', 'Si'],
       ['No', 'Si', 'Si']], dtype='<U2')
```

In [274…
```
np.where(s==t, 1, 0)
```

Out[274]:
```
array([[0, 0, 1],
       [0, 1, 1]])
```

Concatenación de arrays

Crear los arrays

In [275…
```
y = np.array([
    [1, 2],
    [3, 4]
])
y
```

Out[275]:
```
array([[1, 2],
       [3, 4]])
```

In [276…
```
z = np.array([
    [5, 6]
])
```

Concatenación por filas

In [277…
```
np.concatenate((y,z), axis=0)
```

Out[277]:
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

Concatenación por colunmas

In [278…
```
z1 = z.transpose()
z1
```

Out[278]:
```
array([[5],
       [6]])
```

In [279…
```
np.concatenate((y,z1), axis=1)
```

Out[279]:
```
array([[1, 2, 5],
       [3, 4, 6]])
```

Operaciones

In [ ]:
```
# Potencias
```

In [280…
```
r = np.array([1, 2, 3, 4])
r
```

Out[280]:
```
array([1, 2, 3, 4])
```

In [ ]:
```
# Método 1
```

In [281…
```
r**2 # 1^1, 2^2, 3^3, 4^4
```

Out[281]:
```
array([ 1,  4,  9, 16])
```

In [ ]:
```
# Método 2
```

In [282…
```
pow(r, 2)
```

Out[282]:
```
array([ 1,  4,  9, 16])
```

Producto escalar y producto vectorial de 2 vectores

In [283…
```
w = np.array([1, 2, 3])
w
```

Out[283]:
```
array([1, 2, 3])
```

In [284…
```
x = np.array([2, 5, -4])
x
```

Out[284]:
```
array([ 2,  5, -4])
```

Producto escalar:

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

In [285…
```
# np.dot(matriz1, matriz2)
np.dot(w,x)
```

Out[285]:
```
0
```

Producto Vectorial:

In [ ]:
```
## 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)
```

In [286…
```
np.cross(w, x)
```

Out[286]:
```
array([-23,  10,   1])
```

Matriz con "matrix"

In [287…
```
# 4 filas 4 columnas
u = np.matrix([
    [4, -3, 11, 1],
    [5, 9, 7, 2],
    [2, 3, 4, 1],
    [5, 3, -5, -9]
])
u
```

Out[287]:
```
matrix([[ 4, -3, 11,  1],
        [ 5,  9,  7,  2],
        [ 2,  3,  4,  1],
        [ 5,  3, -5, -9]])
```

In [288…
```
# 1 fila y 4 columnas
v = np.matrix([4, 9, 1, 3])
v
```

Out[288]:
```
matrix([[4, 9, 1, 3]])
```

Suma

In [289…
```
u + v
```

Out[289]:
```
matrix([[ 8,  6, 12,  4],
        [ 9, 18,  8,  5],
        [ 6, 12,  5,  4],
        [ 9, 12, -4, -6]])
```

Resta

In [290…
```
u - v
```

Out[290]:
```
matrix([[  0, -12,  10,  -2],
        [  1,   0,   6,  -1],
        [ -2,  -6,   3,  -2],
        [  1,  -6,  -6, -12]])
```

Producto

In [291…
```
u * v
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[291], line 1
----> 1 u * v

File ~/.local/lib/python3.10/site-packages/numpy/matrixlib/defmatrix.py:219, in matrix.__mul__(self, other)
    216 def __mul__(self, other):
    217     if isinstance(other, (N.ndarray, list, tuple)) :
    218         # This promotes 1-D vectors to row vectors
--> 219         return N.dot(self, asmatrix(other))
    220     if isscalar(other) or not hasattr(other, '__rmul__') :
    221         return N.dot(self, other)

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

In [ ]:
```
# ValueError --> es necesario realizar la transpuesta para este caso, ya
```

Opción 1:

In [292…
```
u*v.transpose()
```

Out[292]:
```
matrix([[  3],
        [114],
        [ 42],
        [ 15]])
```

Opción 2:

In [293…
```
u*v.T
```

Out[293]:
```
matrix([[  3],
        [114],
        [ 42],
        [ 15]])
```

Opción 3:

In [294…
```
np.dot(u, v.T)
```

Out[294]:
```
matrix([[  3],
        [114],
        [ 42],
        [ 15]])
```

Traza de una matriz

(suma de los elementos de la diagonal principal)

In [295…
```
u -v
```

Out[295]:
```
matrix([[  0, -12,  10,  -2],
        [  1,   0,   6,  -1],
        [ -2,  -6,   3,  -2],
        [  1,  -6,  -6, -12]])
```

In [296…
```
type(u-v)
```

Out[296]:
```
numpy.matrix
```

In [297…
```
np.trace(u-v) # 0 + 0 + 3 + (-12) = -9 (suma de los elementos de la diago
```

Out[297]:
```
-9
```

Matematical functions

Trigonometric functions

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

Rounding

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

Functions	Description
trunc(x, /[, out, where, casting, order, ...])	Return the truncated value of the input, element-wise.

Sums, products, differences

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

Arithmetic operations

Functions	Description
add(x1, x2, /[, out, where, casting, order, ...])	Add arguments element-wise.
reciprocal(x, /[, out, where, casting, ...])	Return the reciprocal of the argument, element-wise.
positive(x, /[, out, where, casting, order, ...])	Numerical positive, element-wise.
negative(x, /[, out, where, casting, order, ...])	Numerical negative, element-wise.
multiply(x1, x2, /[, out, where, casting, ...])	Multiply arguments element-wise.
divide(x1, x2, /[, out, where, casting, ...])	Divide arguments element-wise.
power(x1, x2, /[, out, where, casting, ...])	First array elements raised to powers from second array, element-wise.
pow(x1, x2, /[, out, where, casting, order, ...])	First array elements raised to powers from second array, element-wise.
subtract(x1, x2, /[, out, where, casting, ...])	Subtract arguments, element-wise.

Functions	Description
true_divide(x1, x2, /[, out, where, ...])	Divide arguments element-wise.
floor_divide(x1, x2, /[, out, where, ...])	Return the largest integer smaller or equal to the division of the inputs.
float_power(x1, x2, /[, out, where, ...])	First array elements raised to powers from second array, element-wise.
fmod(x1, x2, /[, out, where, casting, ...])	Returns the element-wise remainder of division.
mod(x1, x2, /[, out, where, casting, order, ...])	Returns the element-wise remainder of division.

Extrema finding

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

Creado por:

Isabel Maniega