4.1 Análisis de datos con Pandas y NumPy

May 13, 2025

Creado por:

Isabel Maniega

1 4.1.1, 4.1.2 y 4.1.4: Pandas

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
```

[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 apartir 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	2.030408	-0.206107	0.400008	0.048040
2013-01-02	1.351692	-0.026938	-1.942320	-0.571743
2013-01-03	0.231676	-1.104539	0.242625	0.839671
2013-01-04	-1.166002	-0.604175	0.603957	-0.743878
2013-01-05	-0.398931	-2.258892	0.528703	0.477096
2013-01-06	0.191633	0.690677	0.288011	-0.998729

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

df.dtypes
```

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

2.1 Vista de los datos

[8]:
```
# Muestra las primeras filas del dataframe, por defecto las 5 primeras
df.head()
```

[8]:

	A	B	C	D
2013-01-01	2.030408	-0.206107	0.400008	0.048040
2013-01-02	1.351692	-0.026938	-1.942320	-0.571743
2013-01-03	0.231676	-1.104539	0.242625	0.839671
2013-01-04	-1.166002	-0.604175	0.603957	-0.743878

```
2013-01-05 -0.398931 -2.258892  0.528703  0.477096
```

[9]:
```python
df.head(2)
```

[9]:
```
                   A         B         C         D
2013-01-01  2.030408 -0.206107  0.400008  0.048040
2013-01-02  1.351692 -0.026938 -1.942320 -0.571743
```

[10]:
```python
# Muestra las últimas filas de un dataframe, por defecto las 5 últimas:

df.tail()
```

[10]:
```
                   A         B         C         D
2013-01-02  1.351692 -0.026938 -1.942320 -0.571743
2013-01-03  0.231676 -1.104539  0.242625  0.839671
2013-01-04 -1.166002 -0.604175  0.603957 -0.743878
2013-01-05 -0.398931 -2.258892  0.528703  0.477096
2013-01-06  0.191633  0.690677  0.288011 -0.998729
```

[11]:
```python
df.tail(2)
```

[11]:
```
                   A         B         C         D
2013-01-05 -0.398931 -2.258892  0.528703  0.477096
2013-01-06  0.191633  0.690677  0.288011 -0.998729
```

[12]:
```python
# Muestra el valor de la primera columna que suele ser un valor único (id), en␣
↪este ejemplo una fecha:

df.index
```

[12]:
```
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')
```

[13]:
```python
# Muestra el nombre de las columnas:

df.columns
```

[13]:
```
Index(['A', 'B', 'C', 'D'], dtype='object')
```

[14]:
```python
# Podemos convertir un dataframe en una matriz de numpy con:

df.to_numpy()
```

[14]:
```
array([[ 2.03040798, -0.20610733,  0.40000832,  0.04803982],
       [ 1.3516918 , -0.02693779, -1.94231971, -0.5717427 ],
       [ 0.23167575, -1.10453886,  0.24262538,  0.83967059],
       [-1.16600242, -0.60417493,  0.60395658, -0.74387757],
```

```
       [-0.39893146, -2.25889197,  0.52870311,  0.47709551],
       [ 0.19163316,  0.69067682,  0.2880109 , -0.99872857]])
```

[15]:
```python
# Para obtener los estadísticos más representativos usamos:

df.describe()
```

[15]:
```
              A         B         C         D
count  6.000000  6.000000  6.000000  6.000000
mean   0.373412 -0.584996  0.020164 -0.158257
std    1.159492  1.015337  0.971214  0.729713
min   -1.166002 -2.258892 -1.942320 -0.998729
25%   -0.251290 -0.979448  0.253972 -0.700844
50%    0.211654 -0.405141  0.344010 -0.261851
75%    1.071688 -0.071730  0.496529  0.369832
max    2.030408  0.690677  0.603957  0.839671
```

[16]:
```python
# Podemos dar la vuelta a la tabla y poner lo que esta en filas en columnas y␣
 ↪viceversa:

df.T
```

[16]:
```
   2013-01-01  2013-01-02  2013-01-03  2013-01-04  2013-01-05  2013-01-06
A    2.030408    1.351692    0.231676   -1.166002   -0.398931    0.191633
B   -0.206107   -0.026938   -1.104539   -0.604175   -2.258892    0.690677
C    0.400008   -1.942320    0.242625    0.603957    0.528703    0.288011
D    0.048040   -0.571743    0.839671   -0.743878    0.477096   -0.998729
```

[17]:
```python
# Colocar los valores según el indice:

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

[17]:
```
                   D         C         B         A
2013-01-01  0.048040  0.400008 -0.206107  2.030408
2013-01-02 -0.571743 -1.942320 -0.026938  1.351692
2013-01-03  0.839671  0.242625 -1.104539  0.231676
2013-01-04 -0.743878  0.603957 -0.604175 -1.166002
2013-01-05  0.477096  0.528703 -2.258892 -0.398931
2013-01-06 -0.998729  0.288011  0.690677  0.191633
```

[18]:
```python
# Ordenar los datos según una columna:

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

[18]:
```
                   A         B         C         D
2013-01-05 -0.398931 -2.258892  0.528703  0.477096
2013-01-03  0.231676 -1.104539  0.242625  0.839671
2013-01-04 -1.166002 -0.604175  0.603957 -0.743878
```

```
2013-01-01  2.030408 -0.206107  0.400008  0.048040
2013-01-02  1.351692 -0.026938 -1.942320 -0.571743
2013-01-06  0.191633  0.690677  0.288011 -0.998729
```

2.2 Seleccion

2.3 GetItem()

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

[19]:
```
df['A']
```

[19]:
```
2013-01-01    2.030408
2013-01-02    1.351692
2013-01-03    0.231676
2013-01-04   -1.166002
2013-01-05   -0.398931
2013-01-06    0.191633
Freq: D, Name: A, dtype: float64
```

[20]:
```
df.A
```

[20]:
```
2013-01-01    2.030408
2013-01-02    1.351692
2013-01-03    0.231676
2013-01-04   -1.166002
2013-01-05   -0.398931
2013-01-06    0.191633
Freq: D, Name: A, dtype: float64
```

[21]:
```
df[['A']]
```

[21]:
```
                   A
2013-01-01  2.030408
2013-01-02  1.351692
2013-01-03  0.231676
2013-01-04 -1.166002
2013-01-05 -0.398931
2013-01-06  0.191633
```

Selección de filas mediante slicing(:)

[22]:
```
df[0:2]
```

[22]:
```
                   A         B         C         D
2013-01-01  2.030408 -0.206107  0.400008  0.048040
2013-01-02  1.351692 -0.026938 -1.942320 -0.571743
```

[23]:
```
df["20130103":"20130105"]
```

[23]:
```
                   A         B         C         D
2013-01-03  0.231676 -1.104539  0.242625  0.839671
2013-01-04 -1.166002 -0.604175  0.603957 -0.743878
2013-01-05 -0.398931 -2.258892  0.528703  0.477096
```

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

[24]:
```python
# Filas que coinciden con una etiqueta, selección de la primera fila:

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

[24]:
```
A    2.030408
B   -0.206107
C    0.400008
D    0.048040
Name: 2013-01-01 00:00:00, dtype: float64
```

[25]:
```python
# Seleccionar todas las filas de una determinada columna:

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

[25]:
```
                   B         C
2013-01-01 -0.206107  0.400008
2013-01-02 -0.026938 -1.942320
2013-01-03 -1.104539  0.242625
2013-01-04 -0.604175  0.603957
2013-01-05 -2.258892  0.528703
2013-01-06  0.690677  0.288011
```

[26]:
```python
# Seleccionar por filas y columnas:

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

[26]:
```
                   B         C
2013-01-03 -1.104539  0.242625
2013-01-04 -0.604175  0.603957
2013-01-05 -2.258892  0.528703
```

[27]:
```python
# Seleccionar para un valor determinado -0.891699 (20130103, B):

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

[27]:
```
np.float64(-1.1045388557043914)
```

[28]:
```python
df.at[dates[2], 'B']
```

[28]:
```
np.float64(-1.1045388557043914)
```

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

[29]:
```python
# Selección de una fila en posición 3:

df.iloc[3]
```

[29]:
```
A   -1.166002
B   -0.604175
C    0.603957
D   -0.743878
Name: 2013-01-04 00:00:00, dtype: float64
```

[30]:
```python
# Selección de una fila y columna por slicing:

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

[30]:
```
                   B         C
2013-01-04 -0.604175  0.603957
2013-01-05 -2.258892  0.528703
```

[31]:
```python
# Selección por lista de posiciones:
# Filas: 1, 2, 4
# Columnas: 0(A), 2(C)

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

[31]:
```
                   A         C
2013-01-02  1.351692 -1.942320
2013-01-03  0.231676  0.242625
2013-01-05 -0.398931  0.528703
```

[32]:
```python
# Selección por filas o columnas:

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

[32]:
```
                   A         B         C         D
2013-01-02  1.351692 -0.026938 -1.942320 -0.571743
2013-01-03  0.231676 -1.104539  0.242625  0.839671
```

[33]:
```python
df.iloc[:, 1:3]
```

[33]:
```
                   B         C
2013-01-01 -0.206107  0.400008
2013-01-02 -0.026938 -1.942320
2013-01-03 -1.104539  0.242625
2013-01-04 -0.604175  0.603957
2013-01-05 -2.258892  0.528703
2013-01-06  0.690677  0.288011
```

[34]:
```python
# Seleccionar un valor concreto por posición (2013-01-03, 'B'):

df.iloc[2, 1]
```

[34]: np.float64(-1.1045388557043914)

[35]:
```python
df.iat[2, 1]
```

[35]: np.float64(-1.1045388557043914)

2.4 Boolean indexing

[36]:
```python
# Selección por comparativa:

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

[36]:

	A	B	C	D
2013-01-01	2.030408	-0.206107	0.400008	0.048040
2013-01-02	1.351692	-0.026938	-1.942320	-0.571743
2013-01-03	0.231676	-1.104539	0.242625	0.839671

[37]:
```python
df[df > 0]
```

[37]:

	A	B	C	D
2013-01-01	2.030408	NaN	0.400008	0.048040
2013-01-02	1.351692	NaN	NaN	NaN
2013-01-03	0.231676	NaN	0.242625	0.839671
2013-01-04	NaN	NaN	0.603957	NaN
2013-01-05	NaN	NaN	0.528703	0.477096
2013-01-06	0.191633	0.690677	0.288011	NaN

Método isin()

[38]:
```python
# Selección según una coincidencia (filtrado):

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

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

[38]:

	E
0	one
1	one
4	four

2.5 Setting (Modificacion del dataframe)

[39]:
```
# Añadir Valores nuevo

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

[39]:
```
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
```

[40]:
```
df['E'] = serie
df
```

[40]:
```
                   A         B         C         D  E
2013-01-01  2.030408 -0.206107  0.400008  0.048040  1
2013-01-02  1.351692 -0.026938 -1.942320 -0.571743  2
2013-01-03  0.231676 -1.104539  0.242625  0.839671  3
2013-01-04 -1.166002 -0.604175  0.603957 -0.743878  4
2013-01-05 -0.398931 -2.258892  0.528703  0.477096  5
2013-01-06  0.191633  0.690677  0.288011 -0.998729  6
```

[41]:
```
# Modificar valor por etiqueta
# Se modifica el primer valor de df por 0 en la columna A:

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

[41]:
```
                   A         B         C         D  E
2013-01-01  0.000000 -0.206107  0.400008  0.048040  1
2013-01-02  1.351692 -0.026938 -1.942320 -0.571743  2
2013-01-03  0.231676 -1.104539  0.242625  0.839671  3
2013-01-04 -1.166002 -0.604175  0.603957 -0.743878  4
2013-01-05 -0.398931 -2.258892  0.528703  0.477096  5
2013-01-06  0.191633  0.690677  0.288011 -0.998729  6
```

[42]:
```
# Modificación de valor por posición
# Se modifica el primer valor de la columna B:

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

[42]:
```
                   A         B         C         D  E
2013-01-01  0.000000  0.000000  0.400008  0.048040  1
2013-01-02  1.351692 -0.026938 -1.942320 -0.571743  2
2013-01-03  0.231676 -1.104539  0.242625  0.839671  3
2013-01-04 -1.166002 -0.604175  0.603957 -0.743878  4
2013-01-05 -0.398931 -2.258892  0.528703  0.477096  5
2013-01-06  0.191633  0.690677  0.288011 -0.998729  6
```

[43]:
```
# Modificación asignada por Numpy usando array:

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

[43]:
```
                   A         B         C    D  E
2013-01-01  0.000000  0.000000  0.400008  5.0  1
2013-01-02  1.351692 -0.026938 -1.942320  5.0  2
2013-01-03  0.231676 -1.104539  0.242625  5.0  3
2013-01-04 -1.166002 -0.604175  0.603957  5.0  4
2013-01-05 -0.398931 -2.258892  0.528703  5.0  5
2013-01-06  0.191633  0.690677  0.288011  5.0  6
```

[44]:
```
# Modificar según una condición (where):

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

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

[44]:
```
                   A         B         C    D  E
2013-01-01  0.000000  0.000000 -0.400008 -5.0 -1
2013-01-02 -1.351692 -0.026938 -1.942320 -5.0 -2
2013-01-03 -0.231676 -1.104539 -0.242625 -5.0 -3
2013-01-04 -1.166002 -0.604175 -0.603957 -5.0 -4
2013-01-05 -0.398931 -2.258892 -0.528703 -5.0 -5
2013-01-06 -0.191633 -0.690677 -0.288011 -5.0 -6
```

2.6 Missing values

[45]:
```
# 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  0.400008  NaN  1.0
2013-01-02  1.351692 -0.026938 -1.942320  5.0  2.0
2013-01-03  0.231676 -1.104539  0.242625  5.0  NaN
2013-01-04 -1.166002 -0.604175  0.603957  5.0  NaN
```

[46]:
```
# Eliminamos los valores nulos con la función dropna(): eliminando cualquier␣
↪fila que contenga valores nulos

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

[46]:
```
                   A         B        C    D    E
2013-01-02  1.351692 -0.026938 -1.94232  5.0  2.0
```

[47]:
```
# Rellenar valores nulos:

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

[47]:
```
                   A         B         C    D    E
2013-01-01  0.000000  0.000000  0.400008  5.0  1.0
2013-01-02  1.351692 -0.026938 -1.942320  5.0  2.0
2013-01-03  0.231676 -1.104539  0.242625  5.0  5.0
2013-01-04 -1.166002 -0.604175  0.603957  5.0  5.0
```

[48]:
```
# isna() nos muestra si en el df hay valores nulo o no, sustituyendo por un␣
↪booleano (True / False)

pd.isna(df1)
```

[48]:
```
                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
```

2.7 Operaciones

En estos casos no tiene en cuenta los valores nulos.

[49]:
```
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()
```

[49]:
```
   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
```

2.7.1 Tendencia Central

Media

Como calcular la media de las distintas notas:

[50]:
```python
media_1 = df["notas_1"].mean()
media_1
```

[50]: np.float64(15.5)

[51]:
```python
media_2 = df["notas_2"].mean()
media_2
```

[51]: np.float64(16.8)

[52]:
```python
media_3 = df["notas_3"].mean()
media_3
```

[52]: np.float64(17.6)

Mediana

Como calcular la mediana de las distintas notas:

[53]:
```python
mediana_1 = df["notas_1"].median()
mediana_1
```

[53]: np.float64(15.0)

[54]:
```python
mediana_2 = df["notas_2"].median()
mediana_2
```

[54]: np.float64(16.0)

[55]:
```python
mediana_3 = df["notas_3"].median()
mediana_3
```

[55]: np.float64(16.0)

Moda

Como calcular la moda de las distintas notas:

[56]:
```python
moda_1 = df["notas_1"].mode()
moda_1
```

[56]:
```
0    14
1    15
Name: notas_1, dtype: int64
```

[57]:
```python
moda_2 = df["notas_2"].mode()
moda_2
```

[57]:
```
0    15
1    16
Name: notas_2, dtype: int64
```

[58]:
```python
moda_3 = df["notas_3"].mode()
moda_3
```

[58]:
```
0    15
Name: notas_3, dtype: int64
```

[59]:
```python
df.notas_3.value_counts()
```

[59]:
```
notas_3
15    3
22    2
16    2
17    1
14    1
24    1
Name: count, dtype: int64
```

Resultados Nota_1:

[60]:
```python
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:

[61]:
```python
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:

[62]:
```python
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:

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

[63]:
```
var_1 = df["notas_1"].var()
var_1
```

[63]:
```
np.float64(14.5)
```

[64]:
```
var_2 = df["notas_2"].var()
var_2
```

[64]:
```
np.float64(8.399999999999999)
```

[65]:
```
var_3 = df["notas_3"].var()
var_3
```

[65]:
```
np.float64(13.155555555555557)
```

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

[66]:
```
var_1 = df["notas_1"].var(ddof=0)
var_1
```

[66]:
```
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.

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

[67]:
```
std_1 = df["notas_1"].std()
std_1
```

[67]:
```
np.float64(3.8078865529319543)
```

[68]:
```
std_2 = df["notas_2"].std()
std_2
```

[68]:
```
np.float64(2.8982753492378874)
```

[69]:
```
std_3 = df["notas_3"].std()
std_3
```

[69]:
```
np.float64(3.6270588023294517)
```

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

[70]:
```
std_1 = df["notas_1"].std(ddof=0)
std_1
```

[70]:
```
np.float64(3.6124783736376886)
```

Máximo y mínimo

[71]:
```
max_1 = df["notas_1"].max()
min_1 = df["notas_1"].min()
print(max_1, min_1)
```

```
25 10
```

[72]:
```
max_2 = df["notas_2"].max()
min_2 = df["notas_2"].min()
print(max_2, min_2)
```

```
22 13
```

[73]:
```
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

[74]:
```
df.describe()
```

[74]:

	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

[75]:
```
iris = pd.read_csv('./files/Iris.csv')
iris = iris.drop(['Id'], axis=1)
iris_setosa = iris[0:50]
iris_setosa
```

[75]:

	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

[76]:
```
iris_virginica = iris[100:]
iris_virginica
```

[76]:

	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

[77]:
```
iris_versicolor = pd.read_json('./files/iris_versicolor.json')
iris_versicolor
```

[77]:

	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()

[78]:
```
# 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
```

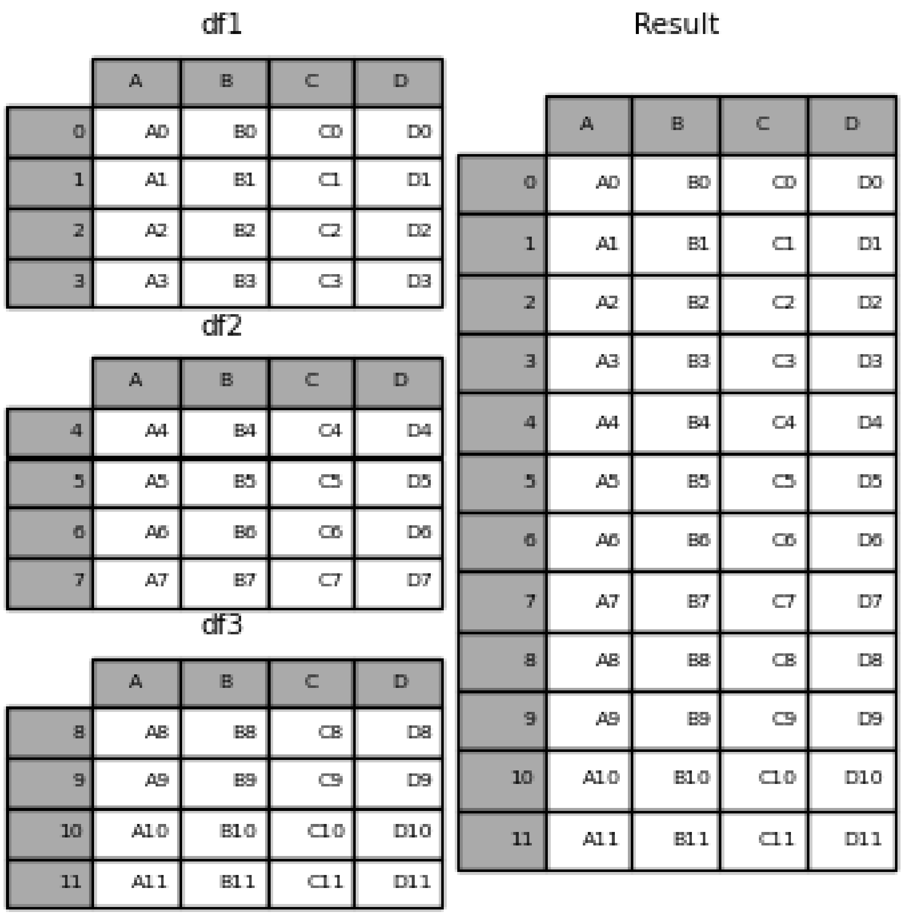
[78]:

	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
..	…	…	…	…	…
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]
```

[79]:
```python
display.Image('./images/merging_concat_basic.png')
```

[79]:



[80]:
```python
iris = pd.read_csv('./files/Iris.csv')
iris_medidas = iris.iloc[:, 0:4]
iris_medidas
```

[80]:
```
      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]
```

[81]:
```
iris_especies = iris[['Species']]
iris_especies
```

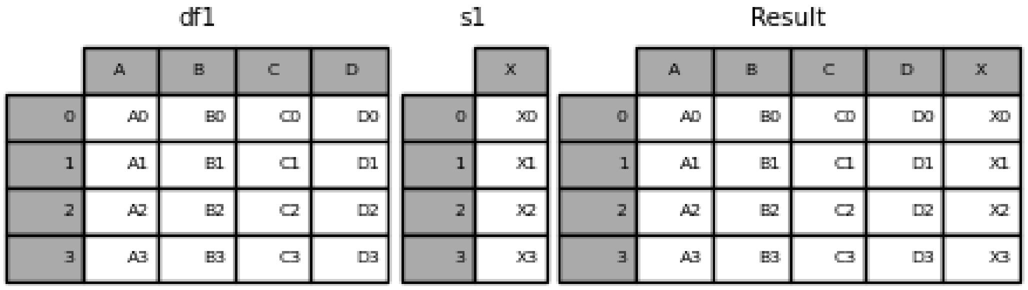
[81]:
```
             Species
0        Iris-setosa
1        Iris-setosa
2        Iris-setosa
3        Iris-setosa
4        Iris-setosa
..               …
145   Iris-virginica
146   Iris-virginica
147   Iris-virginica
148   Iris-virginica
149   Iris-virginica

[150 rows x 1 columns]
```

[82]:
```
# 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
```

[82]:
```
      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]
```

[83]:
```
display.Image('./images/merging_concat_mixed.png')
```

[83]:



3.3 merge()

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

[84]:
```
new_species = iris.loc[:, ['Id', 'Species']]
new_species
```

[84]:
```
      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]
```

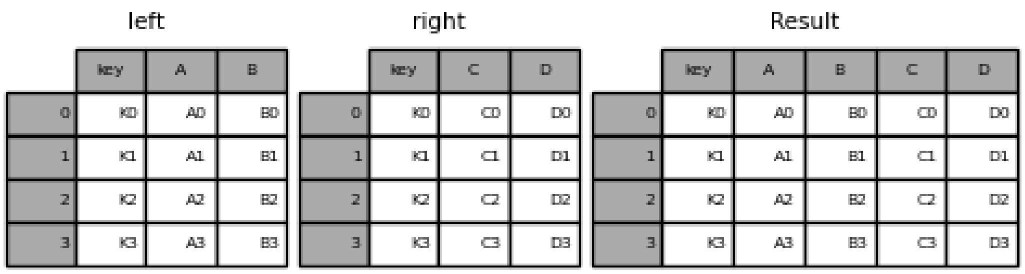
[85]:
```
new_setosa = pd.merge(iris_medidas, new_species, on='Id')
new_setosa
```

[85]:
```
      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]
```

[86]:
```python
display.Image('./images/merging_merge_on_key.png')
```

[86]:



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

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

[88]:
```
      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]
```

[89]:
```python
new_setosa = pd.merge(iris_medidas, new_species, how='inner', 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]:
```python
new_setosa = pd.merge(iris_medidas, new_species, how='outer', 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
```

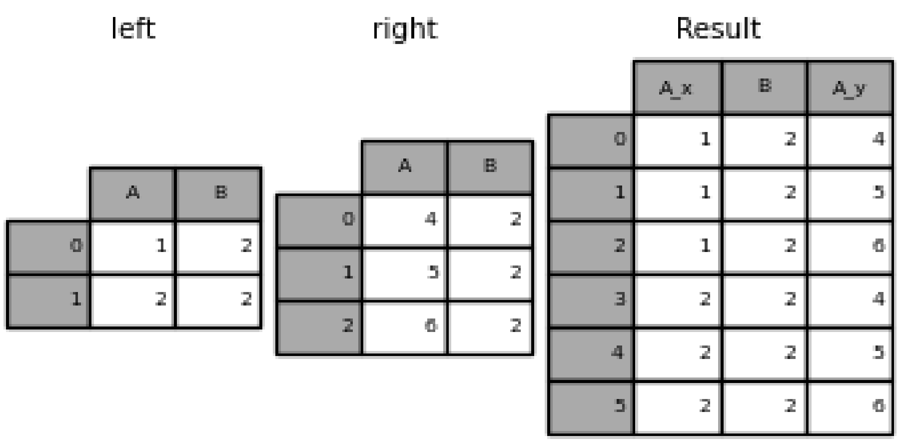
```
4      5             5.0           3.6            1.4     Iris-setosa
..   ...             ...           ...            ...             ...
145  146             6.7           3.0            5.2  Iris-virginica
146  147             6.3           2.5            5.0  Iris-virginica
147  148             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]:
```python
new_setosa = pd.merge(iris_medidas, new_species, how='cross')
new_setosa
```

[91]:
```
       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]
```

[92]:
```python
# Si no existe la clave la duplica en el caso how=cross:

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

[92]:



3.4 join()

[93]:
```python
iris_medidas
```

[93]:
```
      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]
```

[94]:
```python
iris_especies
```

[94]:
```
            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]
```

[95]:
```python
iris_2 = iris_medidas.join(iris_especies)
iris_2
```

[95]:
```
      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

[96]:
```python
iris
```

[96]:
```
      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]
```

[97]:
```python
iris_sepal = iris.groupby('Species')[["SepalLengthCm", "SepalWidthCm"]].mean()
iris_sepal
```

[97]:
```
                 SepalLengthCm  SepalWidthCm
Species
Iris-setosa              5.006         3.418
Iris-versicolor          5.936         2.770
Iris-virginica           6.588         2.974
```

[98]:
```
iris_petal = iris.groupby('Species')[["PetalLengthCm", "PetalWidthCm"]].mean()
iris_petal
```

[98]:
```
                 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()

[99]:
```
# Ponemos como columna de index la de especies, asi aplicaremos los datos segun␣
↪de que
# especie sean:

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

[99]:
```
                     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]
```

[100]:
```python
stack_iris = reiris.stack(future_stack=True)
stack_iris
```

[100]:
```
     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`.

[101]:
```python
unstack_iris = reiris.unstack()
unstack_iris
```

[101]:
```
                 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()

[102]:
```
# 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
```

[102]:
```
                    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
```

[103]:
```
# Agrupación de datos de especie por media:

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

[103]:
```
                   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
```

[104]:
```
# el parametro values nos ayuda a seleccionar las columnas concretas:


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

[104]:
```
                 PetalLengthCm
Species
Iris-setosa              1.464
Iris-versicolor          4.260
Iris-virginica           5.552
```

3.9 Time Series

[105]:
```
# 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
```

[105]:
```
2024-06-01   -0.357801
2024-06-02   -0.004643
2024-06-03   -0.389238
2024-06-04    1.007457
2024-06-05   -0.366685
2024-06-06    1.293329
2024-06-07   -0.154573
2024-06-08    1.671726
2024-06-09   -0.580897
2024-06-10   -1.269996
2024-06-11    1.749514
2024-06-12    1.342715
2024-06-13   -0.610414
2024-06-14    0.422721
2024-06-15   -0.530744
Freq: D, dtype: float64
```

3.10 tz_localize()

[106]:
```
# añadimos la hora al dataframe creado:

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

[106]:
```
2024-06-01 00:00:00+00:00   -0.357801
2024-06-02 00:00:00+00:00   -0.004643
2024-06-03 00:00:00+00:00   -0.389238
2024-06-04 00:00:00+00:00    1.007457
2024-06-05 00:00:00+00:00   -0.366685
2024-06-06 00:00:00+00:00    1.293329
2024-06-07 00:00:00+00:00   -0.154573
2024-06-08 00:00:00+00:00    1.671726
2024-06-09 00:00:00+00:00   -0.580897
2024-06-10 00:00:00+00:00   -1.269996
2024-06-11 00:00:00+00:00    1.749514
2024-06-12 00:00:00+00:00    1.342715
2024-06-13 00:00:00+00:00   -0.610414
2024-06-14 00:00:00+00:00    0.422721
```

```
2024-06-15 00:00:00+00:00   -0.530744
Freq: D, dtype: float64
```

3.11 tz_convert()

[107]:
```
# Ponemos la franja horaria a la cual nos encontramos:

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

[107]:
```
2024-06-01 02:00:00+02:00   -0.357801
2024-06-02 02:00:00+02:00   -0.004643
2024-06-03 02:00:00+02:00   -0.389238
2024-06-04 02:00:00+02:00    1.007457
2024-06-05 02:00:00+02:00   -0.366685
2024-06-06 02:00:00+02:00    1.293329
2024-06-07 02:00:00+02:00   -0.154573
2024-06-08 02:00:00+02:00    1.671726
2024-06-09 02:00:00+02:00   -0.580897
2024-06-10 02:00:00+02:00   -1.269996
2024-06-11 02:00:00+02:00    1.749514
2024-06-12 02:00:00+02:00    1.342715
2024-06-13 02:00:00+02:00   -0.610414
2024-06-14 02:00:00+02:00    0.422721
2024-06-15 02:00:00+02:00   -0.530744
Freq: D, dtype: float64
```

3.12 offsets.BusinessDay()

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

[108]:
```
rng
```

[108]:
```
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')
```

[109]:
```
# se añade 5 como número de días a representar:
rng = rng + pd.offsets.BusinessDay(5)
```

[110]:
```
ts = pd.Series(np.random.randn(len(rng)), rng).tz_localize("UTC")
ts
```

[110]:
```
2024-06-07 00:00:00+00:00   -0.463085
2024-06-07 00:00:00+00:00   -0.143361
2024-06-10 00:00:00+00:00   -0.379153
2024-06-11 00:00:00+00:00    1.383621
```

```
2024-06-12 00:00:00+00:00   -0.579400
2024-06-13 00:00:00+00:00   -0.914193
2024-06-14 00:00:00+00:00    0.363167
2024-06-14 00:00:00+00:00   -0.468667
2024-06-14 00:00:00+00:00   -1.989065
2024-06-17 00:00:00+00:00   -1.350108
2024-06-18 00:00:00+00:00   -0.176041
2024-06-19 00:00:00+00:00   -0.668953
2024-06-20 00:00:00+00:00   -0.401487
2024-06-21 00:00:00+00:00    0.393017
2024-06-21 00:00:00+00:00    0.365841
dtype: float64
```

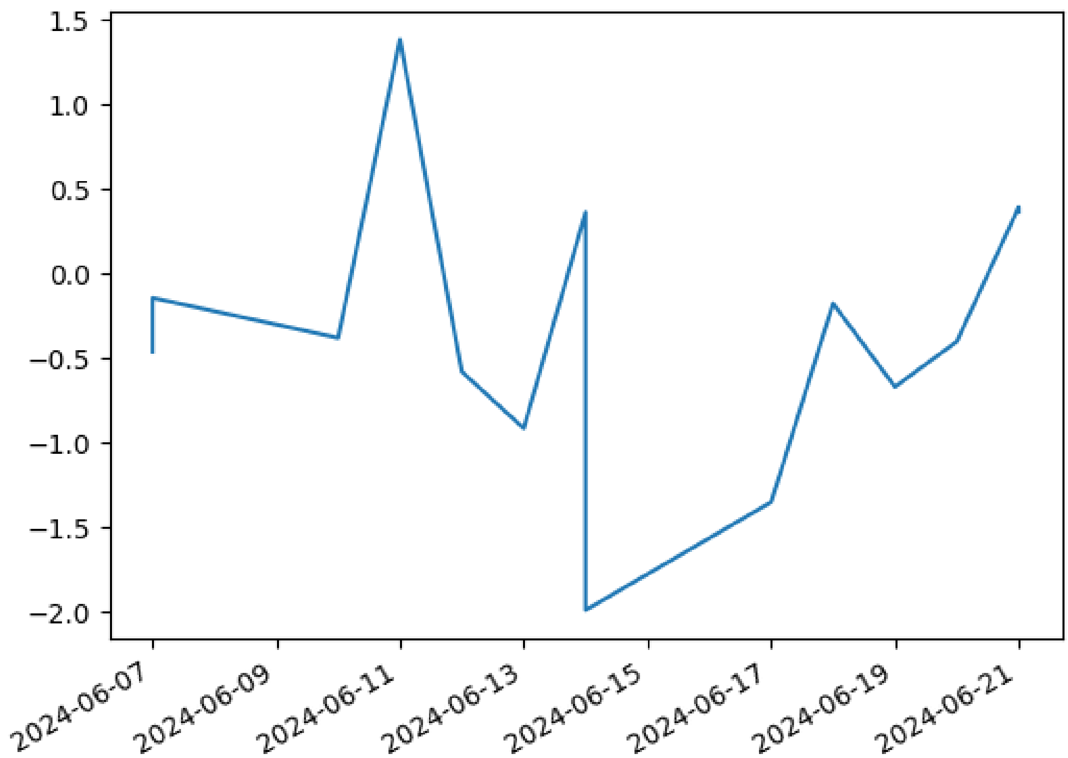
[111]:
```
ts.tz_convert("Europe/Madrid")
```

[111]:
```
2024-06-07 02:00:00+02:00   -0.463085
2024-06-07 02:00:00+02:00   -0.143361
2024-06-10 02:00:00+02:00   -0.379153
2024-06-11 02:00:00+02:00    1.383621
2024-06-12 02:00:00+02:00   -0.579400
2024-06-13 02:00:00+02:00   -0.914193
2024-06-14 02:00:00+02:00    0.363167
2024-06-14 02:00:00+02:00   -0.468667
2024-06-14 02:00:00+02:00   -1.989065
2024-06-17 02:00:00+02:00   -1.350108
2024-06-18 02:00:00+02:00   -0.176041
2024-06-19 02:00:00+02:00   -0.668953
2024-06-20 02:00:00+02:00   -0.401487
2024-06-21 02:00:00+02:00    0.393017
2024-06-21 02:00:00+02:00    0.365841
dtype: float64
```

[112]:
```
import matplotlib.pyplot as plt
```

[113]:
```
ts.plot()
```

[113]:
```
<Axes: >
```



3.13 Categoricals

[114]:
```
iris
```

[114]:
```
      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]
```

[115]:
```python
iris.dtypes
```

[115]:
```
Id                 int64
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species           object
dtype: object
```

[116]:
```python
# Convertimos la columna Species en categoricas:
iris["Species"] = iris["Species"].astype("category")
iris.dtypes
```

[116]:
```
Id                  int64
SepalLengthCm     float64
SepalWidthCm      float64
PetalLengthCm     float64
PetalWidthCm      float64
Species          category
dtype: object
```

3.14 rename_categories()

[117]:
```python
# 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
```

[117]:

	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()

[118]:
```python
# 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
```

[118]:
```
      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()

[119]:
```python
# Colocar las filas según los valores de una columna, en este caso ordenamos
↪por la especie (spc):

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

[119]:
```
      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]
```

[120]:
```python
# Agrupamos para que nos muestre cuantos valores tenemos de cada uno, para ello
↪usamos observed=False en groupby,
# Incluyen categorias vacias si las hubiera:

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

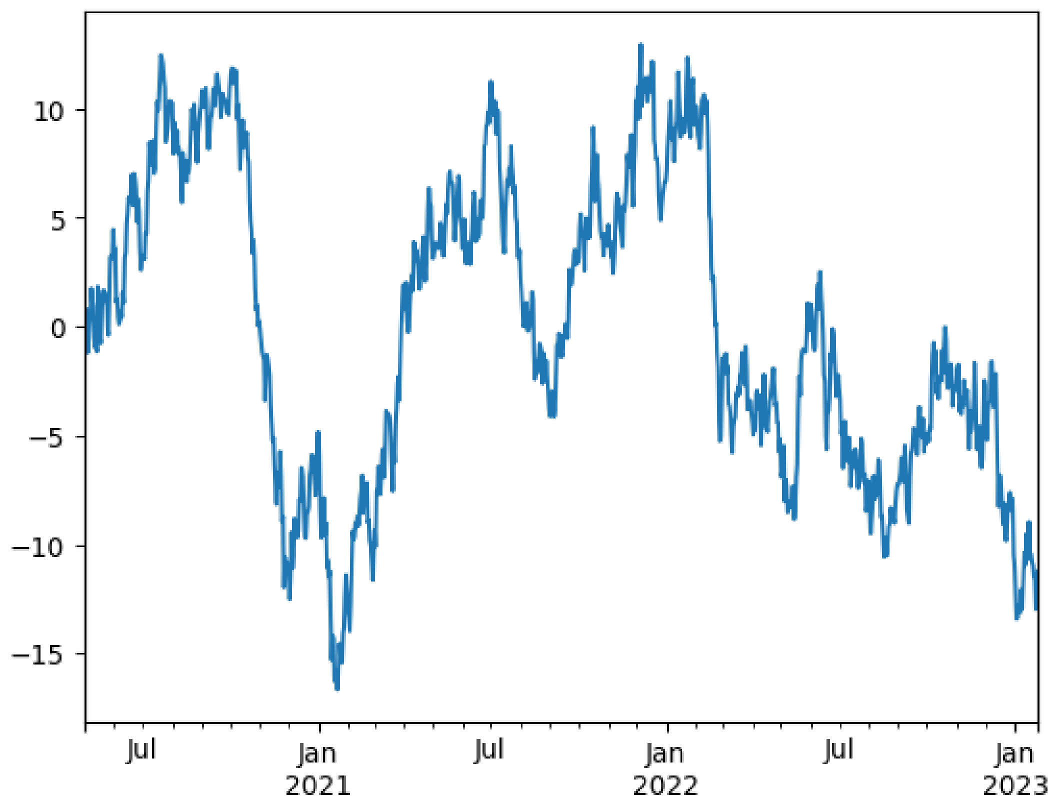
[120]:
```
spc
0    50
1    50
2    50
dtype: int64
```

3.17 Plotting

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

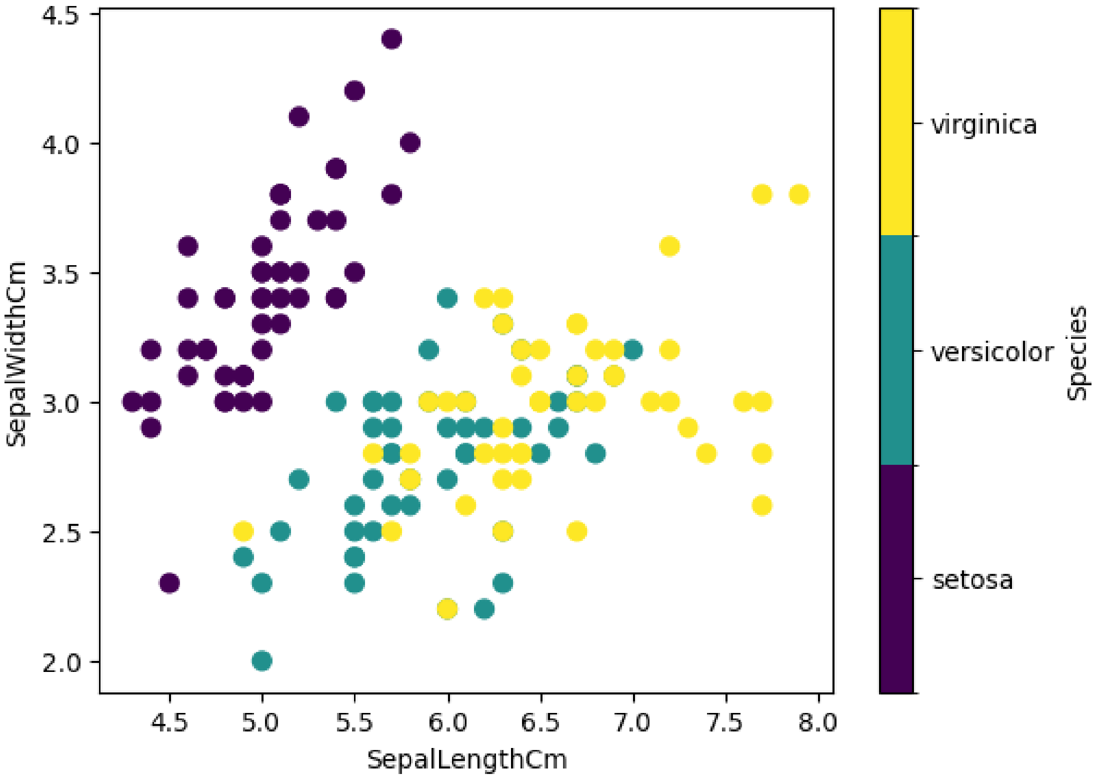
[121]:
```
ts = pd.Series(np.random.randn(1000), index=pd.date_range("5/1/2020",␣
↪periods=1000))

ts = ts.cumsum()

ts.plot()
```

[121]: <Axes: >



[122]:
```
# 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)
```

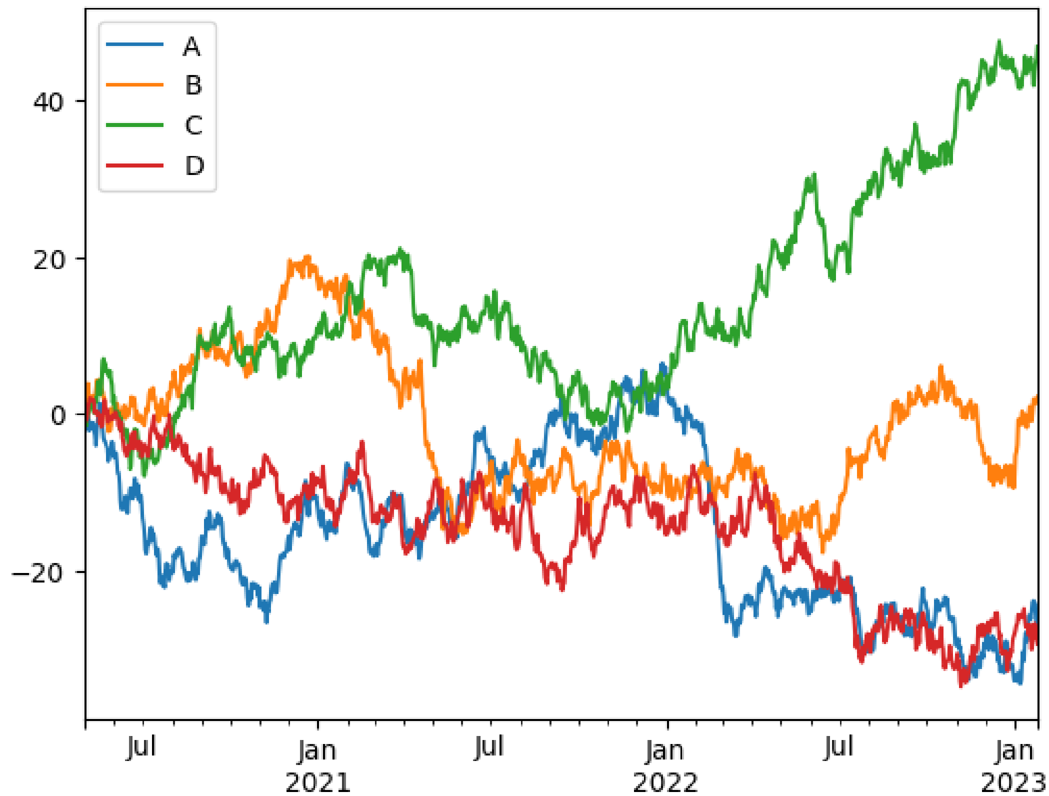
[122]: <Axes: xlabel='SepalLengthCm', ylabel='SepalWidthCm'>



[123]:
```python
# 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')
```

[123]:
```
<matplotlib.legend.Legend at 0x76c890718620>
```

```
<Figure size 640x480 with 0 Axes>
```



4 4.1.3 y 4.1.4: Numpy

1. Método array() (4.1.3)
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 (4.1.3)
11. Matematical functions(4.1.3)

5 Numpy

[124]:
```
# pip install numpy
```

[125]:
```
import numpy as np
```

5.1 Método `array()`

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

[126]:
```python
e = np.array([
    [1, 2],
    [3, 4],
    [5, 6]
])
e
```

[126]:
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

[127]:
```python
len(e)
```

[127]:
```
3
```

[128]:
```python
e.shape
```

[128]:
```
(3, 2)
```

[129]:
```python
e.size
```

[129]:
```
6
```

[130]:
```python
e[0]
```

[130]:
```
array([1, 2])
```

[131]:
```python
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
```

[132]:
```python
a1d = np.array((1, 5, 6))
a1d
```

[132]:
```
array([1, 5, 6])
```

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

[133]:
```python
np.array([127, 128, 129], dtype=np.int8)
```

```
---------------------------------------------------------------------------
OverflowError                             Traceback (most recent call last)
Cell In[133], 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:

[134]:
```python
a = np.array([2, 3, 4], dtype=np.uint32)
b = np.array([5, 6, 7], dtype=np.uint32)
c = a - b
c
```

[134]:
```
array([4294967293, 4294967293, 4294967293], dtype=uint32)
```

[135]:
```python
c_32 = a - b.astype(np.int32)
c_32
```

[135]:
```
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()`:

[136]:
```python
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: 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).

[137]:
```python
np.array(["hello", "world"], dtype="S7").tobytes()
```

[137]:
```
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

[279]:
```
a = np.arange(6)
a
```

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

[280]:
```
type(a)
```

[280]: numpy.ndarray

Formas de imprimir la información

[281]:
```
a
```

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

[282]:
```
for i in a:
    print(i)
```

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

Longitud, forma, tamaño

[283]:
```
a
```

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

[284]:
```
len(a)
```

[284]: 6

[285]:
```
a.shape
```

[285]: (6,)

[286]:
```
a.size
```

[286]: 6

Media, mediana, desviación típica, máximos y mínimos

[287]:
```
a
```

[287]:
```
array([0, 1, 2, 3, 4, 5])
```

[291]:
```
np.mean(a)
```

[291]:
```
np.float64(2.5)
```

[292]:
```
np.median(a)
```

[292]:
```
np.float64(2.5)
```

[293]:
```
np.std(a)
```

[293]:
```
np.float64(1.707825127659933)
```

[288]:
```
max(a)
```

[288]:
```
np.int64(5)
```

[289]:
```
min(a)
```

[289]:
```
np.int64(0)
```

[294]:
```
np.percentile(a, [25, 50, 75])
```

[294]:
```
array([1.25, 2.5 , 3.75])
```

Comprobación de elementos en el array

[149]:
```
a
```

[149]:
```
array([0, 1, 2, 3, 4, 5])
```

[150]:
```
25 in a
```

[150]:
```
False
```

[151]:
```
0 in a
```

[151]:
```
True
```

[152]:
```
25 not in a
```

[152]:
```
True
```

[153]:
```
0 not in a
```

[153]:
```
False
```

Redefinir el tamaño

[154]:
```python
a
```

[154]:
```
array([0, 1, 2, 3, 4, 5])
```

[155]:
```python
a1 = a.reshape(2, 3)
a1
```

[155]:
```
array([[0, 1, 2],
       [3, 4, 5]])
```

Generar números en un intervalo

[156]:
```python
# sin especificar va de 1 en 1
b = np.arange(2,7) # 2, 3, 4, 5, 6
b
```

[156]:
```
array([2, 3, 4, 5, 6])
```

Generar números en un intervalo con salto

[157]:
```python
c = np.arange(10, 40, 5)
c
```

[157]:
```
array([10, 15, 20, 25, 30, 35])
```

[158]:
```python
d = np.arange(10, 41, 5)
d
```

[158]:
```
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:

[159]:
```python
# Definimos un array que empice en 2 y acabe en 9 y sean decimales:

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

[159]:
```
array([2., 3., 4., 5., 6., 7., 8., 9.])
```

5.4 `linspace()`

[160]:
```python
# Recogemos una muestra de los datos, especificamos: min, max, y cada tantos␣
↪recoja un valor
```

[161]:
```python
f = np.linspace(10, 20, 2) # de 10 a 20 con 2 elementos
f
```

[161]:
```
array([10., 20.])
```

[162]:
```python
g = np.linspace(10, 20, 5) # de 10 a 20 muestra 5
g
```

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

[163]:
```
g1 = np.linspace(10, 20, 3) # de 10 a 20 muestra 3
g1
```

[163]: 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)

[164]:
```
h = np.eye(3) # de 3 filas y 3 columnas --> matriz identidad
h
```

[164]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

[165]:
```
i = np.eye(5) # Matriz de 5 filas y 5 columnas
i
```

[165]:
```
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.]])
```

[166]:
```
# n = filas, m = columnas, el resto que no son de la diagonal las rellena con 0:

np.eye(3, 5)
```

[166]:
```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.]])
```

diag()

[167]:
```
# Los elementos estan en la diagonal principal:

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

[167]:
```
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

[168]:
```
# El segundo parámetro es agregar un fila y columna de 0:

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

[168]:
```
array([[0, 1, 0, 0],
       [0, 0, 2, 0],
       [0, 0, 0, 3],
       [0, 0, 0, 0]])
```

vander(x, n)

[169]:
```
# x = array 1d, la lista o tupla de valores, n = al número de columnas:
np.vander([1, 2, 3, 4], 2)
```

[169]:
```
array([[1, 1],
       [2, 1],
       [3, 1],
       [4, 1]])
```

[170]:
```
# 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)
```

[170]:
```
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

[171]:
```
j = 5 * i
j
```

[171]:
```
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

[172]:
```
k = np.ones((3, 4)) # Matriz de 3 filas por 4 columnas --> valores 1
k
```

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

[173]:
```
# se puede añadir un tercer parámetro que es el numero de arrays:

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

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

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

5.7.2 Matriz de todo 0

[174]:
```
l = np.zeros((3, 4)) # Matriz de 0s --> 3 filas por 4 columnas
l
```

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

[175]:
```
l2 = np.zeros((6, 2))
l2
```

[175]:
```
array([[0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.]])
```

[176]:
```
np.zeros((2, 3, 2))  # Idem: a ones()
```

[176]:
```
array([[[0., 0.],
        [0., 0.],
        [0., 0.]],

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

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

5.8 Metodos random() / indices()

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

[177]:
```python
from numpy.random import default_rng

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

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

[177]:
```
array([[0.77395605, 0.43887844, 0.85859792],
       [0.69736803, 0.09417735, 0.97562235]])
```

[178]:
```python
default_rng(42).random((2,3,2))  # idem a ones()
```

[178]:
```
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:

[179]:
```python
# Matriz de 3 filas por 3 columnas:

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

[179]:
```
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

[180]:
```python
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()`

[181]:
```python
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.

[182]:
```python
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()`

[183]:
```
# 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)
```

[183]:
```
array([[0., 0.],
       [1., 1.],
       [2., 4.],
       [3., 9.]])
```

6.2 Modificacion de matrices

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

Intercambio de filas por columnas

[184]:
```
m = np.array([[1, 2, 3],
              [4, 5, 6]])
m
```

[184]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

[185]:
```
# Opción 1
m.transpose()
```

[185]:
```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

[186]:
```
# Opción 2
m.T
```

[186]:
```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

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

[187]:
```
n = np.array([[1, 2, 3],
              [4, 5, 6]])
n
```

[187]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

[188]:
```
# ALL --> ¿Todos los elementos son mayores de 0? --> True/False
np.all(n>0)
```

[188]: np.True_

[189]:
```
np.all(n>2)
```

[189]: np.False_

[190]:
```
# ANY --> ¿Algún elemento son mayores de 2?
np.any(n>2)
```

[190]: np.True_

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

[191]:
```
x = np.array([[1., 2.], [np.nan, 3.], [np.nan, np.nan]])
x
```

[191]:
```
array([[ 1.,  2.],
       [nan,  3.],
       [nan, nan]])
```

[192]:
```
# isnan nos muestra el array resultante con salida de True si es un valor nulo␣
↪o False si no es un valor nulo:

np.isnan(x)
```

[192]:
```
array([[False, False],
       [ True, False],
       [ True,  True]])
```

6.2.3 Función ravel()

[193]:
```
# Pone en una sola dimensión una matriz
```

[194]:
```
p = np.array([[1, 2, 3],
              [4, 5, 6]])
p
```

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

[195]:
```
# np.ravel(matriz a modificar)
np.ravel(p)
```

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

[196]:
```
p1 = np.array([[1, 2, 3],
               [4, 5, 6],
               [7, 8, 9]])
p1
```

[196]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

[197]:
```
np.ravel(p1)
```

[197]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

6.2.4 flatten()

[198]:
```
# Es una copia del array pero en 1 sola dimensión
```

[199]:
```
matriz = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])
matriz
```

[199]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

[200]:
```
# nombre matriz + flatten()
matriz.flatten()
```

[200]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[201]:
```
m = matriz.flatten()
```

[202]:
```
m.shape
```

[202]:
```
(9,)
```

6.2.5 roll()

[203]:
```
# np.roll(array, desplazamiento, eje)
# Desplaza los elementos de manera circular a través de una dimensión
```

[204]:
```
b = np.array([[1, 2, 3, 4],
              [5, 6, 7, 8],
              [9, 10, 11, 12]])
b
```

[204]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

[205]:
```python
# Desplazamiento= 1 y eje horizontal
np.roll(b, 1, axis=0)
```

[205]:
```
array([[ 9, 10, 11, 12],
       [ 1,  2,  3,  4],
       [ 5,  6,  7,  8]])
```

[206]:
```python
# Desplazamiento = 1 y eje vertical
np.roll(b, 1, axis=1)
```

[206]:
```
array([[ 4,  1,  2,  3],
       [ 8,  5,  6,  7],
       [12,  9, 10, 11]])
```

[207]:
```python
# Desplazamiento= -1 y eje horizontal
np.roll(b, -1, axis=0)
```

[207]:
```
array([[ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [ 1,  2,  3,  4]])
```

[208]:
```python
# Desplazamiento = -1 y eje vertical
np.roll(b, -1, axis=1)
```

[208]:
```
array([[ 2,  3,  4,  1],
       [ 6,  7,  8,  5],
       [10, 11, 12,  9]])
```

6.2.6 logspace()

[209]:
```python
# Array de elementos logarítmicos espaciados
```

[210]:
```python
# np.logspace(10^inicio, 10^fin, divisiones(elementos))
# como en linspace se incluye los extremos (inicios-->fin)
c = np.logspace(0, 1, 3)
c
```

[210]:
```
array([ 1.        ,  3.16227766, 10.        ])
```

[211]:
```python
# 10^0 = 1 ; 10^1 = 1; 3 divisiones(elementos)
```

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

6.3 Slicing

6.3.1 Acceso a un elemento de un array:

[212]:
```
matriz = np.array([
    [10, 20],
    [30, 40]
])
matriz
```

[212]:
```
array([[10, 20],
       [30, 40]])
```

[213]:
```
matriz[0][0] # fila 0 columna 0
```

[213]:
```
np.int64(10)
```

[214]:
```
matriz[0][1] # fila 0 columna 1
```

[214]:
```
np.int64(20)
```

Otro ejemplo...

[215]:
```
q = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])
q
```

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

[216]:
```
# Opción 1
q[2][1] # --> fila 2 y columna 1 (listas 0, 1, 2)
```

[216]:
```
np.int64(8)
```

[217]:
```
q[0][2]
```

[217]:
```
np.int64(3)
```

[218]:
```
# Opción 2
q[2, 1]
```

[218]:
```
np.int64(8)
```

[219]:
```
# dos primeras filas (: --> todas)
q[:2]
```

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

[220]:
```
q[2:]
```

[220]: array([[7, 8, 9]])

[221]:
```
# Filtrar por columnas
q[:,[0]]
```

[221]:
```
array([[1],
       [4],
       [7]])
```

[222]:
```
# Filtrar por columnas
q[:,[0,1]]
```

[222]:
```
array([[1, 2],
       [4, 5],
       [7, 8]])
```

[223]:
```
# 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]
```

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

6.3.2 Array de 5 x 5

[224]:
```
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
```

[224]:
```
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

[225]:
```python
a # mostrar la información de la matriz
```

[225]:
```
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]])
```

[226]:
```python
# 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]
```

[226]:
```
array([[ 1,  2],
       [ 6,  7],
       [11, 12],
       [16, 17],
       [21, 22]])
```

[227]:
```python
# todas las columnas de las 2 primeras filas
a[:2]
```

[227]:
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
```

[228]:
```python
a[:2, :]
```

[228]:
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
```

[229]:
```python
a[:, 1:2]
```

[229]:
```
array([[ 2],
       [ 7],
       [12],
       [17],
       [22]])
```

[230]:
```python
# 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...

[231]:
```python
type(a[:,2:])
```

[231]:
```
numpy.ndarray
```

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

[232]:
```
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]])
```

[233]:
```
# Opción 1

a[:, :2]
```

[233]:
```
array([[ 1,  2],
       [ 6,  7],
       [11, 12],
       [16, 17],
       [21, 22]])
```

[234]:
```
# Opción 2

a[:, 0:2]
```

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

6.3.5 Imprimo las pares

[235]:
```
a
```

[235]:
```
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]])
```

[236]:
```
# ":" 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
```

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

[237]:
```
array([[ 2,  4],
       [ 7,  9],
```

```
        [12, 14],
        [17, 19],
        [22, 24]])
```

[238]:
```
a[:, 1::3]
```

[238]:
```
array([[ 2,  5],
       [ 7, 10],
       [12, 15],
       [17, 20],
       [22, 25]])
```

6.3.6 Imprimir las impares

[239]:
```
a
```

[239]:
```
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]])
```

[240]:
```
a[:, 0::2]
```

[240]:
```
array([[ 1,  3,  5],
       [ 6,  8, 10],
       [11, 13, 15],
       [16, 18, 20],
       [21, 23, 25]])
```

[241]:
```
a[:, 0:2:2]
```

[241]:
```
array([[ 1],
       [ 6],
       [11],
       [16],
       [21]])
```

[242]:
```
a[:, 0:3:2]
```

[242]:
```
array([[ 1,  3],
       [ 6,  8],
       [11, 13],
       [16, 18],
       [21, 23]])
```

6.4 Comparacion entre Arrays

[243]:
```
# Creamos los arrays
```

[244]:
```
s = np.array([
    [1, 2, 3],
    [4, 5, 6]
])
s
```

[244]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

[245]:
```
t = np.array([
    [100, 200, 3],
    [400, 5, 6]
])
t
```

[245]:
```
array([[100, 200,   3],
       [400,   5,   6]])
```

Los comparo

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

[246]:
```
np.where(s==t, "True", "False")
```

[246]:
```
array([['False', 'False', 'True'],
       ['False', 'True', 'True']], dtype='<U5')
```

[247]:
```
np.where(s==t, "Si", "No")
```

[247]:
```
array([['No', 'No', 'Si'],
       ['No', 'Si', 'Si']], dtype='<U2')
```

[248]:
```
np.where(s==t, 1, 0)
```

[248]:
```
array([[0, 0, 1],
       [0, 1, 1]])
```

6.5 Concatenación de arrays

Crear los arrays

[249]:
```
y = np.array([
    [1, 2],
    [3, 4]
])
y
```

[249]:
```
array([[1, 2],
       [3, 4]])
```

[250]:
```
z = np.array([
    [5, 6]
])
```

Concatenación por filas

[251]:
```
np.concatenate((y,z), axis=0)
```

[251]:
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

Concatenación por colunmas

[252]:
```
z1 = z.transpose()
z1
```

[252]:
```
array([[5],
       [6]])
```

[253]:
```
np.concatenate((y,z1), axis=1)
```

[253]:
```
array([[1, 2, 5],
       [3, 4, 6]])
```

6.6 Operaciones

[254]:
```
# Potencias
```

[255]:
```
r = np.array([1, 2, 3, 4])
r
```

[255]:
```
array([1, 2, 3, 4])
```

[256]:
```
# Método 1
```

[257]:
```
r**2 # 1^1, 2^2, 3^3, 4^4
```

[257]:
```
array([ 1,  4,  9, 16])
```

[258]:
```
# Método 2
```

[259]:
```
pow(r, 2)
```

[259]:
```
array([ 1,  4,  9, 16])
```

6.6.1 Producto escalar y producto vectorial de 2 vectores

[260]:
```
w = np.array([1, 2, 3])
w
```

[260]:
```
array([1, 2, 3])
```

[261]:
```
x = np.array([2, 5, -4])
x
```

[261]:
```
array([ 2,  5, -4])
```

Producto escalar:

[262]:
```
# w * x = ((1*2) + (2*5) + (3*-4))
```

[263]:
```
# np.dot(matriz1, matriz2)
np.dot(w,x)
```

[263]:
```
np.int64(0)
```

Producto Vectorial:

[264]:
```
## 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)
```

[265]:
```
np.cross(w, x)
```

[265]:
```
array([-23,  10,   1])
```

6.6.2 Matriz con "matrix"

[266]:
```
# 4 filas 4 columnas
u = np.matrix([
    [4, -3, 11, 1],
    [5, 9, 7, 2],
    [2, 3, 4, 1],
    [5, 3, -5, -9]
])
u
```

[266]:
```
matrix([[ 4, -3, 11,  1],
        [ 5,  9,  7,  2],
```

```
        [ 2,  3,  4,  1],
        [ 5,  3, -5, -9]])
```

[267]:
```
# 1 fila y 4 columnas
v = np.matrix([4, 9, 1, 3])
v
```

[267]:
```
matrix([[4, 9, 1, 3]])
```

Suma

[268]:
```
u + v
```

[268]:
```
matrix([[ 8,  6, 12,  4],
        [ 9, 18,  8,  5],
        [ 6, 12,  5,  4],
        [ 9, 12, -4, -6]])
```

Resta

[269]:
```
u - v
```

[269]:
```
matrix([[  0, -12,  10,  -2],
        [  1,   0,   6,  -1],
        [ -2,  -6,   3,  -2],
        [  1,  -6,  -6, -12]])
```

Producto

[270]:
```
u * v
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[270], line 1
----> 1 u * v

File ~/Documentos/PCAD_ES/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)
```

[271]:
```
# ValueError --> es necesario realizar la transpuesta para este caso, ya que
↪las dimensiones No son las adecuadas
```

Opción 1:

[272]:
```
u*v.transpose()
```

[272]:
```
matrix([[  3],
        [114],
        [ 42],
        [ 15]])
```

Opción 2:

[273]:
```
u*v.T
```

[273]:
```
matrix([[  3],
        [114],
        [ 42],
        [ 15]])
```

Opción 3:

[274]:
```
np.dot(u, v.T)
```

[274]:
```
matrix([[  3],
        [114],
        [ 42],
        [ 15]])
```

Traza de una matriz

(suma de los elementos de la diagonal principal)

[275]:
```
u -v
```

[275]:
```
matrix([[  0, -12,  10,  -2],
        [  1,   0,   6,  -1],
        [ -2,  -6,   3,  -2],
        [  1,  -6,  -6, -12]])
```

[276]:
```
type(u-v)
```

[276]:
```
numpy.matrix
```

[277]:
```
np.trace(u-v) # 0 + 0 + 3 + (-12) = -9 (suma de los elementos de la diagonal
↪principal)
```

[277]:
```
np.int64(-9)
```

6.7 Matematical functions

6.7.1 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.

6.7.2 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.
trunc(x, /[, out, where, casting, order, …])	Return the truncated value of the input, element-wise.

6.7.3 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.

Functions	Description
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.

6.7.4 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.
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.

6.7.5 Extrema finding

Functions	Description
maximum(x1, x2, /[, out, where, casting, …])	Element-wise maximum of array elements.

Functions	Description
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