

# Pandas: una introducción a la manipulación de datos con Python

Mathieu Kessler

Departamento de Matemática Aplicada y Estadística  
Universidad Politécnica de Cartagena

@kessler\_mathieu



## Qué es pandas

pandas es una librería para el análisis de datos en Python, que fue desarrollada por Wes McKinney in 2008. Es uno de los proyectos apoyados por [NumFOCUS](#), una asociación que fomenta proyectos y prácticas open source. Destaca por su gran riqueza de procedimientos para manipular y procesar datos, en particular datos con una componente temporal y su integración con numpy.

## Qué es pandas

pandas es una librería para el análisis de datos en Python, que fue desarrollada por Wes McKinney in 2008. Es uno de los proyectos apoyados por [NumFOCUS](#), una asociación que fomenta proyectos y prácticas open source. Destaca por su gran riqueza de procedimientos para manipular y procesar datos, en particular datos con una componente temporal y su integración con numpy.

Para usar pandas en un programa o en un notebook, lo importamos con su alias.

```
import pandas as pd
```

## Series y DataFrames

pandas define dos clases básicas

- **Series:** corresponden a vectores de datos.
  - Es un vector unidimensional.
  - Tienen un “índice” (`index`) que contiene etiquetas para cada dato. Las etiquetas no tienen por que ser únicas.
  - Tienen opcionalmente un nombre (`name`).

## Series y DataFrames

pandas define dos clases básicas

- **Series:** corresponden a vectores de datos.
  - Es un vector unidimensional.
  - Tienen un “índice” (`index`) que contiene etiquetas para cada dato. Las etiquetas no tienen por qué ser únicas.
  - Tienen opcionalmente un nombre (`name`).
- **DataFrame:** Es una estructura 2D que contiene conjuntos de datos,
  - Se puede pensar en una tabla bidimensional, donde cada fila representa un individuo, cada columna es un **Series** que corresponde a una variable o característica del individuo.
  - Un **DataFrame** tiene un “índice” (`index`) con etiquetas asociadas a cada individuo. Las etiquetas no tienen por qué ser únicas.
  - Cada columna tiene un nombre.
  - Podemos ver un **DataFrame** como un dict de **Series**: las claves son los nombres de las columnas, sus valores son las **Series**.

La manera general de crear un Series es

```
pd.Series(data, index=None)
```

- data puede ser un vector o iterable (por ejemplo una lista), un `dict` o un valor escalar.
- Si no se proporciona un valor para `index`, se usarán enteros por defecto
- Admite el argumento opcional `name`.

Ejemplos:

```
s1 = pd.Series(  
    [3, 2.5, -1],  
    index=['a', 'b', 'c'],  
    name='medición'  
)  
s2 = pd.Series(  
    {'a': 3, 'b': 2.5, 'c': -1},  
    name='medición'  
)
```

Los `index` son un concepto clave en `Series` o `DataFrame`, en particular porque sirven para “alinear” las series cuando se combinan a través de operaciones, como sumas, restas, etc...

```
>>> s1 = pd.Series(  
...     [3, 2.5, -1],  
...     index=['a', 'b', 'c'],  
... )  
>>> s2 = pd.Series(  
...     [1, 0.5, 2],  
...     index=['b', 'c', 'd']  
... )  
>>> s1 + s2  
a    NaN  
b    3.5  
c   -0.5  
d    NaN  
dtype: float64
```

Ejemplo con un índice duplicado:

```
>>> s1 = pd.Series(  
...     [3, 2.5, -1],  
...     index=['a', 'b', 'c'],  
... )  
>>> s2 = pd.Series(  
...     [1, 0.5, 2],  
...     index=['b', 'c', 'b']  
... )  
>>> s1 + s2  
a      NaN  
b      3.5  
b      4.5  
c     -0.5  
dtype: float64
```



Y qué pasará en este caso, según vosotros?

```
>>> s1 = pd.Series(  
...     [3, 2.5, -1, 10],  
...     index=['a', 'b', 'c', 'b'],  
... )  
>>> s2 = pd.Series(  
...     [1, 0.5, 2],  
...     index=['b', 'c', 'b']  
... )  
>>> s1 + s2  
a      NaN  
b       3.5  
b       4.5  
b      11.0  
b      12.0  
c      -0.5  
dtype: float64
```

Podéis encontrar en la referencia de la API de Pandas todos los atributos y métodos asociados a un Serie: [pandas.Series](#)



[Getting started](#) [User Guide](#) **[API reference](#)** [Development](#) [Release notes](#)

🔍 Search the docs ...

Input/output

General functions

Series

**pandas.Series**

pandas.Series.Index

pandas.Series.array

pandas.Series.values

pandas.Series.dtype

pandas.Series.shape

pandas.Series.nbytes

pandas.Series.ndim

pandas.Series.size

## Methods

<code>abs()</code>	Return a Series/DataFrame with absolute numeric value of element.
<code>add(other[, level, fill_value, axis])</code>	Return Addition of series and other, element-wise (binary operator <i>add</i> ).
<code>add_prefix(prefix)</code>	Prefix labels with string <i>prefix</i> .
<code>add_suffix(suffix)</code>	Suffix labels with string <i>suffix</i> .
<code>agg([func, axis])</code>	Aggregate using one or more operations over the specified
<code>aggregate([func, axis])</code>	Aggregate using one or more operations over the specified
<code>align(other[, join, axis, level, copy, ...])</code>	Align two objects on their axes with the specified join met
<code>all([axis, bool_only, skipna, level])</code>	Return whether all elements are True, potentially over an :

Estos métodos permiten aplicar una operación al conjunto de la Serie, sin tener que hacer un bucle recorriendo sus elementos.

Por ejemplo:

```
>>> s1 = pd.Series(  
...     [3, 2.5, -1, 2.5],  
...     index=['a', 'b', 'c', 'b'],  
... )  
>>> s1.sum()  
7.0
```

Estos métodos permiten aplicar una operación al conjunto de la Serie, sin tener que hacer un bucle recorriendo sus elementos.

Por ejemplo:

```
>>> s1 = pd.Series(  
...     [3, 2.5, -1, 2.5],  
...     index=['a', 'b', 'c', 'b'],  
... )
```

```
>>> s1.sum()
```

```
7.0
```

o

```
>>> s1.drop_duplicates()
```

```
a    3.0
```

```
b    2.5
```

```
c   -1.0
```

```
dtype: float64
```

## La estructura de datos principal de pandas

La manera general de crear un DataFrame es

```
pd.DataFrame(data, index=None, columns=None)
```

- data puede ser un array numpy, un iterable (por ejemplo una lista de listas), un `dict`.
- Si no se proporciona un valor para `index`, se usarán enteros por defecto
- El argumento `columns` indica las etiquetas para las columnas.
- data puede ser un `dict` donde las pares `key:value` son etiqueta:valores de cada columna.

## La estructura de datos principal de pandas

La manera general de crear un DataFrame es

```
pd.DataFrame(data, index=None, columns=None)
```

- data puede ser un array numpy, un iterable (por ejemplo una lista de listas), un **dict**.
- Si no se proporciona un valor para **index**, se usarán enteros por defecto
- El argumento **columns** indica las etiquetas para las columnas.
- data puede ser un **dict** donde las pares key:value son etiqueta:valores de cada columna.

Ejemplos:

```
df1 = pd.DataFrame(  
    [[3, 'r'],  
     [2.5, 'g'],  
     [-1, 'b']],  
    index=['a', 'b', 'c'],  
    columns=['x', 'color']  
)  
df2 = pd.DataFrame(  
    {'x': [3, 2.5, -1],  
     'color': ['r', 'g', 'b']},  
    index=['a', 'b', 'c']  
)
```

data también podría ser un `dict` de Series, lo que permite más flexibilidad con los índices:

```
>>> pd.DataFrame(  
...     {'x': pd.Series([3, 2.5, -1, 0], index=['a', 'b', 'c', 'd']),  
...     'color': pd.Series(['r', 'g', 'b'], index=['a', 'b', 'c'])}  
... )  
      x color  
a  3.0     r  
b  2.5     g  
c -1.0     b  
d  0.0    NaN
```

## Métodos y atributos útiles con DataFrame

Si df es un pandas DataFrame

*# Métodos útiles para hacerse una idea del conjunto:*

`df.info()` *# número de filas, columnas y tipo de datos*

`df.describe()` *# resumen numérico de cada columna*

`df.head(7)` *# las 7 primeras filas*

`df.tail(7)` *# las 7 últimas filas*

*# Atributos útiles:*

`df.shape` *# Dimensión del conjunto*

`df.columns` *# etiquetas de las columnas*

`df.index` *# etiquetas de las filas*

`df.values` *# valores como ndarray de numpy*

`df.dtypes` *# tipos de datos de cada columnas*



## Construir DataFrame

En todas las prácticas leeremos los datos desde una fuente externa.

- Desde un fichero local que nos habremos descargado
- Desde un fichero situado en internet
- Desde una API de un servicio

## Construir DataFrame

En todas las prácticas leeremos los datos desde una fuente externa.

- Desde un fichero local que nos habremos descargado
- Desde un fichero situado en internet
- Desde una API de un servicio

Para leer desde un archivo local, usaremos casi siempre `pd.read_csv`, que permite muchísima flexibilidad.

## Construir DataFrame

En todas las prácticas leeremos los datos desde una fuente externa.

- Desde un fichero local que nos habremos descargado
- Desde un fichero situado en internet
- Desde una API de un servicio

Para leer desde un archivo local, usaremos casi siempre `pd.read_csv`, que permite muchísima flexibilidad.

`pd.read_csv` utiliza las primeras líneas del fichero para adivinar su estructura. Por ejemplo, infiere cómo están separados las columnas, si tiene cabecera, etc. Por lo tanto, en muchos casos, bastará con

```
pd.read_csv(<camino hasta el fichero de datos>)
```

por ejemplo

```
pd.read_csv('data/datos.csv')
```

## Parámetros más útiles de `pd.read_csv`

En el caso en que tuviéramos que especificar parámetros porque `pd.read_csv` no ha adivinado correctamente la estructura del fichero, éstos son útiles:

- `sep`: un `str` que contiene el caracter que delimita las columnas
- `skiprows`: un entero, número de filas que saltas al inicio del fichero. También puede ser una lista de los índices de filas.
- `encoding`: la codificación del fichero. Normalmente será 'utf-8', pero podría ser 'latin-1'.
- `thousands`: un `str` que contiene el caracter que separa los miles
- `dec`: un `str` que contiene el caracter que hace de separador decimal.

## Parámetros más útiles de `pd.read_csv`

En el caso en que tuviéramos que especificar parámetros porque `pd.read_csv` no ha adivinado correctamente la estructura del fichero, éstos son útiles:

- `sep`: un `str` que contiene el caracter que delimita las columnas
- `skiprows`: un entero, número de filas que saltas al inicio del fichero. También puede ser una lista de los índices de filas.
- `encoding`: la codificación del fichero. Normalmente será 'utf-8', pero podría ser 'latin-1'.
- `thousands`: un `str` que contiene el caracter que separa los miles
- `dec`: un `str` que contiene el caracter que hace de separador decimal.

Por ejemplo

```
df = pd.read_csv('data/datos.csv', sep = ';', skiprows = 3,  
                 encoding = 'latin-1')
```