Módulo I: Introducción a Python

Temario de la Clase 6
Librería Pandas

- Series
- DataFrames
  - Indexado
  - Selección condicional
  - Máscaras
  - Operaciones vectorizadas
  - Modificación
  - Borrado

- CSV

Pandas

Pandas es una librería de Python especializada en el manejo y análisis de estructuras de datos

```
import pandas as pd
```

- Su nombre viene de Panel de Datos.
- Me permite trabajar con datos tabulares, es decir, en forma de tabla.
- A diferencia de los arrays de NumPy, Pandas me permite combinar distintos tipos de variables (numéricas, strings, booleanos) en una misma estructura.
- Permite tratar con bases de datos incompletas.
- Estructuras intuitivas y flexibles, al tratarse de datos etiquetados.
- Está basada en estructuras de NumPy, por lo que hereda sus propiedades de optimización y rapidez.

Series

Una serie de Pandas es un objeto comparable con un array uni-dimensional de NumPy.
Todos los datos son del mismo tipo (como un array).
Se pueden asignar índices para trabajar con sus elementos.
Se pueden crear a partir de una lista o un diccionario.

Se puede definir de la siguiente manera:

```
series = pd.Series(data=None, index=None, dtype=None, name=None)
```

Con **data** podemos pasar la información que queremos almacenar en la serie, **index** nos permite especificar los índices a utilizar, **dtype** se usa de forma similar a Numpy, y **name** me permite darle un nombre a la serie (independiente del nombre de la variable en la que se guarda).

Series

Para ejemplificar la creación de una serie, voy a trabajar con datos de la población del “Grupo de los 7” (G7). Solamente voy a especificar los valores.

```
[3]  # In millions
     g7_pop = pd.Series([35.467, 63.951, 80.940, 60.665, 127.061, 64.511, 318.523])
```
✓ 0.0s

```
[4]  g7_pop
```
✓ 0.0s

```
0     35.467
1     63.951
2     80.940
3     60.665
4    127.061
5     64.511
6    318.523
dtype: float64
```

Series

Le puedo dar un nombre a la serie para identificar mejor su contenido. Al igual que en listas puedo acceder a los elementos mediante su posición.

```
[5]  g7_pop.name = 'G7 Population in millions'
     ✓ 0.0s
```

```
[6]  g7_pop
     ✓ 0.0s
```

```
0     35.467
1     63.951
2     80.940
3     60.665
4    127.061
5     64.511
6    318.523
Name: G7 Population in millions, dtype: float64
```

```
[11] g7_pop[0]
     ✓ 0.0s
```

```
35.467
```

```
[12] g7_pop[1]
     ✓ 0.0s
```

```
63.951
```

```
[13] g7_pop.index
     ✓ 0.0s
```

```
RangeIndex(start=0, stop=7, step=1)
```

Series

Vemos de qué tipo es el objeto creado y que su estructura interna está basada en arrays de NumPy.

```
[67]  type(g7_pop)
      ✓ 0.0s
...   pandas.core.series.Series
```

```
[7]   g7_pop.dtype
      ✓ 0.0s
...   dtype('float64')
```

```
[8]   g7_pop.values
      ✓ 0.0s
...   array([ 35.467,  63.951,  80.94 ,  60.665, 127.061,  64.511, 318.523])
```

```
[9]   type(g7_pop.values)
      ✓ 0.0s
...   numpy.ndarray
```

Series

A diferencia de las listas, puedo especificar el índice de una serie.

```
g7_pop.index = [
    'Canada',
    'France',
    'Germany',
    'Italy',
    'Japan',
    'United Kingdom',
    'United States',
]
```

[14] ✓ 0.0s

```
g7_pop
```

[15] ✓ 0.0s

```
Canada            35.467
France            63.951
Germany           80.940
Italy             60.665
Japan            127.061
United Kingdom    64.511
United States    318.523
Name: G7 Population in millions, dtype: float64
```

Series

Puedo crear directamente desde un diccionario o especificar al crear el objeto.

```
pd.Series(
    [35.467, 63.951, 80.94, 60.665, 127.061, 64.511, 318.523],
    index=['Canada', 'France', 'Germany', 'Italy', 'Japan', 'United Kingdom',
           'United States'],
    name='G7 Population in millions')
```

[17] ✓ 0.0s

```
Canada             35.467
France             63.951
Germany            80.940
Italy              60.665
Japan             127.061
United Kingdom     64.511
United States     318.523
Name: G7 Population in millions, dtype: float64
```

```
pd.Series({
    'Canada': 35.467,
    'France': 63.951,
    'Germany': 80.94,
    'Italy': 60.665,
    'Japan': 127.061,
    'United Kingdom': 64.511,
    'United States': 318.523
}, name='G7 Population in millions')
```

[16] ✓ 0.0s

```
Canada             35.467
France             63.951
Germany            80.940
Italy              60.665
Japan             127.061
United Kingdom     64.511
United States     318.523
Name: G7 Population in millions, dtype: float64
```

Series: indexado

Así como en los diccionarios y las listas, puedo usar el índice para acceder a un elemento.

```
[19]  g7_pop
      ✓ 0.0s
```

```
Canada             35.467
France             63.951
Germany            80.940
Italy              60.665
Japan             127.061
United Kingdom     64.511
United States     318.523
Name: G7 Population in millions, dtype: float64
```

```
[20]  g7_pop['Canada']
      ✓ 0.0s
```

```
35.467
```

```
[21]  g7_pop['Japan']
      ✓ 0.0s
```

```
127.061
```

Series: iloc

El comando `iloc` me permite utilizar los índices numéricos de los elementos. El último caso me devuelve una serie nueva.

```
[22]  g7_pop.iloc[0]
      ✓ 0.0s
...   35.467
```

```
[23]  g7_pop.iloc[-1]
      ✓ 0.0s
...   318.523
```

```
[27]  g7_pop.iloc[0:2]
      ✓ 0.0s
...   Canada    35.467
      France    63.951
      Name: G7 Population in millions, dtype: float64
```

Series: máscaras y slicing

También puedo trabajar con máscaras y slicing, en este caso el elemento superior también es incluido en el resultado.

```
[24]  g7_pop[['Italy', 'France']]
      ✓ 0.0s

...   Italy     60.665
      France    63.951
      Name: G7 Population in millions, dtype: float64
```

```
[25]  g7_pop.iloc[[0, 1]]
      ✓ 0.0s

...   Canada    35.467
      France    63.951
      Name: G7 Population in millions, dtype: float64
```

```
[26]  g7_pop['Canada': 'Italy']
      ✓ 0.0s

...   Canada     35.467
      France     63.951
      Germany    80.940
      Italy      60.665
      Name: G7 Population in millions, dtype: float64
```

Series: selección condicional

Se pueden aplicar condiciones booleanas al igual que en NumPy.

```
[35]  g7_pop
      ✓ 0.0s
```

```
Canada             35.467
France             63.951
Germany            80.940
Italy              60.665
Japan             127.061
United Kingdom     64.511
United States     318.523
Name: G7 Population in millions, dtype: float64
```

```
[36]  g7_pop > 70
      ✓ 0.0s
```

```
Canada            False
France            False
Germany            True
Italy             False
Japan              True
United Kingdom    False
United States      True
Name: G7 Population in millions, dtype: bool
```

```
[37]  g7_pop[g7_pop > 70]
      ✓ 0.0s
```

```
Germany           80.940
Japan            127.061
United States    318.523
Name: G7 Population in millions, dtype: float64
```

```
[38]  g7_pop.mean()
      ✓ 0.0s
```

```
107.30257142857144
```

```
[39]  g7_pop[g7_pop > g7_pop.mean()]
      ✓ 0.0s
```

```
Japan            127.061
United States    318.523
Name: G7 Population in millions, dtype: float64
```

Series: selección condicional

Se pueden aplicar condiciones booleanas al igual que en NumPy.

```
g7_pop[(g7_pop > 80) | (g7_pop < 40)]
```

[58] ✓ 0.0s

```
Canada            35.467
Germany           80.940
Japan            127.061
United States    318.523
Name: G7 Population in millions, dtype: float64
```

```
g7_pop['France': 'Italy'].mean()
```

[50] ✓ 0.0s

```
68.51866666666666
```

```
g7_pop[(g7_pop > 80) & (g7_pop < 200)]
```

[59] ✓ 0.0s

```
Germany     80.940
Japan      127.061
Name: G7 Population in millions, dtype: float64
```

Series: selección condicional

```
[43]  g7_pop
      ✓ 0.0s
```

```
Canada             35.467
France             63.951
Germany            80.940
Italy              60.665
Japan             127.061
United Kingdom     64.511
United States     318.523
Name: G7 Population in millions, dtype: float64
```

```
[44]  g7_pop.mean(), g7_pop.std()
      ✓ 0.0s
```

```
(107.30257142857144, 97.24996987121581)
```

```
[42]  g7_pop[(g7_pop > g7_pop.mean() - g7_pop.std() / 2) & (g7_pop < g7_pop.mean() + g7_pop.std() / 2)]
      ✓ 0.0s
```

```
France             63.951
Germany            80.940
Italy              60.665
Japan             127.061
United Kingdom     64.511
Name: G7 Population in millions, dtype: float64
```

Series: operaciones vectorizadas

```
[47]  g7_pop * 1000000
      ✓ 0.0s
```

```
Canada             35467000.0
France             63951000.0
Germany            80940000.0
Italy              60665000.0
Japan             127061000.0
United Kingdom     64511000.0
United States     318523000.0
Name: G7 Population in millions, dtype: float64
```

```
[48]  np.log(g7_pop)
      ✓ 0.0s
```

```
Canada            3.568603
France            4.158117
Germany           4.393708
Italy             4.105367
Japan             4.844667
United Kingdom    4.166836
United States     5.763695
Name: G7 Population in millions, dtype: float64
```

```
[54]  g7_pop + 30
      ✓ 0.0s
```

```
Canada             65.467
France             93.951
Germany           110.940
Italy              90.665
Japan             157.061
United Kingdom     94.511
United States     348.523
Name: G7 Population in millions, dtype: float64
```

```
[53]  np.exp(g7_pop)
      ✓ 0.0s
```

```
Canada            2.530011e+15
France            5.936991e+27
Germany           1.418389e+35
Italy             2.220623e+26
Japan             1.520167e+55
United Kingdom    1.039373e+28
United States     2.151698e+138
Name: G7 Population in millions, dtype: float64
```

Series: modificación

Hasta ahora vimos que al realizar operaciones sobre una serie de Pandas me devuelve una serie nueva. Veamos como modificar datos de un objeto existente.

```
[60]  g7_pop['Canada'] = 40.5
      ✓ 0.0s
```

```
[61]  g7_pop
      ✓ 0.0s
```

```
Canada             40.500
France             63.951
Germany            80.940
Italy              60.665
Japan             127.061
United Kingdom     64.511
United States     318.523
Name: G7 Population in millions, dtype: float64
```

```
[62]  g7_pop.iloc[-1] = 500
      ✓ 0.0s
```

```
[63]  g7_pop
      ✓ 0.0s
```

```
Canada             40.500
France             63.951
Germany            80.940
Italy              60.665
Japan             127.061
United Kingdom     64.511
United States     500.000
Name: G7 Population in millions, dtype: float64
```

Series: modificación

```
[64]  g7_pop[g7_pop < 70]
      ✓ 0.0s
```

```
Canada            40.500
France            63.951
Italy             60.665
United Kingdom    64.511
Name: G7 Population in millions, dtype: float64
```

```
[65]  g7_pop[g7_pop < 70] = 99.99
      ✓ 0.0s
```

```
[66]  g7_pop
      ✓ 0.0s
```

```
Canada             99.990
France             99.990
Germany            80.940
Italy              99.990
Japan             127.061
United Kingdom     99.990
United States     500.000
Name: G7 Population in millions, dtype: float64
```

Data Frames

Los Data Frames son estructuras de Pandas más complejos que las Series. En lugar de tener una sola lista de valores, puedo tener varias columnas, y cada columna corresponde un mismo tipo de dato.

La información contenida en un data frame puede ser ingresada manualmente, pero lo mas común es importarla desde algún archivo que trabaje con datos tabulares, como ser el CSV.

G7 Stats					
	Population	**GDP**	**Surface**	**HDI**	**Continent**
Canada	35.467	1,785,387.00	9,984,670	0.913	America
France	63.951	2,833,687.00	640,679	0.888	Europe
Germany	80.94	3,874,437.00	357,114	0.916	Europe
Italy	60.665	2,167,744.00	301,336	0.873	Europe
Japan	127.061	4,602,367.00	377,930	0.891	Asia
United Kingdom	64.511	2,950,039.00	242,495	0.907	Europe
United States	318.523	17,348,075.00	9,525,067	0.915	America

Data Frames: creación

```
df = pd.DataFrame({
    'Population': [35.467, 63.951, 80.94 , 60.665, 127.061, 64.511, 318.523],
    'GDP': [
        1785387, 2833687, 3874437, 2167744, 4602367,2950039, 17348075
    ],
    'Surface Area': [
        9984670, 640679, 357114, 301336, 377930, 242495, 9525067
    ],
    'HDI': [
        0.913, 0.888, 0.916, 0.873, 0.891, 0.907, 0.915
    ],
    'Continent': [
        'America', 'Europe', 'Europe', 'Europe', 'Asia', 'Europe', 'America'
    ]
})
```

✓ 0.0s

	Population	GDP	Surface Area	HDI	Continent
0	35.467	1785387	9984670	0.913	America
1	63.951	2833687	640679	0.888	Europe
2	80.940	3874437	357114	0.916	Europe
3	60.665	2167744	301336	0.873	Europe
4	127.061	4602367	377930	0.891	Asia
5	64.511	2950039	242495	0.907	Europe
6	318.523	17348075	9525067	0.915	America

Data Frames: índices

Puedo tener un resumen de los datos en el DF.

```
df.index = [
    'Canada',
    'France',
    'Germany',
    'Italy',
    'Japan',
    'United Kingdom',
    'United States',
]
```

[4] ✓ 0.0s

```
df
```

[5] ✓ 0.0s

	Population	GDP	Surface Area	HDI	Continent
Canada	35.467	1785387	9984670	0.913	America
France	63.951	2833687	640679	0.888	Europe
Germany	80.940	3874437	357114	0.916	Europe
Italy	60.665	2167744	301336	0.873	Europe
Japan	127.061	4602367	377930	0.891	Asia
United Kingdom	64.511	2950039	242495	0.907	Europe
United States	318.523	17348075	9525067	0.915	America

```
df.info()
```

[8] ✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
Index: 7 entries, Canada to United States
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Population    7 non-null      float64
 1   GDP           7 non-null      int64
 2   Surface Area  7 non-null      int64
 3   HDI           7 non-null      float64
 4   Continent     7 non-null      object
dtypes: float64(2), int64(2), object(1)
memory usage: 336.0+ bytes
```

Data Frames: dimensiones

Para conocer las dimensiones del data frame:

```
[9]  df.size
     ✓ 0.0s
...  35
```

```
[10] df.shape
     ✓ 0.0s
...  (7, 5)
```

```
[6]  df.columns
     ✓ 0.0s
...  Index(['Population', 'GDP', 'Surface Area', 'HDI', 'Continent'], dtype='object')
```

```
[7]  df.index
     ✓ 0.0s
...  Index(['Canada', 'France', 'Germany', 'Italy', 'Japan', 'United Kingdom',
            'United States'],
           dtype='object')
```

Data Frames: información estadística

```
df.describe()
```

[11] ✓ 0.0s

	Population	GDP	Surface Area	HDI
count	7.000000	7.000000e+00	7.000000e+00	7.000000
mean	107.302571	5.080248e+06	3.061327e+06	0.900429
std	97.249970	5.494020e+06	4.576187e+06	0.016592
min	35.467000	1.785387e+06	2.424950e+05	0.873000
25%	62.308000	2.500716e+06	3.292250e+05	0.889500
50%	64.511000	2.950039e+06	3.779300e+05	0.907000
75%	104.000500	4.238402e+06	5.082873e+06	0.914000
max	318.523000	1.734808e+07	9.984670e+06	0.916000

```
df.dtypes
```

[12] ✓ 0.0s

```
Population      float64
GDP               int64
Surface Area      int64
HDI             float64
Continent        object
dtype: object
```

Data Frames: filtros

```
df['Population']
```

[17] ✓ 0.0s

```
Canada            35.467
France            63.951
Germany           80.940
Italy             60.665
Japan            127.061
United Kingdom    64.511
United States    318.523
Name: Population, dtype: float64
```

```
type(df['Population'])
```

[18] ✓ 0.0s

```
pandas.core.series.Series
```

```
df[['Population', 'GDP']]
```

[21] ✓ 0.0s

	Population	GDP
Canada	35.467	1785387
France	63.951	2833687
Germany	80.940	3874437
Italy	60.665	2167744
Japan	127.061	4602367
United Kingdom	64.511	2950039
United States	318.523	17348075

```
df[:3]
```

[22] ✓ 0.0s

	Population	GDP	Surface Area	HDI	Continent
Canada	35.467	1785387	9984670	0.913	America
France	63.951	2833687	640679	0.888	Europe
Germany	80.940	3874437	357114	0.916	Europe

Data Frames: loc y iloc

Con `loc` puedo filtrar según el nombre del índice o de la columna.

Con `loc` puedo filtrar según el valor numérico del índice o de la columna.

```
[15]  df.loc['Canada']
      ✓ 0.0s

Population       35.467
GDP            1785387
Surface Area    9984670
HDI               0.913
Continent        America
Name: Canada, dtype: object
```

```
[16]  df.iloc[-1]
      ✓ 0.0s

Population       318.523
GDP            17348075
Surface Area    9525067
HDI               0.915
Continent        America
Name: United States, dtype: object
```

```
df.loc['Japan', 'Surface Area']
✓ 0.0s

377930
```

```
df.iloc[4, 2]
✓ 0.0s

377930
```

Data Frames: loc

```
df.loc['Italy']
```

[23] ✓ 0.0s

```
Population       60.665
GDP             2167744
Surface Area     301336
HDI               0.873
Continent        Europe
Name: Italy, dtype: object
```

```
df.loc['France': 'Italy']
```

[24] ✓ 0.0s

	Population	GDP	Surface Area	HDI	Continent
France	63.951	2833687	640679	0.888	Europe
Germany	80.940	3874437	357114	0.916	Europe
Italy	60.665	2167744	301336	0.873	Europe

```
df.loc['France': 'Italy', 'Population']
```

[25] ✓ 0.0s

```
France     63.951
Germany    80.940
Italy      60.665
Name: Population, dtype: float64
```

```
df.loc['France': 'Italy', ['Population', 'GDP']]
```

[26] ✓ 0.0s

	Population	GDP
France	63.951	2833687
Germany	80.940	3874437
Italy	60.665	2167744

Data Frames: iloc

```
[30]  df.iloc[[0, 1, -1]]
      ✓ 0.0s
```

	Population	GDP	Surface Area	HDI	Continent
Canada	35.467	1785387	9984670	0.913	America
France	63.951	2833687	640679	0.888	Europe
United States	318.523	17348075	9525067	0.915	America

```
[31]  df.iloc[1:3]
      ✓ 0.0s
```

	Population	GDP	Surface Area	HDI	Continent
France	63.951	2833687	640679	0.888	Europe
Germany	80.940	3874437	357114	0.916	Europe

```
[32]  df.iloc[1:3, 3]
      ✓ 0.0s
```

```
France     0.888
Germany    0.916
Name: HDI, dtype: float64
```

```
[33]  df.iloc[1:3, [0, 3]]
      ✓ 0.0s
```

	Population	HDI
France	63.951	0.888
Germany	80.940	0.916

```
[34]  df.iloc[1:3, 1:3]
      ✓ 0.0s
```

	GDP	Surface Area
France	2833687	640679
Germany	3874437	357114

Data Frames: selección condicional

```
df
```

[35] ✓ 0.0s

	Population	GDP	Surface Area	HDI	Continent
Canada	35.467	1785387	9984670	0.913	America
France	63.951	2833687	640679	0.888	Europe
Germany	80.940	3874437	357114	0.916	Europe
Italy	60.665	2167744	301336	0.873	Europe
Japan	127.061	4602367	377930	0.891	Asia
United Kingdom	64.511	2950039	242495	0.907	Europe
United States	318.523	17348075	9525067	0.915	America

```
df['Population'] > 70
```

[36] ✓ 0.0s

```
Canada           False
France           False
Germany           True
Italy            False
Japan             True
United Kingdom   False
United States     True
Name: Population, dtype: bool
```

Data Frames: selección condicional

```
df.loc[df['Population'] > 70, 'Population']
```

✓ 0.0s

```
Germany          80.940
Japan           127.061
United States   318.523
Name: Population, dtype: float64
```

```
df.loc[df['Population'] > 70, ['Population', 'GDP']]
```

✓ 0.0s

	Population	GDP
Germany	80.940	3874437
Japan	127.061	4602367
United States	318.523	17348075

```
df.loc[df['Population'] > 70]
```

✓ 0.0s

	Population	GDP	Surface Area	HDI	Continent
Germany	80.940	3874437	357114	0.916	Europe
Japan	127.061	4602367	377930	0.891	Asia
United States	318.523	17348075	9525067	0.915	America

Data Frames: borrado de datos

Estos métodos devuelven un data frame nuevo, para modificar el DF existente se usa el parámetro `inplace`.

```
df.drop('Canada')
```

```
df.drop(['Canada', 'Japan'])
```

```
df.drop(columns=['Population', 'HDI'])
```

```
df.drop(['Italy', 'Canada'], axis=0)
```

```
df.drop(['Population', 'HDI'], axis=1)
```

```
df.drop(['Population', 'HDI'], axis=1)
```

```
df.drop(['Population', 'HDI'], axis='columns')
```

```
df.drop(['Canada', 'Germany'], axis='rows')
```

```
df.drop(columns='Language', inplace=True)
```

[63] ✓ 0.0s

Data Frames: borrado de datos

Estos métodos devuelven un data frame nuevo, para modificar el DF existente se usa el parámetro `inplace`.

```
[67]  df
      ✓ 0.0s
```

	Population	GDP	Surface Area	HDI	Continent
Canada	35.467	1785387	9984670	0.913	America
France	63.951	2833687	640679	0.888	Europe
Germany	80.940	3874437	357114	0.916	Europe
Italy	60.665	2167744	301336	0.873	Europe
Japan	127.061	4602367	377930	0.891	Asia
United Kingdom	64.511	2950039	242495	0.907	Europe
United States	318.523	17348075	9525067	0.915	America

```
[68]  df.drop('Canada', inplace=True)
      ✓ 0.0s
```

```
[69]  df
      ✓ 0.0s
```

	Population	GDP	Surface Area	HDI	Continent
France	63.951	2833687	640679	0.888	Europe
Germany	80.940	3874437	357114	0.916	Europe
Italy	60.665	2167744	301336	0.873	Europe
Japan	127.061	4602367	377930	0.891	Asia
United Kingdom	64.511	2950039	242495	0.907	Europe
United States	318.523	17348075	9525067	0.915	America

```
[70]  df
      ✓ 0.0s
```

	Population	GDP	Surface Area	HDI	Continent
France	63.951	2833687	640679	0.888	Europe
Germany	80.940	3874437	357114	0.916	Europe
Italy	60.665	2167744	301336	0.873	Europe
Japan	127.061	4602367	377930	0.891	Asia
United Kingdom	64.511	2950039	242495	0.907	Europe
United States	318.523	17348075	9525067	0.915	America

```
[71]  df.drop(columns=['Population', 'HDI'], inplace=True)
      ✓ 0.0s
```

```
[72]  df
      ✓ 0.0s
```

	GDP	Surface Area	Continent
France	2833687	640679	Europe
Germany	3874437	357114	Europe
Italy	2167744	301336	Europe
Japan	4602367	377930	Asia
United Kingdom	2950039	242495	Europe
United States	17348075	9525067	America

Data Frames: añadir una nueva columna

Puedo crear una nueva columna a partir de una serie. En caso de haber datos faltantes, Panda los rellena con NaN.

```
[54]
langs = pd.Series(
    ['French', 'German', 'Italian'],
    index=['France', 'Germany', 'Italy'],
    name='Language'
)
```
✓ 0.0s

```
[55]
langs
```
✓ 0.0s

```
France      French
Germany     German
Italy      Italian
Name: Language, dtype: object
```

```
[56]
df['Language'] = langs
```
✓ 0.0s

```
[57]
df
```
✓ 0.0s

	Population	GDP	Surface Area	HDI	Continent	Language
Canada	35.467	1785387	9984670	0.913	America	NaN
France	63.951	2833687	640679	0.888	Europe	French
Germany	80.940	3874437	357114	0.916	Europe	German
Italy	60.665	2167744	301336	0.873	Europe	Italian
Japan	127.061	4602367	377930	0.891	Asia	NaN
United Kingdom	64.511	2950039	242495	0.907	Europe	NaN
United States	318.523	17348075	9525067	0.915	America	NaN

Data Frames: renombrar una columna o índice

Con el comando `rename` puedo cambiar el nombre de una columna o un índice existente.

```
df.rename(
    columns={
        'HDI': 'Human Development Index',
        'TLD': 'Top Level Domain'
    }, index={
        'United States': 'USA',
        'United Kingdom': 'UK',
        'Argentina': 'AR'
    })
```

[60] ✓ 0.0s

...

	Population	GDP	Surface Area	Human Development Index	Continent	Language
Canada	35.467	1785387	9984670	0.913	America	English
France	63.951	2833687	640679	0.888	Europe	English
Germany	80.940	3874437	357114	0.916	Europe	English
Italy	60.665	2167744	301336	0.873	Europe	English
Japan	127.061	4602367	377930	0.891	Asia	English
UK	64.511	2950039	242495	0.907	Europe	English
USA	318.523	17348075	9525067	0.915	America	English

Data Frames: añadir nuevas filas

Puedo usar el comando `concat` para concatenar dos DF distintos.

```
df
```

[67] ✓ 0.0s

	Population	GDP	Surface Area	HDI	Continent
Canada	35.467	1785387	9984670	0.913	America
France	63.951	2833687	640679	0.888	Europe
Germany	80.940	3874437	357114	0.916	Europe
Italy	60.665	2167744	301336	0.873	Europe
Japan	127.061	4602367	377930	0.891	Asia
United Kingdom	64.511	2950039	242495	0.907	Europe
United States	318.523	17348075	9525067	0.915	America

```
df3 = pd.DataFrame({'Population' : 1409670, 'GDP': 35291000}, index=['China'])
df2 = pd.concat([df, df3])
df2
```

[77] ✓ 0.0s

	Population	GDP	Surface Area	HDI	Continent
Canada	35.467	1785387	9984670.0	0.913	America
France	63.951	2833687	640679.0	0.888	Europe
Germany	80.940	3874437	357114.0	0.916	Europe
Italy	60.665	2167744	301336.0	0.873	Europe
Japan	127.061	4602367	377930.0	0.891	Asia
United Kingdom	64.511	2950039	242495.0	0.907	Europe
United States	318.523	17348075	9525067.0	0.915	America
China	1409670.000	35291000	NaN	NaN	NaN

Data Frames: añadir nuevas filas

En caso de solo tener índices numéricos, los debo ignorar al hacer la concatenación.

```
dict = {'Name':['Martha', 'Tim', 'Rob', 'Georgia'],
        'Maths':[87, 91, 97, 95],
        'Science':[83, 99, 84, 76]
       }

df = pd.DataFrame(dict)

display(df)

df2 = {'Name': 'Amy', 'Maths': 89, 'Science': 93}
df = df._append(df2, ignore_index = True)

display(df)
```

✓ 0.0s

	Name	Maths	Science
0	Martha	87	83
1	Tim	91	99
2	Rob	97	84
3	Georgia	95	76

	Name	Maths	Science
0	Martha	87	83
1	Tim	91	99
2	Rob	97	84
3	Georgia	95	76
4	Amy	89	93

Data Frames: calcular una nueva columna

Puedo crear columnas nuevas a partir de columnas existentes.

```
df[['Population', 'GDP']]
```
[147] ✓ 0.0s

	Population	GDP
Canada	35.467	1785387
France	63.951	2833687
Germany	80.940	3874437
Italy	60.665	2167744
Japan	127.061	4602367
United Kingdom	64.511	2950039
United States	318.523	17348075

```
df['GDP'] / df['Population']
```
[148] ✓ 0.0s

```
Canada           50339.385908
France           44310.284437
Germany          47868.013343
Italy            35733.025633
Japan            36221.712406
United Kingdom   45729.239975
United States    54464.120330
dtype: float64
```

```
df['GDP Per Capita'] = df['GDP'] / df['Population']
```
✓ 0.0s

```
df
```
✓ 0.0s

	Population	GDP	Surface Area	HDI	Continent	GDP Per Capita
Canada	35.467	1785387	9984670	0.913	America	50339.385908
France	63.951	2833687	640679	0.888	Europe	44310.284437
Germany	80.940	3874437	357114	0.916	Europe	47868.013343
Italy	60.665	2167744	301336	0.873	Europe	35733.025633
Japan	127.061	4602367	377930	0.891	Asia	36221.712406
United Kingdom	64.511	2950039	242495	0.907	Europe	45729.239975
United States	318.523	17348075	9525067	0.915	America	54464.120330

Data Frames: obtención de datos

En ciencia de datos y Machine Learning es común que los datos se transfieran en un tipo de archivo desprovisto de formato (a diferencia, por ejemplo, de un archivo Excel) para mejorar la compatibilidad entre librerías y programas.

Un formato de archivo de este tipo es el de extensión CSV (comma separated values). Es parecido a un archivo .XLSX en el cual la información tabular está contenida en varias filas pero en una sola columna, donde se utiliza una coma (,) para significar la separación de datos (similar a los elementos en una lista de Python).

Por lo general, en la primer fila del archivo, se tiene una descripción del contenido de las columnas, lo que se conoce como header.

Data Frames: obtención de datos

- A la hora de importar se pueden definir varios parámetros para que el DF tenga el formato que deseamos.
- La primera fila del mismo se interpeta como la que contiene información de los encabezados, por lo que se debe ignorar si no los tuviere.
- También puedo establecer una columna como la que define los índices.
- Puedo establecer el delimitador de columnas (coma, punto y coma).
- Si quiero ignorar algunas filas o columnas.
- Establecer tipos de datos.

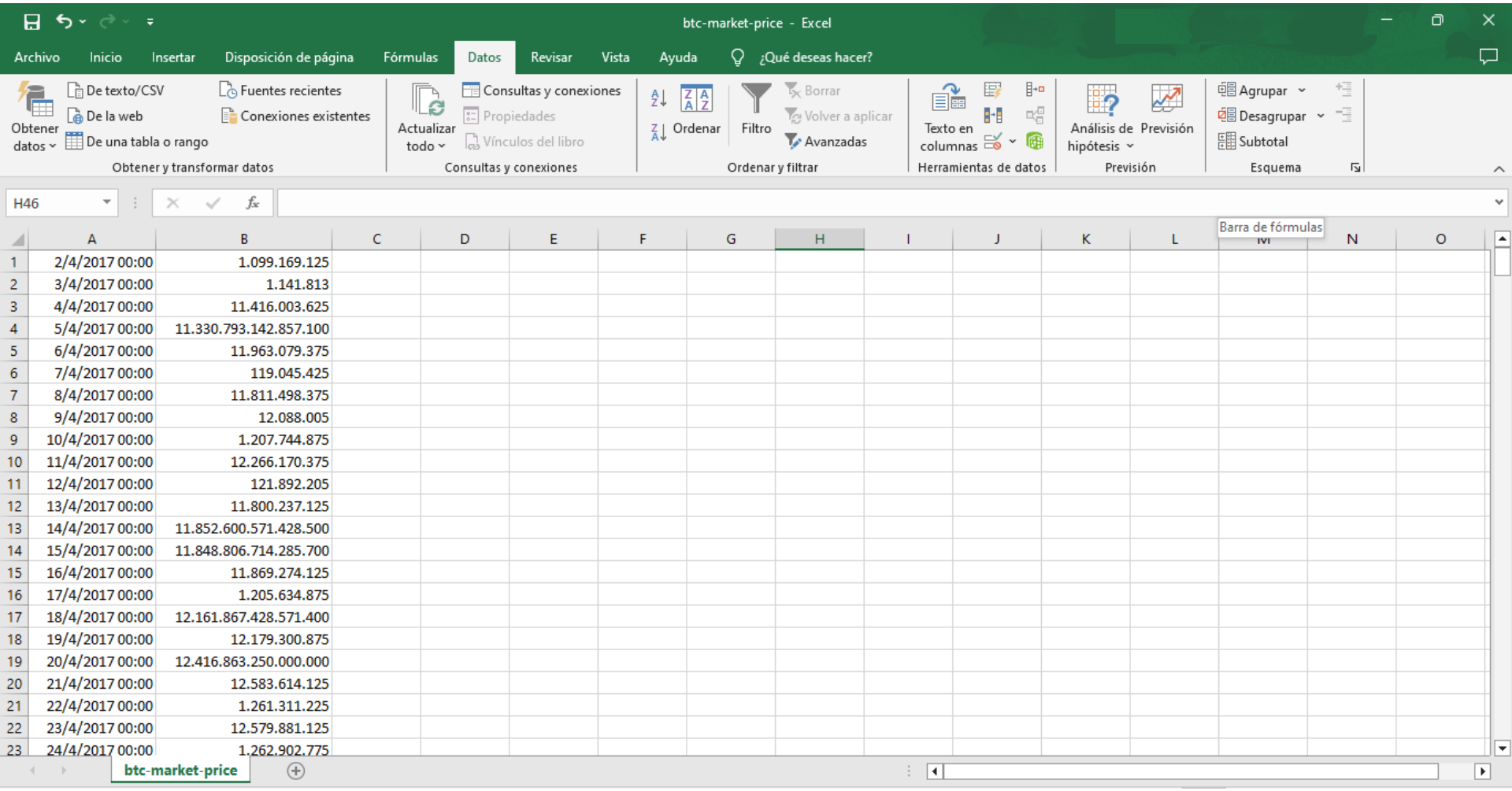
Data Frames: obtención de datos

pandas.read_csv

```
pandas.read_csv(filepath_or_buffer, *, sep=_NoDefault.no_default,
delimiter=None, header='infer', names=_NoDefault.no_default, index_col=None,
usecols=None, dtype=None, engine=None, converters=None, true_values=None,
false_values=None, skipinitialspace=False, skiprows=None, skipfooter=0,
nrows=None, na_values=None, keep_default_na=True, na_filter=True,
verbose=_NoDefault.no_default, skip_blank_lines=True, parse_dates=None,
infer_datetime_format=_NoDefault.no_default,
keep_date_col=_NoDefault.no_default, date_parser=_NoDefault.no_default,
date_format=None, dayfirst=False, cache_dates=True, iterator=False,
chunksize=None, compression='infer', thousands=None, decimal='.',
lineterminator=None, quotechar='"', quoting=0, doublequote=True,
escapechar=None, comment=None, encoding=None, encoding_errors='strict',
dialect=None, on_bad_lines='error', delim_whitespace=_NoDefault.no_default,
low_memory=True, memory_map=False, float_precision=None,
storage_options=None, dtype_backend=_NoDefault.no_default)        [source]
```

Read a comma-separated values (csv) file into DataFrame.

Data Frames: obtención de datos



	A	B
1	2/4/2017 00:00	1.099.169.125
2	3/4/2017 00:00	1.141.813
3	4/4/2017 00:00	11.416.003.625
4	5/4/2017 00:00	11.330.793.142.857.100
5	6/4/2017 00:00	11.963.079.375
6	7/4/2017 00:00	119.045.425
7	8/4/2017 00:00	11.811.498.375
8	9/4/2017 00:00	12.088.005
9	10/4/2017 00:00	1.207.744.875
10	11/4/2017 00:00	12.266.170.375
11	12/4/2017 00:00	121.892.205
12	13/4/2017 00:00	11.800.237.125
13	14/4/2017 00:00	11.852.600.571.428.500
14	15/4/2017 00:00	11.848.806.714.285.700
15	16/4/2017 00:00	11.869.274.125
16	17/4/2017 00:00	1.205.634.875
17	18/4/2017 00:00	12.161.867.428.571.400
18	19/4/2017 00:00	12.179.300.875
19	20/4/2017 00:00	12.416.863.250.000.000
20	21/4/2017 00:00	12.583.614.125
21	22/4/2017 00:00	1.261.311.225
22	23/4/2017 00:00	12.579.881.125
23	24/4/2017 00:00	1.262.902.775

Data Frames: obtención de datos

```
df = pd.read_csv(
    'data/btc-market-price.csv',
    header=None,
    names=['Timestamp', 'Price'],
    parse_dates=True
)
```

[30] ✓ 0.0s

```
df.head()
```

[31] ✓ 0.0s

...

	Timestamp	Price
0	2017-04-02 00:00:00	1099.169125
1	2017-04-03 00:00:00	1141.813000
2	2017-04-04 00:00:00	1141.600363
3	2017-04-05 00:00:00	1133.079314
4	2017-04-06 00:00:00	1196.307937