Introducción al objeto DataFrame

Ya hemo visto que el objeto Series generaliza el concepto de array 1-D. El objeto DataFrame generaliza el concepto de array 2-D (matriz) con la ventaja de que el objeto Dataframe ofrece la posibilidad indexar con nombres de filas y columnas y combinar distintos tipos de datos, es decir, permite representar tablas.

Inicialización del entorno.

Lo primero será importar el paquete Pandas con alias `pd`. Posteriormente visualizamos la versión usada de Pandas ya que es un dato importante para consultar la documentación. En el momento de editar este notebook la versión de pandas es: 1.4.3

Además para facilitar los ejercicios también importamos el paquete Numpy con el alias `np`.

In [1]:
```
import pandas as pd
import numpy as np
np.set_printoptions(floatmode='fixed', precision=3)
print('Pandas versión: ', pd.__version__)
```

```
Pandas versión:  1.4.3
```

Uso básico del objeto DataFrame.

Hay varias formas de crear un objeto DataFrame. La más básica es usar una lista de listas Python.

Ejercicio 01: Data la lista `[[1.0, -1.5], [2.3, 3.4]]` crear un objeto Series.

La salida debería ser algo parecido a lo siguiente:

```
     0    1
0  1.0 -1.5
1  2.3  3.4
dtype: float64
```

In [2]:
```
df = 0
#Pon tu código aquí.
#Sugerencia: usa el constructor pd.DataFrame(...)

#
print(df)
```

```
0
```

También podemos crear un objeto DataFrame desde un numpy.ndarray con la condición de que tenga una o dos dimensiones.

Ejercicio 02: Generar un objeto DataFrame usando un Numpy.ndarray.

La salida debería ser algo parecido a lo siguiente:

```
          0         1         2         3
0  0.548814  0.715189  0.602763  0.544883
1  0.423655  0.645894  0.437587  0.891773
2  0.963663  0.383442  0.791725  0.528895
```

In [3]:
```
gen = np.random.seed(0)#para poder reproducir.
a = np.random.rand(12).reshape(3,4)
df = 0
#Pon tu código aquí.
#Sugerencia: usa el constructor pd.DataFrame(...)

#
print(df)
```

```
0
```

Como se puede ver, el objeto DataFrame tiene dos atributos principales:

- Los datos de la tabla accesibles con DataFrame.values.
- Dos índices, uno para las filas y el otro para las columnas, ambos accesibles con DataFrame.axes.

Cuando se crea un objeto DataFrame si no se indica índice de fila y/o columna, este se genera de forma automática como un rango entero `[0, <num-filas>)` o `[0, <num-columnas>)`.

Ejercicio 03: Imprime los índices fila/columna asociados a un objeto Dataframe.

La salida debería ser algo parecido a lo siguiente:

```
[RangeIndex(start=0, stop=3, step=1), RangeIndex(start=0, stop=4,
step=1)]
```

In [4]:
```
gen = np.random.seed(0)#para poder reproducir.
df = pd.DataFrame(np.random.rand(12).reshape(3,4))

#Pon tu código aquí.
#Sugerencia: imprime el el atributo axes del dataFrame.

#
```

También podemos acceder acceder por separado a los dós índices:

- Para el índice de las filas usamos DataFrame.index.
- Para el Índice de las columnas usamos DataFrame.columns.

Ejercicio 04: Imprime los índices asociados a las filas y columnas de un objeto Dataframe.

La salida debería ser algo parecido a lo siguiente:

```
Ínidice de filas   : RangeIndex(start=0, stop=3, step=1)
Ínidice de columnas: RangeIndex(start=0, stop=4, step=1)
```

In [5]:
```
gen = np.random.seed(0)#para poder reproducir.
df = pd.DataFrame(np.random.rand(12).reshape(3,4))

print('Índice de filas   : ', end='')

#Pon tu código aquí.
```

```
#Sugerencia: imprime el el atributo index.

#
print('Índice de columnas: ', end='')

#Pon tu código aquí.
#Sugerencia: imprime el el atributo columns.

#
```

```
Índice de filas   : Índice de columnas:
```

Podemos indicar los índices de filas y columnas cuando creamos un objeto DataFrame usando los argumentos `index` y `columns` del constructor.

Ejercicio 05: Crea un dataframe usando un ndarray con forma (2,3) y usando cómo índice para las filas las etiquetas `['A', 'B']` y, para las columnas, las etiquetas `['a', 'b', 'c']`.

La salida debería ser algo parecido a lo siguiente:

```
          a         b         c
A  0.548814  0.715189  0.602763
B  0.544883  0.423655  0.645894
```

In [6]:
```
gen = np.random.seed(0)#para poder reproducir.
a = np.random.rand(6).reshape(2,3)
f_etiq = ['A', 'B']
c_etiq = ['a', 'b', 'c']
df = 0

#Pon tu código aquí.
#Crea el DataFrame a partir del ndarray a.
#Sugerencia: usa el constructor pd.DataFrame(...)

#
print(df)
```

```
0
```

Por último podemos generar un DataFrame usando un diccionario de diccionarios, es decir, cada valor del primer diccionario, es a su vez, otro diccionario que define una columna. La etiqueta en el primer diccionario sería la etiqueta de la columna, las etiquetas del diccionario que define una columna serían las etiquetas de las filas, por lo que deben ser las mismas usadas en todas las columnas.

Ejercicio 06: Crea un dataframe usando dos diccionarios que proporcionan las cotizaciones, el valor de capitalización y una categoría de varias empresas que cotizan en el IBEX.

La salida debería ser algo parecido a lo siguiente:

```
         Cotización  Capitalización Categoría
BANQIA        0.880            2769         A
VVBA          2.892           19777        AA
SANTAN        2.076           35521         A
CAJANOR       1.763           10779         B
```

In [7]:
```
cot_dict = {'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763}
cap_dict = {'BANQIA':2769, 'VVBA':19777, 'SANTAN':35521,
            'CAJANOR':10779}
cat_dict = {'BANQIA':'A', 'VVBA':'AA', 'SANTAN':'A',
```

```
            'CAJANOR':'B'}
ibex = 0
#Pon tu código aquí.
#Sugerencia: usa el constructor del objeto DataFrame indicando un
#diccionario con las etiquetas 'Cotización' y 'Capitalización' y valores
#los correspondientes diccionarios

#

print(ibex)
```

```
0
```

Observa como cada columna tiene un tipo de dato distinto.

Usando un objeto DataFrame como un diccionario.

Una forma conveniente de ver un objeto DataFrame es como una colección de objetos Series (cada columna sería una serie) que comparten un objeto índice común para las filas.

Utilizando operador `[]` podremos usar el objeto DataFrame como si fuera un diccionario obteniendo la serie correspondiente a la etiqueta del índice de columnas que hayamos usado.

Ejercicio 07: Dado el DataFrame 'ibex', obtén la serie correspondiente a la columna 'Cotización'.

La salida debería ser algo parecido a lo siguiente:

```
BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
Name: Cotización, dtype: float64
```

In [8]:
```
cot_dict = {'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763}
cap_dict = {'BANQIA':2769, 'VVBA':19777, 'SANTAN':35521,
            'CAJANOR':10779}
cat_dict = {'BANQIA':'A', 'VVBA':'AA', 'SANTAN':'A',
            'CAJANOR':'B'}
ibex = pd.DataFrame({'Cotización':cot_dict, 'Capitalización':cap_dict,
                     'Categoría':cat_dict})
s = 0
#Pon tu código aquí.
#Sugerencia: usa la etiqueta 'Cotización'

#

print(s)
```

```
0
```

Dado que al aplicar el operador `[]` a un objeto DataFrame, obtenemos el objeto Series correspondiente a la columna indicada con la etiqueta, a su vez, el objeto Series que se obtiene puede ser indexado con el operador `[]` indicando la etiqueta de la fila deseada. Esta es una forma (veremos más) de acceder a los valores almacenados en el DataFrame.

Esta forma de acceder al DataFrame se denomina, método encadenado ("*chaining*) tiene el inconveniente de que la serie obtenida puede ser tanto una "vista" como una copia de la columna indicada (esto dependerá de los datos subyacentes almacenados e incluso del entorno de ejecución), y por lo tanto, no es

un método recomendable para modificar la serie (que podría ser una copia) con el objetivo de modificar el correspondiente valor del objeto DataFrame, además de que sería una operación poco eficiente (se recomienda leer la información sobre los avisos *returning-a-view-versus-a-copy* y *setting-with-copy*).

Ejercicio 08: Dado el DataFrame 'ibex', imprime usando el operador `'[]'` el valor de capitalización asociado a la empresa 'SANTAN'.

La salida debería ser algo parecido a lo siguiente:

```
DataFrame:
          Cotización  Capitalización Categoría
BANQIA        0.880            2769         A
VVBA          2.892           19777        AA
SANTAN        2.076           35521         A
CAJANOR       1.763           10779         B

Valor de capitalización de 'SANTAN': 35521
```

In [9]:
```
cot_dict = {'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763}
cap_dict = {'BANQIA':2769, 'VVBA':19777, 'SANTAN':35521,
            'CAJANOR':10779}
cat_dict = {'BANQIA':'A', 'VVBA':'AA', 'SANTAN':'A',
            'CAJANOR':'B'}
ibex = pd.DataFrame({'Cotización':cot_dict, 'Capitalización':cap_dict,
                     'Categoría':cat_dict})
print('DataFrame:\n', ibex)

cap = 0

#Pon tu código aquí.
#Sugerencia: usa la etiqueta 'Captialización' para accecer a la serie y la etiqueta
#'SANTAN' para acceder al valor deseado.

#
print("\nValor de capitalización de 'SANTAN': {}".format(cap))
```

```
DataFrame:
          Cotización  Capitalización Categoría
BANQIA        0.880            2769         A
VVBA          2.892           19777        AA
SANTAN        2.076           35521         A
CAJANOR       1.763           10779         B

Valor de capitalización de 'SANTAN': 0
```

Ejercicio 09: Dado el DataFrame 'ibex', imprime usando el operador `'[]'` el valor de capitalización asociado a la empresa 'SANTAN' y modifícalo aumentando el valor un 10%.

La salida debería ser algo parecido a lo siguiente:

```
DataFrame original:
          Cotización  Capitalización Categoría
BANQIA        0.880            2769         A
VVBA          2.892           19777        AA
SANTAN        2.076           35521         A
CAJANOR       1.763           10779         B

Valor de capitalización de 'SANTAN' 35521
```

```
Valor de capitalización actualizado 'SANTAN' 39073.100

DataFrame modificado:
          Cotización  Capitalización Categoría
BANQIA        0.880          2769.0         A
VVBA          2.892         19777.0        AA
SANTAN        2.076         39073.1         A
CAJANOR       1.763         10779.0         B

/tmp/ipykernel_6584/2270280984.py:26: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  ibex['Capitalización']['SANTAN'] = cap
```

In [10]:

```
cot_dict = {'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763}
cap_dict = {'BANQIA':2769, 'VVBA':19777, 'SANTAN':35521,
            'CAJANOR':10779}
cat_dict = {'BANQIA':'A', 'VVBA':'AA', 'SANTAN':'A',
            'CAJANOR':'B'}
ibex = pd.DataFrame({'Cotización':cot_dict, 'Capitalización':cap_dict,
                     'Categoría':cat_dict})
print('DataFrame original:\n', ibex)

cap = 0

#Pon tu código aquí.
#Sugerencia: usa la etiqueta 'Captialización' para accecer a la serie y la etiqueta
#'SANTAN' para acceder al valor deseado.

#
print("\nValor de capitalización de 'SANTAN' {}".format(cap))

cap *= 1.1;
print("\nValor de capitalización actualizado 'SANTAN' {:.3f}".format(cap))

#Pon tu código aquí.
#Sugerencia: usa la etiqueta 'Captialización' para accecer a la serie y la etiqueta
#'SANTAN' para acceder al valor deseado.

#

print('\nDataFrame modificado:\n', ibex)
```

```
DataFrame original:
          Cotización  Capitalización Categoría
BANQIA        0.880            2769         A
VVBA          2.892           19777        AA
SANTAN        2.076           35521         A
CAJANOR       1.763           10779         B

Valor de capitalización de 'SANTAN' 0

Valor de capitalización actualizado 'SANTAN' 0.000

DataFrame modificado:
          Cotización  Capitalización Categoría
BANQIA        0.880            2769         A
VVBA          2.892           19777        AA
```

```
SANTAN        2.076          35521         A
CAJANOR       1.763          10779         B
```

Usando un objeto DataFrame como un numpy.ndarray

Como hemos visto una objeto DataFrame representa una tabla. Es posible obtener una vista de la tabla como un mumpy.ndarray de dos dimensiones. Para ello podemos utilizar el atributo `values` de objeto DataFrame, pero se recomienda usar el método `DataFrame.to_numpy()` para acceder a esta vista.

Hay que tener en cuenta que dado que los datos en el objeto DataFrame en general no son homogéneos, el tipo de elemento asociado al ndarray devuelto será el "mínimo común denominador" numpy.dtype que representen a todos. Por ejemplo si los tipos son todos numéricos, mezclando enteros de 16 bits, flotantes de 32 bits y flotantes de 64 bits, el dtype que puede representarlos a todos será np.float64.

En general, si mezclamos tipos numéricos con otros, el tipo común será `np.object`. Para más información sobre las reglas aplicadas para la promoción de tipos (*casting*) se recomienda leer esta referencia.

Ejercicio 10: Dado el DataFrame 'ibex', obtén la versión como numpy.ndarray y visualiza el tipo asociado al mismo. Intenta inferir tú mismo el tipo antes de realizar la prueba y compara con el resultado obtenido.

La salida debería ser algo parecido a lo siguiente:

```
Array values:
 [[0.88 2769 'A']
 [2.892 19777 'AA']
 [2.076 35521 'A']
 [1.763 10779 'B']]

Array dtype :  ¿?
```

In [11]:
```
cot_dict = {'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763}
cap_dict = {'BANQIA':2769, 'VVBA':19777, 'SANTAN':35521,
            'CAJANOR':10779}
cat_dict = {'BANQIA':'A', 'VVBA':'AA', 'SANTAN':'A',
            'CAJANOR':'B'}
ibex = pd.DataFrame({'Cotización':cot_dict, 'Capitalización':cap_dict,
                     'Categoría':cat_dict})
print('DataFrame:')
print(ibex)

array = np.zeros(0)
#Pon tu código aquí.
#Sugerencia: usa el método to_numpy()

#
print('')
print('Array values:\n', array)
print('')
print('\nArray dtype : ', array.dtype)
```

```
DataFrame:
         Cotización  Capitalización Categoría
BANQIA        0.880            2769         A
VVBA          2.892           19777        AA
SANTAN        2.076           35521         A
CAJANOR       1.763           10779         B
```

```
Array values:
 []


Array dtype :  float64
```

Ejercicio 11: Dado el DataFrame 'ibex', obtén la versión como numpy.ndarray y visualiza la segunda y tercera filas.

La salida debería ser algo parecido a lo siguiente:

```
Array values:
 [[0.88 2769 'A']
 [2.892 19777 'AA']
 [2.076 35521 'A']
 [1.763 10779 'B']]

Segunda y tercera filas:
[[2.892 19777 'AA']
 [2.076 35521 'A']]
```

In [12]:
```
cot_dict = {'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763}
cap_dict = {'BANQIA':2769, 'VVBA':19777, 'SANTAN':35521,
            'CAJANOR':10779}
cat_dict = {'BANQIA':'A', 'VVBA':'AA', 'SANTAN':'A',
            'CAJANOR':'B'}
ibex = pd.DataFrame({'Cotización':cot_dict, 'Capitalización':cap_dict,
                     'Categoría':cat_dict})
array = np.zeros(0)
#Pon tu código aquí.
#Sugerencia: usa el método to_numpy()

#
print('Array values:\n', array)
print('\nSegunda y tercera filas:')
#Pon tu código aquí.
#Sugerencia: usa el rango inicio:fin:paso corresondiente.

#
```

```
Array values:
 []

Segunda y tercera filas:
```

Ejercicio 12: Dado el DataFrame 'ibex', obtén la versión como numpy.ndarray y visualiza la segundo columna.

La salida debería ser algo parecido a lo siguiente:

```
Array values:
 [[0.88 2769 'A']
 [2.892 19777 'AA']
 [2.076 35521 'A']
 [1.763 10779 'B']]

Segunda columna:
[2769 19777 35521 10779]
```

In [13]:
```
cot_dict = {'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763}
cap_dict = {'BANQIA':2769, 'VVBA':19777, 'SANTAN':35521,
            'CAJANOR':10779}
cat_dict = {'BANQIA':'A', 'VVBA':'AA', 'SANTAN':'A',
            'CAJANOR':'B'}
ibex = pd.DataFrame({'Cotización':cot_dict, 'Capitalización':cap_dict,
                     'Categoría':cat_dict})
array = np.zeros(0)
#Pon tu código aquí.
#Sugerencia: usa el método to_numpy()

#
print('Array values:\n', array)
print('\nSegunda columna:')
#Pon tu código aquí.
#Sugerencia: usa los rangos adecuados.

#
```

```
Array values:
 []

Segunda columna:
```

El ndarray devuelto por el método `to_numpy()` puede ser una vista del array subyacente usado por Pandas, o una copia. En general se devolverá una copia cuando los datos almacenados en el DataFrame no sean homogéneos y se necesite aplicar coerción de tipos (*casting*).

Este es un detalle es importante como ya vimos, porque si se obtiene una vista, modificar la vista implica modificar el DataFrame, mientras que si se obtiene una copia, modificar la copia no modifica el DataFrame.

Para resolver la ambigüedad podemos usar el parámetro `copy=True` en el método `to_numpy()` para asegurarnos que obtenemos una copia en cualquier caso.

Ejercicio 13: Dado el DataFrame 'ibex', comprobar que la versión como numpy.ndarray es una copia, y por lo tanto al modificarla, no se modificará los datos del objeto DataFrame original.

La salida debería ser algo parecido a lo siguiente:

```
DataFrame:
          Cotización  Capitalización Categoría
BANQIA         0.880            2769         A
VVBA           2.892           19777        AA
SANTAN         2.076           35521         A
CAJANOR        1.763           10779         B

Array:
 [[0.88 2769 'A']
 [2.892 19777 'AA']
 [2.076 35521 'A']
 [1.763 10779 'B']]

Array modificado:
 [[1.0 2769 'A']
 [2.892 19777 'AA']
 [2.076 35521 'A']
 [1.763 10779 'B']]

DataFrame:
```

```
           Cotización  Capitalización Categoría
BANKIA          0.880            2769         A
BBVA            2.892           19777        AA
SANTANDER       2.076           35521         A
CAIXABANK       1.763           10779         B
```

In [14]:
```
cot_dict = {'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763}
cap_dict = {'BANQIA':2769, 'VVBA':19777, 'SANTAN':35521,
            'CAJANOR':10779}
cat_dict = {'BANQIA':'A', 'VVBA':'AA', 'SANTAN':'A',
            'CAJANOR':'B'}
ibex = pd.DataFrame({'Cotización':cot_dict, 'Capitalización':cap_dict,
                     'Categoría':cat_dict})
print('DataFrame:\n', ibex)

array = np.zeros(0)
#Pon tu código aquí.
#Sugerencia: usa el método to_numpy()

#
print('\nArray:\n', array)

#Modifica en el ndarray 'array' el valor de cotización correspondiente
#a la empresa BANQIA al valor 1.0
#Pon tu código aquí.
#Sugerencia: indexa el ndarray convenientemente.

#
print('\nArray modificado:\n', array)
print('\nDataFrame:\n', ibex)
```

```
DataFrame:
          Cotización  Capitalización Categoría
BANQIA         0.880            2769         A
VVBA           2.892           19777        AA
SANTAN         2.076           35521         A
CAJANOR        1.763           10779         B

Array:
 []

Array modificado:
 []

DataFrame:
          Cotización  Capitalización Categoría
BANQIA         0.880            2769         A
VVBA           2.892           19777        AA
SANTAN         2.076           35521         A
CAJANOR        1.763           10779         B
```

Otros atributos de un objeto DataFrame.

Además de los atributos `axes` , `index` y `columns` ya vistos, el objeto DataFrame tiene otros atributos interesantes.

Ejercicio 14: Dado el DataFrame 'ibex', queremos obtener su forma.

La salida debería ser algo parecido a lo siguiente:

```
DataFrame shape: (4, 3)
```

In [15]:
```
cot_dict = {'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763}
cap_dict = {'BANQIA':2769, 'VVBA':19777, 'SANTAN':35521,
            'CAJANOR':10779}
cat_dict = {'BANQIA':'A', 'VVBA':'AA', 'SANTAN':'A',
            'CAJANOR':'B'}
ibex = pd.DataFrame({'Cotización':cot_dict, 'Capitalización':cap_dict,
                     'Categoría':cat_dict})
shape = 0
#Pon tu código aquí.
#Sugerencia: usa el atributo shape.

#
print('DataFrame shape:', shape)
```

```
DataFrame shape: 0
```

Ejercicio 15: Dado el DataFrame 'ibex', queremo saber cuántos valores almacena.

La salida debería ser algo parecido a lo siguiente:

```
DataFrame size: 12
```

In [16]:
```
cot_dict = {'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763}
cap_dict = {'BANQIA':2769, 'VVBA':19777, 'SANTAN':35521,
            'CAJANOR':10779}
cat_dict = {'BANQIA':'A', 'VVBA':'AA', 'SANTAN':'A',
            'CAJANOR':'B'}
ibex = pd.DataFrame({'Cotización':cot_dict, 'Capitalización':cap_dict,
                     'Categoría':cat_dict})
size = 0
#Pon tu código aquí.
#Sugerencia: usa el atributo size.

#
print('DataFrame size:', size)
```

```
DataFrame size: 0
```

Ejercicio 16: Dado el DataFrame 'ibex', queremo saber los tipos asociados a cada una de las columnas.

La salida debería ser algo parecido a lo siguiente:

```
Tipos de las columnas:
 Cotización        float64
Capitalización      int64
Categoría          object
dtype: object
```

In [17]:
```
cot_dict = {'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763}
cap_dict = {'BANQIA':2769, 'VVBA':19777, 'SANTAN':35521,
            'CAJANOR':10779}
cat_dict = {'BANQIA':'A', 'VVBA':'AA', 'SANTAN':'A',
            'CAJANOR':'B'}
ibex = pd.DataFrame({'Cotización':cot_dict, 'Capitalización':cap_dict,
                     'Categoría':cat_dict})
tipos = 0

#Pon tu código aquí.
#Sugerencia: usa el atributo dtypes.
```

```
#
print('Tipos de las columnas:\n', tipos)
```

```
Tipos de las columnas:
 0
```

In [ ]: