Indexación jerárquica

Hasta ahora hemos visto que podemos manipular datos que tienen una dimensión (Series) o dos dimensiones (DataFrame).

En muchas situaciones los datos asociados a estas dimensiones lo pueden estar en más de un nivel. Por ejemplo en la serie usada como ejemplo en anteriores cuadernos 'Cotizaciones', hasta ahora hemos asociado una cotización para cada empresa (un nivel), pero no es difícil imaginar que podemos tener varias cotizaciones para cada empresa, una para cada mes del año. Esto hace que aumentemos un nivel el índice [empresa, mes].

La indexación jerárquica nos permite representar datos multidimensionales y combinada con las funciones de agrupación hace a Pandas una poderosa herramienta para el análisis de datos.

Este cuaderno se basa en la documentación oficial accesible en esta referencia.

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 [11]:
```
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
```

El objeto MultiIndex.

Pandas utiliza el objeto MultiIndex para representar índices con mas de un nivel.

Ejercicio 01: Se requiere crear un objeto MultiIndex a partir de una descripción de los niveles usando una lista de tuplas. Cada elemento de la tupla indica un nivel.

La salida obtenida debería ser algo parecido a lo siguiente:

```
El índice creado es:
 MultiIndex([( 'BANQIA', 'ENE'),
             ( 'BANQIA', 'FEB'),
             (   'VVBA', 'ENE'),
             (   'VVBA', 'FEB'),
             ( 'SANTAN', 'ENE'),
```

```
                ( 'SANTAN', 'FEB'),
                ('CAJANOR', 'ENE'),
                ('CAJANOR', 'FEB')],
               )
```

In [12]:
```
#Etiquetas del MultiIndex.
idx_t = [('BANQIA'   , 'ENE'), ('BANQIA'   , 'FEB'),
         ('VVBA'     , 'ENE'), ('VVBA'     , 'FEB'),
         ('SANTAN', 'ENE'), ('SANTAN', 'FEB'),
         ('CAJANOR', 'ENE'), ('CAJANOR', 'FEB')]

idx = 0

#Pon tu código aquí.
#Sugerencia: utiliza el método from_tuples() de la clase MultiIndex.

#

print('El índice creado es:\n', idx)
```

```
El índice creado es:
 0
```

Ejercicio 02: Se requiere crear un objeto MultiIndex a partir de una descripción de los niveles usando el producto escalar de dos listas de etiquetas.

La salida obtenida debería ser algo parecido a lo siguiente:

```
El índice creado es:
 MultiIndex([( 'BANQIA', 'ENE'),
             ( 'BANQIA', 'FEB'),
             (   'VVBA', 'ENE'),
             (   'VVBA', 'FEB'),
             ( 'SANTAN', 'ENE'),
             ( 'SANTAN', 'FEB'),
             ('CAJANOR', 'ENE'),
             ('CAJANOR', 'FEB')],
            )
```

In [13]:
```
#Etiquetas del MultiIndex.
nivel1 = ['BANQIA', 'VVBA','SANTAN', 'CAJANOR']
nivel2 = ['ENE', 'FEB']
idx = 0
#Pon tu código aquí.
#Sugerencia: crea el objeto MultiIndex utilizando el método
#from_produc() con las dos listas de etiquetas nivel1 y nivel2.

#

print('El índice creado es:\n', idx)
```

```
El índice creado es:
 0
```

Ejercicio 03: Tenemos los datos de cotización de varias empresas en distintos meses. Crear una serie con un índice de dos niveles que represente estos datos.

La salida obtenida debería ser algo parecido a lo siguiente:

```
Serie:
 BANQIA   ENE    0.880
```

```
            FEB    0.900
    VVBA    ENE    2.892
            FEB    2.710
    SANTAN  ENE    2.076
            FEB    2.100
    CAJANOR ENE    1.763
            FEB    1.800
    dtype: float64
```

In [14]:
```python
def cotizaciones():
    """Crear una serie para los ejercicios."""
    #Etiquetas del MultiIndex.
    nivel1 = ['BANQIA', 'VVBA','SANTAN', 'CAJANOR']
    nivel2 = ['ENE', 'FEB']
    data = [ 0.88, 0.9, 2.892, 2.71, 2.076, 2.1, 1.763, 1.8 ]
    idx = None

    #Pon tu código aquí.
    #Sugerencia: crea el objeto MultiIndex utilizando el método
    #   from_produc() con las dos listas de etiquetas nivel1 y nivel2.

    #

    s = None

    #Pon tu código aquí.
    #Sugerencia: crea el objeto Series s indicando con el argumento index
    #   el índice que hemos creado en el paso anterior.
    #Observa que los datos en la lista 'data' se corresponden con las parejas
    #   del índice creado.

    #
    return s

s = cotizaciones()
print('Serie:\n', s)
```

```
Serie:
 None
```

En el resultado del ejemplo anterior, podemos observar como las dos primeras columnas muestran la estructura jerárquica del índice. Los huecos en el índice se deben interpretar como el valor anterior del mismo nivel repetido.

Ejercicio 03b: Tenemos los datos de varia empresas (filas) con los valores de cotización y capitalización en distintos meses (columnas). Crear un DataFrame con un multiindex para las columnas que represente estos datos.

La salida obtenida debería ser algo parecido a lo siguiente:

```
    Dataframe ejemplo:
             Cotización       Capitalización
                   ENE   FEB            ENE      FEB
    BANQIA       0.880  0.90         2769.0   2569.0
    VVBA         2.892  2.71        19777.0  20000.0
    SANTAN       2.076  2.10        35521.0  34321.0
    CAJANOR      1.763  1.80        10779.0  11079.0
```

In [15]:
```python
def ibex_df ():
```

```
    """Crear un DataFrame para los ejemplos."""
    etiquetas_filas = ['BANQIA', 'VVBA', 'SANTAN', 'CAJANOR']
    etiquetas_columnas_1 = ['Cotización', 'Capitalización']
    etiquetas_columnas_2 = ['ENE', 'FEB']

    idx_filas = None
    #Pon tu código aquí.
    #Sugerencia: aquí creamos un índice simple con las etiquetas de
    #filas.

    #

    idx_columnas = None
    #Pon tu código aquí.
    #Sugerencia: aquí creamos el multi índice de columnas usando el
    #método from_product.

    #

    datos = [[ 0.88,  0.9,  2769.0,  2569.0],
             [2.892, 2.71, 19777.0, 20000.0],
             [2.076,  2.1, 35521.0, 34321.0],
             [1.763,  1.8, 10779.0, 11079.0]]

    df = None
    #Pon tu código aquí.
    #Sugerencia: crea el objeto dataframe usando las cotizaciones y
    #   los índices de filas y columnas creados.

    #

    return df

ibex = ibex_df()
print("Dataframe ejemplo:\n", ibex)
```

```
Dataframe ejemplo:
 None
```

Indexación jerárquica en objetos Series.

Cuando utilizamos un objeto MultiIndex para indexar un objeto Series, podemos seleccionar datos de la serie con el atributo `.loc` indicando con una lista las etiquetas del índice jerárquico requeridas. El primer elemento de la lista sería la etiqueta del mayor nivel, el segundo elemento, la etiqueta del segundo nivel y así sucesivamente.

Recuerda que también podemos especificar rangos con la notación `etiqueta-inicio:etiqueta-fin` para un nivel y que el rango `:` significa todos los valores de ese nivel.

Ejercicio 04: Dada una serie con cotizaciones para varias empresas, queremos obtener la cotizaciónn de la empres `'VVBA'` en el mes `'FEB'`.

La salida obtenida debería ser algo parecido a lo siguiente:

```
   Cotización de 'VVBA' en 'FEB' es 2.71
```

In [16]:
```
s = cotizaciones()

v = 0
```

```
#Pon tu código aquí.
#todos los elementos del primer nivel (':') con etiqueta del
# del multiindex usando una lista. Respeta el orden jerarquico.

#

print(f"Cotización de 'VVBA' en 'FEB' es {v}")
```

```
Cotización de 'VVBA' en 'FEB' es 0
```

Ejercicio 04b: Dada una serie con cotizaciones para varias empresas, queremos obtener la subserie con las cotizaciones de todas las empresas en el mes `'ENE'`.

La salida obtenida debería ser algo parecido a lo siguiente:

```
Cotizaciones en el mes 'ENE':
 BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64
```

In [17]:
```
s = cotizaciones()

s_ene = None
#Pon tu código aquí.
#Sugerencia: indexa la serie con el método loc, indicando que queremos
#   todos los elementos del primer nivel del multiindex (':') y con
#   etiqueta del segundo nivel 'ENE'.

#

print("Cotizaciones en el mes 'ENE':\n", s_ene)
```

```
Cotizaciones en el mes 'ENE':
 None
```

Ejercicio 05: Dada una serie con cotizaciones para varias empresas, queremos obtener la subserie con todas las cotizaciones de la empresa `'SANTAN'`.

La salida obtenida debería ser algo parecido a lo siguiente:

```
Cotizaciones de la empresa 'SANTAN':
 ENE    2.076
FEB    2.100
dtype: float64
```

In [18]:
```
s = cotizaciones()

s_santan = None
#Pon tu código aquí.
#Sugerencia: indexa la serie con el método loc, indicando que queremos
#   todos los elementos del segundo nivel del multiindex (':') y con
#   etiqueta de primer nivel 'SANTAN'.

#

print("Cotizaciones de la empresa 'SANTAN':\n", s_santan)
```

```
Cotizaciones de la empresa 'SANTAN':
```

```
None
```

Indexación jerárquica de objetos DataFrame.

Como es de esperarar, los objetos DataFrame también admiten indexación jerárquica para obtener subconjuntos de datos. Utilizaremos el atributo `.loc` indicando el índice de fila y de columna. Si alguno de estos índices es un objeto MultiIndex (este es el caso), para indicar un valor multiindex se utilizará una *tupla* con la notación `(primer-nivel, segundo-nivel, ...)`.

Ejercicio 06: Dados los datos de cotización y capitalización de varias empresas, queremos obtener el valor de `'Capitalización'` de la empresa `'CAJANOR'` para el mes `'ENE'`.

La salida obtenida debería ser algo parecido a lo siguiente:

```
Valor de Capitalización en Enero para la empresa CAJANOR: 10779.0
```

In [19]:
```
ibex = ibex_df()

v = 0
#Pon tu código aquí.
#Sugerencia: indexa con el atributo .loc usando las etiquetas especificadas.
#   Recuerda que el índice de columnas es un multiindex y debes usar una.
#   tupla.

#

print(f'Valor de Capitalización en Enero para la empresa CAJANOR: {v}')
```

```
Valor de Capitalización en Enero para la empresa CAJANOR: 0
```

Ejercicio 07: Dados los datos de cotización y capitalización de varias empresas, queremos visualizar la columna con los valores de `Capitalización`.

La salida obtenida debería ser algo parecido a lo siguiente:

```
Capitalización:
              ENE      FEB
BANQIA     2769.0   2569.0
VVBA      19777.0  20000.0
SANTAN    35521.0  34321.0
CAJANOR   10779.0  11079.0
```

In [20]:
```
ibex = ibex_df()

capitalizaciones = None
#Pon tu código aquí.
#Sugerencia: utiliza el método .loc para seleccionar los valores
#   de la columna 'Capitalización'. Recuerda que queremos todos los
#   valores de fila, usa para ello el rango ':'

#

print('Capitalización:\n', capitalizaciones)
```

```
Capitalización:
 None
```

Ejercicio 08: Dados los datos de cotización y capitalización de varias empresas, obtener todos los valores asociados a la empresa `'BANQIA'`.

La salida obtenida debería ser algo parecido a lo siguiente:

```
BANQIA:
 Cotización      ENE       0.88
                FEB       0.90
Capitalización  ENE    2769.00
                FEB    2569.00
Name: BANQIA, dtype: float64
```

In [21]:
```
ibex = ibex_df()

Banqia_data = None
#Pon tu código aquí.
#Sugerencia: utiliza el método .loc para seleccionar los valores
#   asociados a la empresa 'BANQIA'. Como queremos todos los datos
#   (columna) utiliza el rango ':'.

#

print('BANQIA:\n', Banqia_data)
```

```
BANQIA:
 None
```

Ejercicio 09: Dados los datos de cotización y capitalización de varias empresas, seleccionar los valores de cotización para la empresa `'SANTAN'`.

La salida obtenida debería ser algo parecido a lo siguiente:

```
Cotizaciones de 'SANTAN':
 Cotización  ENE    2.076
            FEB    2.100
Name: SANTAN, dtype: float64
```

In [22]:
```
ibex = ibex_df()

data = 0.0

#Pon tu código aquí.
#Sugerencia: utiliza el método .loc para seleccionar el valor,
#   debes utilizar valores para los dos ejes. Como el eje de columnas
#   es un multiindex, usa una tupla con dos valores, y como queremos todos
#   los valores del segundo nivel ('ENE' y 'FEB') usa una lista
#   para indicarlos.

#

print("Cotizaciones de 'SANTAN': \n", data)
```

```
Cotizaciones de 'SANTAN':
 0.0
```

Uso de *ufuncs* con MultiIndex.

Como se puede esperar, podemos utilizar *ufuncs* a las subseries obtenidas al indexar.

Ejercicio 10: Dada una serie con cotizaciones para varias empresas para los meses 'ENE' y 'FEB', queremos obtener una serie con el valor de cotización máximo de cada empresa.

La salida obtenida debería ser algo parecido a lo siguiente:

```
Valores máximos:
 BANQIA     0.900
VVBA       2.892
SANTAN     2.100
CAJANOR    1.800
dtype: float64
```

In [23]:
```
s = cotizaciones()

s_max = None
#Pon tu código aquí.
#Sugerencia: usa la ufunc numpy.maximum con las dos subseries que
#   se obtienen para las etiquetas de segundo nivel 'ENE' y 'FEB'

#

print("Valores máximos:\n", s_max)
```

```
Valores máximos:
 None
```

Agregación de datos con MultiIndex.

Ya hemos visto que Pandas proporciona funciones para obtener datos agregados, por ejemplo, mean(), min(), max(), sum().

Para obtener datos agregados para un nivel del multiindex usaremos el método `groupby` con los parámetros `axis` para indicar el eje ('rows' o 'columns') y el nivel en el eje con `level`.

Ejercicio 11: Dado el data frame `df` se desea agregar los valores máximos del nivel `'F1'` del multiindex de las filas.

La salida obtenida debería ser algo parecido a lo siguiente:

```
DataFrame:
 C1     P     Q
C2     p  q  p  q
F1 F2
A  a   6  4  0  5
   b   7  8  8  0
B  a   4  5  6  8
   b   3  7  6  9

Valores máximos:
 C1  P     Q
C2  p  q  p  q
F1
A   7  8  8  5
B   4  7  6  9
```

In [24]:
```
df = pd.DataFrame(np.random.randint(0, 10, (4,4)),
                  index=pd.MultiIndex.from_product([list('AB'), list('ab')], names=['F1'
                  columns=pd.MultiIndex.from_product([list('PQ'), list('pq')], names=['C

print('DataFrame:\n',df)

valores_max = None #Valor máximo agregado para L1 de las filas.
#Pon tu código aquí.
#Sugerencia: Usa el argumento 'axis' y 'level' para agrupar.
#Sugerencia: usa el método de agregación max() para obtener valores máximos.

#
print('\nValores máximos:\n', valores_max)
```

```
DataFrame:
 C1     P     Q
C2     p  q  p  q
F1 F2
A  a   5  1  4  7
   b   2  9  5  5
B  a   6  2  1  9
   b   0  6  4  1

Valores máximos:
 None
```

Ejercicio 12: Dado el dataframe `df` se desea agregar los valores máximos del nivel 'C1' del multiindex de las columnas.

La salida obtenida debería ser algo parecido a lo siguiente:

```
DataFrame:
 C1     P     Q
C2     p  q  p  q
F1 F2
A  a   4  9  7  9
   b   5  4  2  6
B  a   7  8  9  5
   b   2  7  0  7

Valores máximos:
 C1     P  Q
F1 F2
A  a   9  9
   b   5  6
B  a   8  9
   b   7  7
```

In [25]:
```
df = pd.DataFrame(np.random.randint(0, 10, (4,4)),
                  index=pd.MultiIndex.from_product([list('AB'), list('ab')], names=['F1'
                  columns=pd.MultiIndex.from_product([list('PQ'), list('pq')], names=['C

print('DataFrame:\n',df)

valores_max = None #Valor máximo agregado para L1 de las columnas.
#Pon tu código aquí.
#Sugerencia: Usa el argumentos 'axis' y 'level' del la función max().

#
print('\nValores máximos:\n', valores_max)
```

```
DataFrame:
```

```
 C1     P     Q
C2     p  q  p  q
F1 F2
A  a   1  7  4  0
   b   9  2  6  4
B  a   3  7  5  5
   b   7  8  1  3

Valores máximos:
 None
```

Ordenación de índices no ordenados.

En algunas circunstancias las etiquetas del índice no están en orden (numérico o lexicográfico). Esto se puede corregir con las funciones `Series.sort_index()` y `DataFrame.sort_index()`.

Con un objeto DataFrame usaremos el argumento `axis` para indicar el índice a ordenar (fila o columnas), y si el índice es un objeto multiindex, con el argumento `level` indicaremos el nivel a ordenar.

Ejercicio 11: A partir de un dataframe `df` queremos obtener una versión con el índice de filas ordenado.

La salida obtenida debería ser algo parecido a lo siguiente:

```
Dataframe original:
    C  A  E  B  D
A  6  0  4  3  6
C  7  9  4  3  0
E  6  7  8  6  7
D  3  2  8  1  8
B  1  9  3  1  2

Dataframe ordenado (filas):
    C  A  E  B  D
A  6  0  4  3  6
B  1  9  3  1  2
C  7  9  4  3  0
D  3  2  8  1  8
E  6  7  8  6  7
```

In [26]:
```
df = pd.DataFrame(np.random.randint(0,10, (5, 5)), index=list('ACEDB'), columns=list('CA
print('Dataframe original:\n', df)

df2 = None

#Pon tu código aquí.
#Sugerencia: usa el método sort_index indicando con el argumento
#   axis que queremos ordenar el índice de filas.

#

print('\nDataframe ordenado (filas):\n', df2)
```

```
Dataframe original:
    C  A  E  B  D
A  8  3  1  0  7
C  4  0  7  4  1
E  7  9  8  2  7
D  2  6  4  5  7
B  6  4  4  6  9
```

```
Dataframe ordenado (filas):
 None
```

Ejercicio 12: A partir de un dataframe `df` queremos obtener una versión con el índice de columnas ordenado.

La salida obtenida debería ser algo parecido a lo siguiente:

```
Dataframe original:
    C  A  E  B  D
A  1  8  1  8  7
C  5  0  1  4  0
E  4  3  1  0  0
D  7  0  6  8  0
B  4  8  0  1  1

Dataframe ordenado (columnas):
    A  B  C  D  E
A  8  8  1  7  1
C  0  4  5  0  1
E  3  0  4  0  1
D  0  8  7  0  6
B  8  1  4  1  0
```

In [27]:
```
df = pd.DataFrame(np.random.randint(0,10, (5, 5)), index=list('ACEDB'), columns=list('CA
print('Dataframe original:\n', df)

df2 = None

#Pon tu código aquí.
#Sugerencia: usa el método sort_index indicando con el argumento
#   axis que queremos ordenar el índice de filas.

#

print('\nDataframe ordenado (columnas):\n', df2)
```

```
Dataframe original:
    C  A  E  B  D
A  8  2  8  0  4
C  8  7  5  1  5
E  5  1  3  0  7
D  2  5  5  2  9
B  5  8  2  1  6

Dataframe ordenado (columnas):
 None
```

Ejercicio 13: A partir de un dataframe `df` queremos obtener una versión con los índices de filas y columnas ordenados.

La salida obtenida debería ser algo parecido a lo siguiente:

```
Dataframe original:
    C  A  E  B  D
A  9  1  5  0  5
C  7  7  7  5  4
E  7  1  5  3  2
```

```
D  8  9  5  5  4
B  9  9  9  6  7

Dataframe ordenado:
    A  B  C  D  E
A  1  0  9  5  5
B  9  6  9  7  9
C  7  5  7  4  7
D  9  5  8  4  5
E  1  3  7  2  5
```

In [28]:

```
df = pd.DataFrame(np.random.randint(0,10, (5, 5)), index=list('ACEDB'), columns=list('CA
print('Dataframe original:\n', df)

df2 = None

#Pon tu código aquí.
#Sugerencia: Encadena las dos operaciones de ordenación realizas
#   en los ejercicios anteriores.

#

print('\nDataframe ordenado:\n', df2)
```

```
Dataframe original:
    C  A  E  B  D
A  6  1  8  3  0
C  3  1  8  7  4
E  9  9  6  0  7
D  5  6  1  1  9
B  1  5  1  0  7

Dataframe ordenado:
 None
```

Ejercicio 14: A partir de un dataframe `df` queremos obtener una versión con los índices de segundo nivel tanto de filas como de columnas ordenados.

La salida obtenida debería ser algo parecido a lo siguiente:

```
Dataframe original:
C1     Q        P
C2     q  o  p  q  o  p
F1 F2
B  b   4  8  3  0  1  0
   a   1  1  1  0  6  9
   c   2  2  4  1  4  7
A  b   7  6  6  6  4  2
   a   7  3  8  6  1  5
   c   9  8  2  8  7  1

Dataframe ordenado:
C1     P  Q  P  Q  P  Q
C2     o  o  p  p  q  q
F1 F2
A  a   1  3  5  8  6  7
B  a   6  1  9  1  0  1
A  b   4  6  2  6  6  7
B  b   1  8  0  3  0  4
```

```
   A  c   7  8  1  2  8  9
   B  c   4  2  7  4  1  2
```

In [29]:

```
df = pd.DataFrame(np.random.randint(0, 10, (6,6)),
                  index=pd.MultiIndex.from_product([list('BA'), list('bac')], names=['F1
                  columns=pd.MultiIndex.from_product([list('QP'), list('qop')], names=['
print('Dataframe original:\n', df)
df2 = None

#Pon tu código aquí.
#Sugerencia: Encadena las dos operaciones de ordenación realizas
#   en los ejercicios anteriores.

#

print('\nDataframe ordenado:\n', df2)
```

```
Dataframe original:
 C1     Q        P
C2     q  o  p  q  o  p
F1 F2
B  b   5  8  7  3  6  1
   a   2  3  9  2  3  8
   c   7  5  9  9  6  4
A  b   5  1  2  8  4  2
   a   6  0  2  0  2  5
   c   1  6  2  4  1  3

Dataframe ordenado:
 None
```

In [ ]: