Guide 7 pandas functionality

May 14, 2019

Guía 7: Funcionalidad de pandas
Ingeniería en Estadística, Universidad de Valparaíso
Profesor: Eduardo Jorquera - eduardo.jorquera@postgrado.uv.cl

1 Funcionalidad básica de pandas

`axes`: Retorna una lista de las indexaciones de etiqueta
`dtype`: Retorna el dtype de un objeto
`empty`: Retorna si una serie es vacía
`ndim`: Retorna el número de dimensiones de los datos subyacentes, por definición 1
`size`: Retorna el número de elementos en los datos subyacentes
`values`: Retorna la Series como ndarray
`head()`: Retorna las primeras n filas
`tail()`: Retorna las últimas n filas

2 Funcionalidades de agrupación

Cualquier operación para agrupar datos envuelve una de las siguientes operaciones al objeto original:

- Dividir un objeto
- Aplicar una función
- Combinar los resultados

En muchas situaciones, dividimos los datos en conjuntos y aplicamos alguna funcionalidad en cada subconjunto. En la funcionalidad de aplicar, podemos encontrar los siguientes operadores:

- Agregación : calcular un estadístico
- Transformation: hacer alguna operación de grupo específica
- Filtración: descartar los datos que cumplan ciertas condiciones

Ahora crearemos un objeto dataframe y haremos todas las operaciones en éste.

```
In [1]: import pandas as pd

        ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
           'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
```

```
            'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
            'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
            'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
        df = pd.DataFrame(ipl_data)

        print (df)
```

```
      Team  Rank  Year  Points
0   Riders     1  2014     876
1   Riders     2  2015     789
2   Devils     2  2014     863
3   Devils     3  2015     673
4    Kings     3  2014     741
5    kings     4  2015     812
6    Kings     1  2016     756
7    Kings     1  2017     788
8   Riders     2  2016     694
9   Royals     4  2014     701
10  Royals     1  2015     804
11  Riders     2  2017     690
```

2.1 Dividir los datos en grupos

Los objetos de pandas pueden ser divididos en cualquiera de sus objetos. Hay varias maneras de dividir un objeto, por ejemplo:

- `obj.groupby('key')`
- `obj.groupby(['key1','key2'])`
- `obj.groupby(key,axis=1)`

Ahora veamos cómo la agrupación puede ser aplicada en un objeto dataframe:

```
In [7]: import pandas as pd

        ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
            'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
            'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
            'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
            'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
        df = pd.DataFrame(ipl_data)
        print("Mostramos el dataframe completo:\n",df,"\nAhora mostramos cómo agrupar por equi
        print (df.groupby('Team'))
```

```
Mostramos el dataframe completo:
      Team  Rank  Year  Points
0   Riders     1  2014     876
1   Riders     2  2015     789
2   Devils     2  2014     863
```

```
3   Devils     3  2015     673
4    Kings     3  2014     741
5    kings     4  2015     812
6    Kings     1  2016     756
7    Kings     1  2017     788
8   Riders     2  2016     694
9   Royals     4  2014     701
10  Royals     1  2015     804
11  Riders     2  2017     690
Ahora mostramos cómo agrupar por equipo (Team)
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7f0e364a5f60>
```

Ahora veamos cómo ver las agrupaciones:

```
In [9]: import pandas as pd

        ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
           'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
           'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
           'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
           'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
        df = pd.DataFrame(ipl_data)

        print (df.groupby('Team').groups)
```

```
{'Devils': Int64Index([2, 3], dtype='int64'), 'Kings': Int64Index([4, 6, 7], dtype='int64'), '
```

2.1.1 Ejemplo: agrupar usando múltiples columnas

```
In [11]: import pandas as pd

         ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
            'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
            'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
            'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
            'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
         df = pd.DataFrame(ipl_data)

         print (df.groupby(['Team','Year']).groups)
```

```
{('Devils', 2014): Int64Index([2], dtype='int64'), ('Devils', 2015): Int64Index([3], dtype='in
```

2.2 Iterando en los grupos

Con el objeto `groupby` a la mano, ahora podemos iterar a través de objetos similares a `itertools.obj`.

```
In [12]: import pandas as pd

         ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
            'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
            'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
            'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
            'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
         df = pd.DataFrame(ipl_data)

         grouped = df.groupby('Year')

         for name,group in grouped:
             print (name)
             print (group)
```

```
2014
     Team  Rank  Year  Points
0  Riders     1  2014     876
2  Devils     2  2014     863
4   Kings     3  2014     741
9  Royals     4  2014     701
2015
      Team  Rank  Year  Points
1   Riders     2  2015     789
3   Devils     3  2015     673
5    kings     4  2015     812
10  Royals     1  2015     804
2016
     Team  Rank  Year  Points
6   Kings     1  2016     756
8  Riders     2  2016     694
2017
      Team  Rank  Year  Points
7    Kings     1  2017     788
11  Riders     2  2017     690
```

2.3 Seleccionar un grupo

Usando el método `get_group`, podemos seleccionar un sólo grupo.

```
In [16]: import pandas as pd

         ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
            'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
            'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
            'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
            'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
```

```
        df = pd.DataFrame(ipl_data)

        grouped = df.groupby('Year')
        print( grouped.get_group(2014))
```

```
     Team  Rank  Year  Points
0  Riders     1  2014     876
2  Devils     2  2014     863
4   Kings     3  2014     741
9  Royals     4  2014     701
```

2.4 Agregación

Una función agregada retorna un sólo valor agregado por cada grupo. Una vez el objeto groupby está creado, muchas operaciones de agregación pueden ser hechas en los datos agrupados.

Una agregación obvia es mediante el método agg:

```
In [18]: import pandas as pd
         import numpy as np

         ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
            'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
            'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
            'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
            'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
         df = pd.DataFrame(ipl_data)

         grouped = df.groupby('Year')
         print( grouped['Points'].agg(np.mean))
```

```
Year
2014    795.25
2015    769.50
2016    725.00
2017    739.00
Name: Points, dtype: float64
```

Otra forma de ver el tamaño de cada grupo es aplicando la función size:

```
In [21]: import pandas as pd
         import numpy as np

         ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
            'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
            'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
            'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
            'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
```

```
         df = pd.DataFrame(ipl_data)

         grouped = df.groupby('Team')
         print (grouped.agg(np.size))
```

```
        Rank  Year  Points
Team
Devils     2     2       2
Kings      3     3       3
Riders     4     4       4
Royals     2     2       2
kings      1     1       1
```

2.4.1 Aplicando funciones de agregación múltiple simultáneamente

Con Series agrupadas, también puedes pasar a lista o diccionario de funciones para hacer agregación y generar un dataframe como output:

```
In [23]: import pandas as pd
         import numpy as np

         ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
            'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
            'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
            'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
            'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
         df = pd.DataFrame(ipl_data)

         grouped = df.groupby('Team')
         print (grouped['Points'].agg([np.sum, np.mean, np.std]))
```

```
         sum        mean         std
Team
Devils  1536  768.000000  134.350288
Kings   2285  761.666667   24.006943
Riders  3049  762.250000   88.567771
Royals  1505  752.500000   72.831998
kings    812  812.000000         NaN
```

2.5 Transformaciones

Transformación en un grupo o una columna retorna un objeto que es indexado del mismo tamaño que el grupo en el que está. Entonces, la transformación debería retornar un resultado que es del mismo tamaño de un grupo cortado:

```
In [25]: import pandas as pd
         import numpy as np
```

```
        ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
           'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
           'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
           'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
           'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
        df = pd.DataFrame(ipl_data)

        grouped = df.groupby('Team')
        score = lambda x: (x - x.mean()) / x.std()*10
        print (grouped.transform(score))
```

```
         Rank       Year     Points
0  -15.000000 -11.618950  12.843272
1    5.000000  -3.872983   3.020286
2   -7.071068  -7.071068   7.071068
3    7.071068   7.071068  -7.071068
4   11.547005 -10.910895  -8.608621
5         NaN        NaN        NaN
6   -5.773503   2.182179  -2.360428
7   -5.773503   8.728716  10.969049
8    5.000000   3.872983  -7.705963
9    7.071068  -7.071068  -7.071068
10  -7.071068   7.071068   7.071068
11   5.000000  11.618950  -8.157595
```

2.6 Filtración

Filtra los datos según un criterio definido y retorna el conjunto de los datos. La función `filtrer` es usada para filtrar los datos.

```
In [26]: import pandas as pd
         import numpy as np

         ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
            'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
            'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
            'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
            'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
         df = pd.DataFrame(ipl_data)

         print (df.groupby('Team').filter(lambda x: len(x) >= 3))
```

```
      Team  Rank  Year  Points
0   Riders     1  2014     876
1   Riders     2  2015     789
4    Kings     3  2014     741
6    Kings     1  2016     756
```

```
7    Kings      1  2017     788
8   Riders      2  2016     694
11  Riders      2  2017     690
```

3 Estadísticas Descriptivas

Una gran cantidad de métodos colectivamente calculan estadísticos descriptivos y otras operaciones relacionadas al dataframe. La mayoría de éstas son agregaciones como la suma `sum()`, media `mean()`, pero hay otras como `sumsum()` que producen un objeto del mismo tamaño. Hablando en términos generales, estos métodos toman un eje (o columna) como argumento, como por ejemplo `ndarray.{sum, std, ..}`, pero el eje puede ser especificado por el nombre o entero * `DataFrame  index (axis=0, default), columns (axis=1)`

Creemos un dataframe y veamos el uso de estos objetos en esta guía para el resto de las operaciones.

```
In [28]: import pandas as pd
         import numpy as np

         #Crear un diccionario de series
         d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',
            'Lee','David','Gasper','Betina','Andres']),
            'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
            'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])
         }

         #Crear un dataframe
         df = pd.DataFrame(d)
         print (df)
```

```
      Name  Age  Rating
0      Tom   25    4.23
1    James   26    3.24
2    Ricky   25    3.98
3      Vin   23    2.56
4    Steve   30    3.20
5    Smith   29    4.60
6     Jack   23    3.80
7      Lee   34    3.78
8    David   40    2.98
9   Gasper   30    4.80
10  Betina   51    4.10
11  Andres   46    3.65
```

3.1 sum()

Retorna la suma de los valores para el eje (o columna) requeridos. Por defecto, es ele eje índice (axis=0)

```
In [29]: import pandas as pd
         import numpy as np

         #Crear un diccionario de series
         d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',
            'Lee','David','Gasper','Betina','Andres']),
            'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
            'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])
         }

         #Crear un dataframe
         df = pd.DataFrame(d)
         #print (df)
         print (df.sum())
```

```
Name      TomJamesRickyVinSteveSmithJackLeeDavidGasperBe...
Age                                                     382
Rating                                                44.92
dtype: object
```

Cada columna individual es agregada individualmetne.

3.2 axis=1:

```
In [30]: import pandas as pd
         import numpy as np

         #Crear un diccionario de series
         d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',
            'Lee','David','Gasper','Betina','Andres']),
            'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
            'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])
         }

         #Crear un dataframe
         df = pd.DataFrame(d)
         #print (df)
         print (df.sum(1))
```

```
0     29.23
1     29.24
2     28.98
3     25.56
```

```
4     33.20
5     33.60
6     26.80
7     37.78
8     42.98
9     34.80
10    55.10
11    49.65
dtype: float64
```

3.3 mean():

Retorna la media aritmética

```
In [31]: import pandas as pd
         import numpy as np

         #Crear un diccionario de series
         d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',
            'Lee','David','Gasper','Betina','Andres']),
            'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
            'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])
         }

         #Crear un dataframe
         df = pd.DataFrame(d)
         #print (df)
         print (df.mean())
```

```
Age       31.833333
Rating     3.743333
dtype: float64
```

3.4 std():

Retorna la desviación estándar de Bressel de las columnas numéricas

```
In [32]: import pandas as pd
         import numpy as np

         #Crear un diccionario de series
         d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',
            'Lee','David','Gasper','Betina','Andres']),
            'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
            'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])
         }
```

```
         #Crear un dataframe
         df = pd.DataFrame(d)
         #print (df)
         print (df.std())
```

```
Age       9.232682
Rating    0.661628
dtype: float64
```

Así hay varias funciones que son parte de la estadística descriptiva:

```
* count()       Número de observaciones no nulas
* sum()         Suma de los valores
* mean()        Media de los valores
* median()      Mediana de los valores
* mode()        Moda de los valores
* std()         Desviación estándar de los valores
* min()         Valor mínimo
* max()         Valor máximo
* abs()         Valor absoluto
* prod()        Producto de los valores
* cumsum()      Suma acumulada
* cumprod()     Producto acumulado
```

3.5 Resumiendo los datos

La función `describe()` calcula los estadísticos de resumen pertinentes a las columnas del dataframe

```
In [33]: import pandas as pd
         import numpy as np

         #Crear un diccionario de series
         d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',
            'Lee','David','Gasper','Betina','Andres']),
            'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
            'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])
         }

         #Crear un dataframe
         df = pd.DataFrame(d)
         #print (df)
         print (df.describe())
```

```
             Age     Rating
count  12.000000  12.000000
mean   31.833333   3.743333
std     9.232682   0.661628
```

```
min     23.000000   2.560000
25%     25.000000   3.230000
50%     29.500000   3.790000
75%     35.500000   4.132500
max     51.000000   4.800000
```

Esta función entrega la media, desviación estándar, y medidas de posición. Y, la función excluye las columnas de caracteres. El argumento `include` permite pasar la información necesaria para saber cómo la columna debe ser tomada para ser resumida. Por defecto toma el valor `number`: * `object` resume columnas de caracteres. * `number` resume columnas numéricas * `all` resume todas las columnas juntas

Ahora, usemos en las siguientes especificaciones dentro del programa para ver el output:

```
In [34]: import pandas as pd
         import numpy as np

         #Crear un diccionario de series
         d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',
            'Lee','David','Gasper','Betina','Andres']),
            'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
            'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])
         }

         #Crear un dataframe
         df = pd.DataFrame(d)
         #print (df)
         print (df.describe(include=['object']))
```

```
        Name
count     12
unique    12
top      Vin
freq       1
```

```
In [37]: import pandas as pd
         import numpy as np

         #Crear un diccionario de series
         d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',
            'Lee','David','Gasper','Betina','Andres']),
            'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
            'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])
         }

         #Crear un dataframe
         df = pd.DataFrame(d)
```

```
        #print (df)
        print (df.describe(include='all'))
```

```
       Name        Age     Rating
count    12  12.000000  12.000000
unique   12        NaN        NaN
top     Vin        NaN        NaN
freq      1        NaN        NaN
mean    NaN  31.833333   3.743333
std     NaN   9.232682   0.661628
min     NaN  23.000000   2.560000
25%     NaN  25.000000   3.230000
50%     NaN  29.500000   3.790000
75%     NaN  35.500000   4.132500
max     NaN  51.000000   4.800000
```

4 Juntando datos

Pandas tiene muchas operaciones para juntar distintos dataframes en uno solo, la sintaxis básica es la siguiente:

```
In [ ]: pd.merge(left, right, how='inner', on=None, left_on=None, right_on=None,
        left_index=False, right_index=False, sort=True)
```

Aquí, tenemos los siguientes parámetros:

```
left - Un objeto dataframe
right - Otro objeto dataframe
on  Columnas (nombres) sobre la cual se juntan. Deben existir en el dataframe de la izquierda
left_on  Columnas del datframe de la izquierda (left) para usar como llaves. Pueden ser nombres
right_on  Columnas del datframe de la derecha (right) para usar como llaves. Pueden ser nombres
left_index  Si es True (verdadero), usa la indexación (las etiquetas de las filas) del datafram
right_index  Musmo uso que el left_index para el dataframe de la derecha.
how  Puede ser `left`, `right`, `outer`, `inner`. Por defecto es `inner`. Cada método es descri
sort  Ordena el dataframe de resultado por las claves de unión en orden lexicográfico order. Po
```

Creamos un dataframe y vemos cómo funciona el mege:

```
In [40]: import pandas as pd
         left = pd.DataFrame({
            'id':[1,2,3,4,5],
            'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
            'subject_id':['sub1','sub2','sub4','sub6','sub5']})
         right = pd.DataFrame(
            {'id':[1,2,3,4,5],
            'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
            'subject_id':['sub2','sub4','sub3','sub6','sub5']})
         print (left)
         print ("\n",right)
```

```
   id    Name subject_id
0   1    Alex       sub1
1   2     Amy       sub2
2   3   Allen       sub4
3   4   Alice       sub6
4   5  Ayoung       sub5

    id   Name subject_id
0   1  Billy       sub2
1   2  Brian       sub4
2   3   Bran       sub3
3   4  Bryce       sub6
4   5  Betty       sub5
```

4.0.1 Juntamos dos dataframes en una clave (o llave)

```
In [42]: import pandas as pd
         left = pd.DataFrame({
            'id':[1,2,3,4,5],
            'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
            'subject_id':['sub1','sub2','sub4','sub6','sub5']})
         right = pd.DataFrame({
             'id':[1,2,3,4,5],
             'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
             'subject_id':['sub2','sub4','sub3','sub6','sub5']})
         print (pd.merge(left,right,on='id'))
```

```
   id  Name_x subject_id_x Name_y subject_id_y
0   1    Alex         sub1  Billy         sub2
1   2     Amy         sub2  Brian         sub4
2   3   Allen         sub4   Bran         sub3
3   4   Alice         sub6  Bryce         sub6
4   5  Ayoung         sub5  Betty         sub5
```

4.0.2 Juntamos dos dataframes en varias claves

```
In [43]: import pandas as pd
         left = pd.DataFrame({
            'id':[1,2,3,4,5],
            'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
            'subject_id':['sub1','sub2','sub4','sub6','sub5']})
         right = pd.DataFrame({
             'id':[1,2,3,4,5],
             'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
             'subject_id':['sub2','sub4','sub3','sub6','sub5']})
         print (pd.merge(left,right,on=['id','subject_id']))
```

```
   id  Name_x subject_id Name_y
0   4   Alice       sub6  Bryce
1   5  Ayoung       sub5  Betty
```

4.1 Unión usando el argumento `how`

Se usa para especificar cómo determinar cuáles claves son incluidas en la tabla resultante. Si una combinación de clave no aparece ni en el dataframe de la derecha ni en el de la izquierda, los valores de la tabla al juntarse serán `NA`.

Aquí está el cómo funcionan:

- `Merge Method  Equicalente a SQL  Description`
- `left  LEFT OUTER JOIN  Usa las claves del objeto de la izquierda`
- `right  RIGHT OUTER JOIN  Usa las claves del objeto de la derecha`
- `outer  FULL OUTER JOIN  Usa uniones de clave`
- `inner  INNER JOIN  Usa intersección de claves`

4.1.1 Por la izquierda

```
In [44]: import pandas as pd
         left = pd.DataFrame({
            'id':[1,2,3,4,5],
            'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
            'subject_id':['sub1','sub2','sub4','sub6','sub5']})
         right = pd.DataFrame({
            'id':[1,2,3,4,5],
            'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
            'subject_id':['sub2','sub4','sub3','sub6','sub5']})
         print (pd.merge(left, right, on='subject_id', how='left'))
```

```
   id_x  Name_x subject_id  id_y Name_y
0     1    Alex       sub1   NaN    NaN
1     2     Amy       sub2   1.0  Billy
2     3   Allen       sub4   2.0  Brian
3     4   Alice       sub6   4.0  Bryce
4     5  Ayoung       sub5   5.0  Betty
```

4.1.2 Por la derecha

```
In [45]: import pandas as pd
         left = pd.DataFrame({
            'id':[1,2,3,4,5],
            'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
            'subject_id':['sub1','sub2','sub4','sub6','sub5']})
         right = pd.DataFrame({
            'id':[1,2,3,4,5],
```

```
            'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
            'subject_id':['sub2','sub4','sub3','sub6','sub5']})
        print (pd.merge(left, right, on='subject_id', how='right'))
```

```
   id_x  Name_x subject_id  id_y Name_y
0   2.0     Amy       sub2     1  Billy
1   3.0   Allen       sub4     2  Brian
2   4.0   Alice       sub6     4  Bryce
3   5.0  Ayoung       sub5     5  Betty
4   NaN     NaN       sub3     3   Bran
```

4.1.3 Exterior (outer)

```
In [46]: import pandas as pd
         left = pd.DataFrame({
            'id':[1,2,3,4,5],
            'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
            'subject_id':['sub1','sub2','sub4','sub6','sub5']})
         right = pd.DataFrame({
            'id':[1,2,3,4,5],
            'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
            'subject_id':['sub2','sub4','sub3','sub6','sub5']})
         print (pd.merge(left, right, how='outer', on='subject_id'))
```

```
   id_x  Name_x subject_id  id_y Name_y
0   1.0    Alex       sub1   NaN    NaN
1   2.0     Amy       sub2   1.0  Billy
2   3.0   Allen       sub4   2.0  Brian
3   4.0   Alice       sub6   4.0  Bryce
4   5.0  Ayoung       sub5   5.0  Betty
5   NaN     NaN       sub3   3.0   Bran
```

4.1.4 Interior (inner)

```
In [47]: import pandas as pd
         left = pd.DataFrame({
            'id':[1,2,3,4,5],
            'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
            'subject_id':['sub1','sub2','sub4','sub6','sub5']})
         right = pd.DataFrame({
            'id':[1,2,3,4,5],
            'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
            'subject_id':['sub2','sub4','sub3','sub6','sub5']})
         print (pd.merge(left, right, on='subject_id', how='inner'))
```

```
   id_x  Name_x subject_id  id_y Name_y
0     2     Amy       sub2     1  Billy
```

```
1     3   Allen       sub4     2  Brian
2     4   Alice       sub6     4  Bryce
3     5  Ayoung       sub5     5  Betty
```