Crear DataFrames agrupados

`pandas` tiene una clase de objetos llamados `GroupedDataFrame` que permite agrupar las filas (o columnas) de un DataFrame.

- Un ejemplo podría ser un conjunto de datos asociados a individuos para el que que queramos distinguir entre hombres y mujeres a la hora de calcular indicadores.
- Otro caso podría ser el de mediciones asociadas a instantes de tiempo, y queremos calcular resúmenes para cada hora del día.

Para crear dataframes agrupados, usaremos el método `groupby`. (ver User guide, Group by: split-apply-combine)

```
import pandas as pd
import numpy as np
from os import path
```

```
DATA_DIRECTORY = path.join('..', '..', 'data')
```

Consideremos el siguiente DataFrame:

```
df = pd.DataFrame(
    {
     "X": ['a', 'a', 'a', 'a', 'b', 'b', 'c', 'c'],
     "Y": np.arange(8),
     "Z": np.arange(8,16)
    }
)
df
```

	X	Y	Z
0	a	0	8
1	a	1	9
2	a	2	10
3	a	3	11
4	b	4	12
5	b	5	13
6	c	6	14
7	c	7	15

Agrupamos según los valores de la columna X

```
agrupados = df.groupby('X')
```

Podemos iterar un `GroupedDataFrame` para recorrer los grupos:

```
for n, g in agrupados:
    print(f'Nombre del grupo: {n}')
    print(g)
```

```
Nombre del grupo: a
   X  Y   Z
0  a  0   8
1  a  1   9
2  a  2  10
3  a  3  11
Nombre del grupo: b
   X  Y   Z
4  b  4  12
5  b  5  13
Nombre del grupo: c
   X  Y   Z
6  c  6  14
7  c  7  15
```

Podemos aplicarle métodos preparados para `GroupedDataFrame` como `mean`, `sum` o `describe`

```
agrupados.mean()
```

X	Y	Z
a	1.5	9.5
b	4.5	12.5
c	6.5	14.5

Nos devuelve el valor de la media por grupos de las dos columnas Y y Z.

```
agrupados.sum()
```

	Y	Z
X		
a	6	38
b	9	25
c	13	29

```
agrupados.describe()
```

	Y							Z						
	count	mean	std	min	25%	50%	75%	max	count	mean	std	min	25%	50%	75%	max
X																
a	4.0	1.5	1.290994	0.0	0.75	1.5	2.25	3.0	4.0	9.5	1.290994	8.0	8.75	9.5	10.25	11.0
b	2.0	4.5	0.707107	4.0	4.25	4.5	4.75	5.0	2.0	12.5	0.707107	12.0	12.25	12.5	12.75	13.0
c	2.0	6.5	0.707107	6.0	6.25	6.5	6.75	7.0	2.0	14.5	0.707107	14.0	14.25	14.5	14.75	15.0

Podemos especificar más de una columna para crear los grupos.

Para ilustrarlo, cargamos el DataFrame de `flights` que contiene todos los vuelos que salieron de los tres aeropuertos de NYC en 2013.

```
flights = pd.read_feather(path.join(DATA_DIRECTORY, 'flights.feather'))
flights.head()
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailnum	origin	dest	air_time
0	2013	1	1	517.0	515	2.0	830.0	819	11.0	UA	1545	N14228	EWR	IAH	227.0
1	2013	1	1	533.0	529	4.0	850.0	830	20.0	UA	1714	N24211	LGA	IAH	227.0
2	2013	1	1	542.0	540	2.0	923.0	850	33.0	AA	1141	N619AA	JFK	MIA	160.0
3	2013	1	1	544.0	545	-1.0	1004.0	1022	-18.0	B6	725	N804JB	JFK	BQN	183.0
4	2013	1	1	554.0	600	-6.0	812.0	837	-25.0	DL	461	N668DN	LGA	ATL	116.0

Agrupamos los vuelos para hacer recuentos desglosados por mes y por aeropuerto de origen

```
agrupados = flights.groupby(['month', 'origin'])
```

Aplicamos el método `size` para hacer el recuento de filas (vuelos) por grupo:

```
agrupados.size()
```

```
month  origin
1      EWR       9893
       JFK       9161
       LGA       7950
2      EWR       9107
       JFK       8421
       LGA       7423
3      EWR      10420
       JFK       9697
       LGA       8717
4      EWR      10531
       JFK       9218
       LGA       8581
5      EWR      10592
       JFK       9397
       LGA       8807
6      EWR      10175
       JFK       9472
       LGA       8596
7      EWR      10475
       JFK      10023
       LGA       8927
```

```
8       EWR        10359
        JFK         9983
        LGA         8985
9       EWR         9550
        JFK         8908
        LGA         9116
10      EWR        10104
        JFK         9143
        LGA         9642
11      EWR         9707
        JFK         8710
        LGA         8851
12      EWR         9922
        JFK         9146
        LGA         9067
dtype: int64
```

Podemos crear grupos al especificar una función que se aplica a los valores del index

Para ilustrarlo, cargamos el fichero `mompean`

```
mompean = pd.read_csv(path.join(DATA_DIRECTORY, 'mompean.csv'), parse_dates=['FechaHora'], index_col='FechaHora')
mompean.head()
```

FechaHora	NO	NO2	SO2	O3	TMP	HR	NOX	DD	PRB	RS	VV	C6H6	C7H8	XIL	PM10	Ruido
2010-01-01 00:00:00	4.0	7.0	17.0	NaN	NaN	NaN	14.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	5.0	56.0
2010-01-01 01:00:00	6.0	12.0	18.0	NaN	NaN	NaN	21.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	15.0	58.0
2010-01-01 02:00:00	6.0	17.0	17.0	NaN	NaN	NaN	26.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12.0	60.0
2010-01-01 03:00:00	5.0	10.0	18.0	NaN	NaN	NaN	18.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.0	57.0
2010-01-01 04:00:00	4.0	8.0	19.0	NaN	NaN	NaN	14.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	5.0	55.0

Queremos crear grupos que correspondan a las distintas horas del día, para ver si hay diferencias en el patrón horario de contaminación.

Para ello, pasamos a `groupby` una función que se aplique a las etiquetas de las filas (index) y que extraiga la hora de un objeto de tipo `dtime`

```
agrupados = mompean.groupby(lambda x: x.hour)
```

Hemos usado una función anónima. Aplicamos el método `mean`

```
agrupados.mean()
```

FechaHora	NO	NO2	SO2	O3	TMP	HR	NOX	DD	PRB	RS	VV	C6H6	C7H8	XIL	PM10	Ruid
0	9.618014	27.402604	7.516146	53.871709	NaN	NaN	42.020890	NaN	NaN	NaN	NaN	NaN	NaN	NaN	24.416717	57.28003
1	7.779196	25.304997	7.399091	51.332306	NaN	NaN	37.115698	NaN	NaN	NaN	NaN	NaN	NaN	NaN	22.790531	55.93553
2	6.599348	22.842077	7.323167	49.531390	NaN	NaN	32.844795	NaN	NaN	NaN	NaN	NaN	NaN	NaN	21.687349	54.25980
3	4.858891	19.067700	7.234885	48.993829	NaN	NaN	26.410005	NaN	NaN	NaN	NaN	NaN	NaN	NaN	20.119487	53.24942
4	4.147523	16.349483	7.184957	48.060045	NaN	NaN	22.607512	NaN	NaN	NaN	NaN	NaN	NaN	NaN	18.802399	52.39661
5	4.095316	15.585240	7.150509	45.970250	NaN	NaN	21.786220	NaN	NaN	NaN	NaN	NaN	NaN	NaN	17.874605	51.65307
6	5.992364	18.083174	7.177069	41.162830	NaN	NaN	27.149168	NaN	NaN	NaN	NaN	NaN	NaN	NaN	17.907378	53.04695
7	12.616621	25.141144	7.252747	33.026404	NaN	NaN	44.355586	NaN	NaN	NaN	NaN	NaN	NaN	NaN	20.484621	56.91692
8	18.700545	30.695095	7.376575	27.968004	NaN	NaN	59.242234	NaN	NaN	NaN	NaN	NaN	NaN	NaN	24.919906	59.56812
9	34.211874	36.715686	7.711188	26.932959	NaN	NaN	89.038399	NaN	NaN	NaN	NaN	NaN	NaN	NaN	32.868169	60.27097
10	26.159482	32.184573	7.810072	39.895430	NaN	NaN	72.175290	NaN	NaN	NaN	NaN	NaN	NaN	NaN	34.702703	60.94709
11	13.300164	25.110716	7.697393	55.724283	NaN	NaN	45.379716	NaN	NaN	NaN	NaN	NaN	NaN	NaN	29.696712	61.31312
12	9.329697	22.087810	7.825563	65.378219	NaN	NaN	36.264249	NaN	NaN	NaN	NaN	NaN	NaN	NaN	27.035784	61.99770
13	8.356833	20.896150	7.995197	71.142339	NaN	NaN	33.565618	NaN	NaN	NaN	NaN	NaN	NaN	NaN	25.787697	62.33816
14	7.735246	19.927721	8.148641	75.562849	NaN	NaN	31.660531	NaN	NaN	NaN	NaN	NaN	NaN	NaN	24.944570	62.38167
15	7.167794	18.990257	8.609205	78.299637	NaN	NaN	29.848714	NaN	NaN	NaN	NaN	NaN	NaN	NaN	23.915736	61.67838
16	6.474696	17.648714	8.834398	80.232617	NaN	NaN	27.457645	NaN	NaN	NaN	NaN	NaN	NaN	NaN	23.579356	60.81240
17	6.404491	18.028950	9.035134	80.065624	NaN	NaN	27.702922	NaN	NaN	NaN	NaN	NaN	NaN	NaN	24.367599	60.75114
18	6.553600	20.003249	9.063149	77.836409	NaN	NaN	29.928803	NaN	NaN	NaN	NaN	NaN	NaN	NaN	24.708589	61.22035
19	7.001896	23.205038	8.684084	73.640817	NaN	NaN	33.828548	NaN	NaN	NaN	NaN	NaN	NaN	NaN	24.993455	61.56967
20	8.330986	25.681203	8.342933	69.847144	NaN	NaN	38.321506	NaN	NaN	NaN	NaN	NaN	NaN	NaN	25.506697	61.81226
21	10.382592	28.194685	7.983729	65.330907	NaN	NaN	43.979393	NaN	NaN	NaN	NaN	NaN	NaN	NaN	26.555189	61.72226
22	11.746273	29.620222	7.777807	60.919373	NaN	NaN	47.500678	NaN	NaN	NaN	NaN	NaN	NaN	NaN	26.788431	60.70367
23	12.008955	29.771235	7.646823	56.430933	NaN	NaN	48.046676	NaN	NaN	NaN	NaN	NaN	NaN	NaN	25.949129	58.90804

Podemos aplicar más de una función en `groupby`

Pasamos una lista de funciones

```
agrupados = mompean.groupby([lambda x: x.dayofweek, lambda x: x.hour])
```

Calculamos la media desglosada por grupo:

```
agrupados.mean()
```

		NO	NO2	SO2	O3	TMP	HR	NOX	DD	PRB	RS	VV	C6H6	C7H8	XIL	PM10
FechaHora	**FechaHora**															
0	**0**	7.562380	25.581574	7.512287	55.924303	NaN	NaN	37.084453	NaN	NaN	NaN	NaN	NaN	NaN	NaN	22.472486
	1	6.180077	23.312261	7.435606	53.521912	NaN	NaN	32.668582	NaN	NaN	NaN	NaN	NaN	NaN	NaN	20.853890
	2	5.517241	21.136015	7.361742	51.169323	NaN	NaN	29.501916	NaN	NaN	NaN	NaN	NaN	NaN	NaN	20.206831
	3	4.126437	17.105364	7.229167	50.878486	NaN	NaN	23.373563	NaN	NaN	NaN	NaN	NaN	NaN	NaN	18.842505
	4	3.624521	14.111111	7.143939	50.537849	NaN	NaN	19.616858	NaN	NaN	NaN	NaN	NaN	NaN	NaN	17.838710
...	**...**	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6	**19**	5.676245	18.860153	8.581132	77.117296	NaN	NaN	27.421456	NaN	NaN	NaN	NaN	NaN	NaN	NaN	22.644612
	20	6.545977	20.641762	8.330189	73.753479	NaN	NaN	30.588123	NaN	NaN	NaN	NaN	NaN	NaN	NaN	22.718336
	21	7.767754	23.921305	7.835539	68.711155	NaN	NaN	35.700576	NaN	NaN	NaN	NaN	NaN	NaN	NaN	23.905303
	22	8.694818	26.278311	7.667297	63.896414	NaN	NaN	39.462572	NaN	NaN	NaN	NaN	NaN	NaN	NaN	23.990512
	23	9.168906	27.220729	7.655955	59.272908	NaN	NaN	41.084453	NaN	NaN	NaN	NaN	NaN	NaN	NaN	23.339658

168 rows × 16 columns

Realizar operaciones sobre `GroupedDataFrame`s

Podemos realizar tres tipos de operaciones que se traducen en tres verbos

- `aggregate`: se trata de resumir cada grupo calculando uno o varios indicadores, por ejemplo su media o su media y desviación típica
- `transform`: se trata de calcular para cada grupo una o varias columnas con el mismo `index` que el grupo, por lo tanto con el mismo número de filas y las mismas etiquetas. Por ejemplo puedo, dentro de cada grupo, normalizar los valores respecto a la media y la desviación típica del grupo.
- `filter`: se trata de seleccionar grupos que cumplen un determinado criterio

Consideramos el DataFrame:

```
df = pd.DataFrame(
    {
        "X": ['a', 'a', 'a', 'a', 'b', 'b', 'c', 'c'],
        "Y": np.arange(8),
        "Z": np.arange(8,16)
    }
)
df
```

	X	Y	Z
0	a	0	8
1	a	1	9
2	a	2	10
3	a	3	11
4	b	4	12
5	b	5	13
6	c	6	14
7	c	7	15

Agrupamos según los valores de `X` y aplicamos el método `agg`, pasándole la función para calcular el indicador.

```
df.groupby('X').agg(np.mean)
```

	Y	Z
X		
a	1.5	9.5
b	4.5	12.5
c	6.5	14.5

Ya vimos que se puede obtener el mismo resultado sin usar `agg`

```
df.groupby('X').mean()
```

	Y	Z
X		
a	1.5	9.5
b	4.5	12.5
c	6.5	14.5

Usar `agg` permite, por una parte, aplicar más de una función

```
df.groupby('X').agg([np.mean, np.std])
```

	Y		Z	
	mean	**std**	**mean**	**std**
X				
a	1.5	1.290994	9.5	1.290994
b	4.5	0.707107	12.5	0.707107
c	6.5	0.707107	14.5	0.707107

Por otra parte, permite usar nuestras propias funciones.

Para ilustrarlo, modificamos `df` para introducir datos faltantes

```
df.loc[[0, 2, 5], 'Y'] = np.NaN
df.loc[6, 'Z'] = np.NaN
df
```

	X	Y	Z
0	a	NaN	8.0
1	a	1.0	9.0
2	a	NaN	10.0
3	a	3.0	11.0
4	b	4.0	12.0
5	b	NaN	13.0
6	c	6.0	NaN
7	c	7.0	15.0

Calculamos el número de datos faltantes por columna, desglosándolo por grupos

```
df.groupby('X').agg(lambda x: x.isna().sum())
```

	Y	Z
X		
a	2	0
b	1	0
c	0	1

Cargamos el conjunto de datos de vuelos que salieron en 2013 de uno de los tres aeropuertos de NYC

```
flights = pd.read_feather(path.join(DATA_DIRECTORY, 'flights.feather'))
```

Queremos obtener el número de vuelos cancelados por hora (tienen `NaN` en la columna `dep_time`)

```
flights.groupby('hour').agg(lambda x: x.isna().sum())
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailnum	origin	dest	air_ti
hour															
1.0	0	0	0	1	0	1	1	0	1	0	0	1	0	0	
5.0	0	0	0	9	0	9	9	0	13	0	0	6	0	0	
6.0	0	0	0	425	0	425	454	0	504	0	0	127	0	0	5
7.0	0	0	0	289	0	289	305	0	346	0	0	102	0	0	3
8.0	0	0	0	442	0	442	465	0	508	0	0	161	0	0	5
9.0	0	0	0	327	0	327	343	0	381	0	0	128	0	0	3
10.0	0	0	0	290	0	290	303	0	338	0	0	96	0	0	3
11.0	0	0	0	296	0	296	314	0	344	0	0	61	0	0	3
12.0	0	0	0	388	0	388	404	0	437	0	0	80	0	0	4
13.0	0	0	0	429	0	429	456	0	499	0	0	90	0	0	4
14.0	0	0	0	566	0	566	611	0	684	0	0	170	0	0	6
15.0	0	0	0	670	0	670	733	0	806	0	0	222	0	0	8
16.0	0	0	0	840	0	840	891	0	957	0	0	249	0	0	9
17.0	0	0	0	660	0	660	691	0	759	0	0	150	0	0	7
18.0	0	0	0	626	0	626	667	0	711	0	0	289	0	0	
19.0	0	0	0	861	0	861	898	0	934	0	0	283	0	0	9
20.0	0	0	0	636	0	636	656	0	678	0	0	232	0	0	6
21.0	0	0	0	409	0	409	418	0	430	0	0	62	0	0	4
22.0	0	0	0	78	0	78	81	0	81	0	0	3	0	0	
23.0	0	0	0	13	0	13	13	0	19	0	0	0	0	0	

Así obtenemos el número de valores faltantes para todas las columnas. Como solamente nos interesan las de `dep_time`, modificamos nuestra petición

```
flights.groupby('hour' )['dep_time'].agg(lambda x: x.isna().sum())
```

```
hour
1.0       1
5.0       9
6.0     425
7.0     289
8.0     442
9.0     327
10.0    290
11.0    296
12.0    388
13.0    429
14.0    566
15.0    670
16.0    840
17.0    660
18.0    626
19.0    861
20.0    636
21.0    409
```

```
22.0     78
23.0     13
Name: dep_time, dtype: int64
```

Para aplicar funciones diferentes a diferentes columnas:

Hasta el momento, hemos obtenido los mismos indicadores para las diferentes columnas. Es posible especificar funciones diferentes para distintas columnas, usando un dictionario.

```
df.groupby('X').agg(
    {
        'Y': 'describe',
        'Z': np.mean
    }
)
```

	Y							Z
	count	mean	std	min	25%	50%	75%	max	Z
X									
a	2.0	2.0	1.414214	1.0	1.50	2.0	2.50	3.0	9.5
b	1.0	4.0	NaN	4.0	4.00	4.0	4.00	4.0	12.5
c	2.0	6.5	0.707107	6.0	6.25	6.5	6.75	7.0	15.0