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)

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)

```
In [1]:

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

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, Groupby: split-apply-combine)

```
In [1]:
    import pandas as pd
    import numpy as np
    from os import path

In [2]:
    DATA_DIRECTORY = path.join('...', '...', 'data')
```

Consideremos el siguiente DataFrame:

Consideremos el siguiente DataFrame:

Consideremos el siguiente DataFrame:

Agrupamos según los valores de la columna X

```
In [4]:
    agrupados = df.groupby('X')
```

Podemos iterar un GroupedDataFrame para recorrer los grupos:

Podemos iterar un GroupedDataFrame para recorrer los grupos:

```
In [5]:
```

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

```
Nombre del grupo: a
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

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

In [6]:

agrupados.mean()

Out [6]: x a 1.5 9.5 b 4.5 12.5 c 6.5 14.5 Podemos aplicarle métodos preparados para GroupedDataFrame como mean, sum o describe

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

In [7]:

agrupados.sum()

In [8]:

agrupados.describe()

Out[8]:

								Υ								Z
	count	mean	std	min	25%	50%	75%	max	count	mean	std	min	25%	50%	75%	max
X																
а	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
С	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.

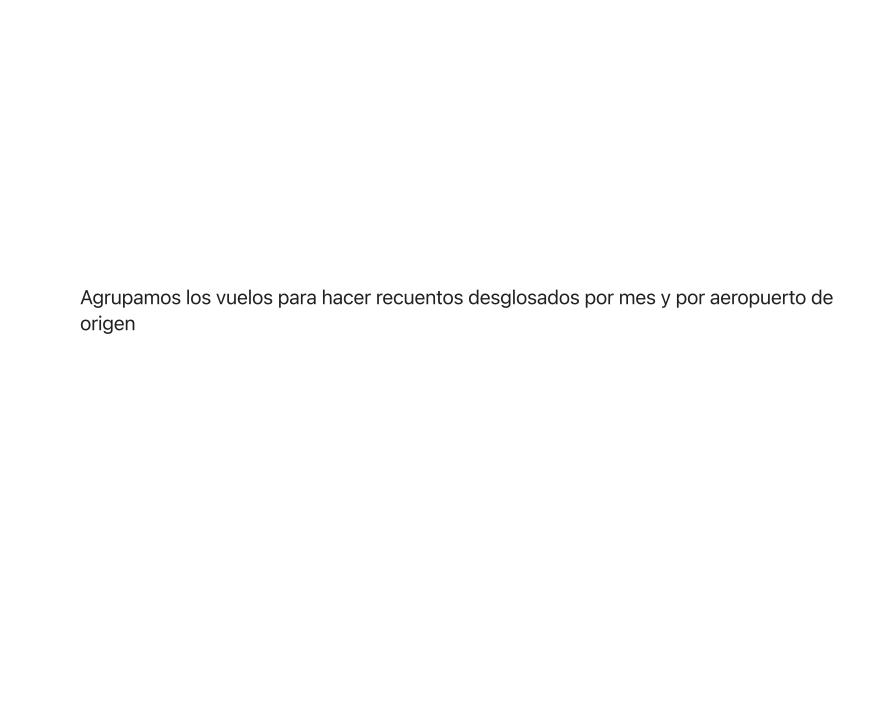
Podemos especificar más de una columna para crear los grupos.

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

In [9]:		lights : lights	= pd.read	d_feat	her(path.jo	in(DATA_DIRECTORY,	'flights.fe	ather'))					
Out[9]:		year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailr
001[9].	0	2013	1	1	517.0	515	2.0	830.0	819	11.0	UA	1545	N14
	1	2013	1	1	533.0	529	4.0	850.0	830	20.0	UA	1714	N24
	2	2013	1	1	542.0	540	2.0	923.0	850	33.0	АА	1141	N61!
	3	2013	1	1	544.0	545	-1.0	1004.0	1022	-18.0	В6	725	N80
	4	2013	1	1	554.0	600	-6.0	812.0	837	-25.0	DL	461	N66{
	•••	•••	•••	•••								•••	

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailr
336771	2013	9	30	NaN	1455	NaN	NaN	1634	NaN	9E	3393	N
336772	2013	9	30	NaN	2200	NaN	NaN	2312	NaN	9E	3525	N
336773	2013	9	30	NaN	1210	NaN	NaN	1330	NaN	MQ	3461	N535
336774	2013	9	30	NaN	1159	NaN	NaN	1344	NaN	MQ	3572	N511
336775	2013	9	30	NaN	840	NaN	NaN	1020	NaN	MQ	3531	N839

336776 rows × 19 columns



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

```
In [10]:
    agrupados = flights.groupby(['month', 'origin'])
```

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

```
In [10]:
    agrupados = flights.groupby(['month', 'origin'])
```

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

```
In [11]:
```

agrupados.size()

Out[11]:

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

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

Para ilustrarlo, cargamos el fichero mompean

```
In [12]:
    mompean = pd.read_csv(path.join(DATA_DIRECTORY, 'mompean.csv'), parse_dates=['FechaHora'], index_col='FechaHora')
    mompean
```

Out[12]:

	NO	NO2	502	03	IMP	HR	NOX	טט	PRB	RS	VV	СбНб	C/H8	XIL	PM10	Ruido
FechaHora																
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

	NO	NO2	SO2	03	TMP	HR	NOX	DD	PRB	RS	VV	C6H6	C7H8	XIL	PM10	Ruido
FechaHora																
•••											•••				•••	
2019-12-31 19:00:00	9.0	35.0	8.0	47.0	NaN	NaN	49.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	19.0	NaN
2019-12-31 20:00:00	29.0	59.0	9.0	24.0	NaN	NaN	102.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	41.0	NaN
2019-12-31 21:00:00	59.0	65.0	8.0	10.0	NaN	NaN	155.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	48.0	NaN
2019-12-31 22:00:00	51.0	51.0	9.0	11.0	NaN	NaN	130.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	45.0	NaN
2019-12-31 23:00:00	32.0	42.0	9.0	7.0	NaN	NaN	90.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	44.0	NaN

96408 rows × 16 columns

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

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

```
In [13]:

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

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

```
In [13]:

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

Hemos usado una función anónima. Aplicamos el método mean

In [14]:

agrupados.mean()

Out[14]:

	NO	NO2	SO2	03	TMP	HR	NOX	DD	PRB	RS	VV	C6H6	C7H8	XIL	PM1
0	9.618014	27.402604	7.516146	53.871709	NaN	NaN	42.020890	NaN	NaN	NaN	NaN	NaN	NaN	NaN	24.41671
1	7.779196	25.304997	7.399091	51.332306	NaN	NaN	37.115698	NaN	NaN	NaN	NaN	NaN	NaN	NaN	22.79053
2	6.599348	22.842077	7.323167	49.531390	NaN	NaN	32.844795	NaN	NaN	NaN	NaN	NaN	NaN	NaN	21.68734
3	4.858891	19.067700	7.234885	48.993829	NaN	NaN	26.410005	NaN	NaN	NaN	NaN	NaN	NaN	NaN	20.11948
4	4.147523	16.349483	7.184957	48.060045	NaN	NaN	22.607512	NaN	NaN	NaN	NaN	NaN	NaN	NaN	18.80239
5	4.095316	15.585240	7.150509	45.970250	NaN	NaN	21.786220	NaN	NaN	NaN	NaN	NaN	NaN	NaN	17.87460
6	5.992364	18.083174	7.177069	41.162830	NaN	NaN	27.149168	NaN	NaN	NaN	NaN	NaN	NaN	NaN	17.90737
7	12.616621	25.141144	7.252747	33.026404	NaN	NaN	44.355586	NaN	NaN	NaN	NaN	NaN	NaN	NaN	20.48462
8	18.700545	30.695095	7.376575	27.968004	NaN	NaN	59.242234	NaN	NaN	NaN	NaN	NaN	NaN	NaN	24.91990
9	34.211874	36.715686	7.711188	26.932959	NaN	NaN	89.038399	NaN	NaN	NaN	NaN	NaN	NaN	NaN	32.86816
10	26.159482	32.184573	7.810072	39.895430	NaN	NaN	72.175290	NaN	NaN	NaN	NaN	NaN	NaN	NaN	34.70270
11	13.300164	25.110716	7.697393	55.724283	NaN	NaN	45.379716	NaN	NaN	NaN	NaN	NaN	NaN	NaN	29.69671
12	9.329697	22.087810	7.825563	65.378219	NaN	NaN	36.264249	NaN	NaN	NaN	NaN	NaN	NaN	NaN	27.03578
13	8.356833	20.896150	7.995197	71.142339	NaN	NaN	33.565618	NaN	NaN	NaN	NaN	NaN	NaN	NaN	25.78769
14	7.735246	19.927721	8.148641	75.562849	NaN	NaN	31.660531	NaN	NaN	NaN	NaN	NaN	NaN	NaN	24.94457
15	7.167794	18.990257	8.609205	78.299637	NaN	NaN	29.848714	NaN	NaN	NaN	NaN	NaN	NaN	NaN	23.91573
16	6.474696	17.648714	8.834398	80.232617	NaN	NaN	27.457645	NaN	NaN	NaN	NaN	NaN	NaN	NaN	23.57935
17	6.404491	18.028950	9.035134	80.065624	NaN	NaN	27.702922	NaN	NaN	NaN	NaN	NaN	NaN	NaN	24.36759
18	6.553600	20.003249	9.063149	77.836409	NaN	NaN	29.928803	NaN	NaN	NaN	NaN	NaN	NaN	NaN	24.70858
19	7.001896	23.205038	8.684084	73.640817	NaN	NaN	33.828548	NaN	NaN	NaN	NaN	NaN	NaN	NaN	24.99345
20	8.330986	25.681203	8.342933	69.847144	NaN	NaN	38.321506	NaN	NaN	NaN	NaN	NaN	NaN	NaN	25.50669
21	10.382592	28.194685	7.983729	65.330907	NaN	NaN	43.979393	NaN	NaN	NaN	NaN	NaN	NaN	NaN	26.55518
22	11.746273	29.620222	7.777807	60.919373	NaN	NaN	47.500678	NaN	NaN	NaN	NaN	NaN	NaN	NaN	26.78843
23	12.008955	29.771235	7.646823	56.430933	NaN	NaN	48.046676	NaN	NaN	NaN	NaN	NaN	NaN	NaN	25.94912

Pasamos una lista de funciones

Pasamos una lista de funciones

```
In [15]:
    agrupados = mompean.groupby([lambda x: x.dayofweek, lambda x: x.hour])
```

Pasamos una lista de funciones

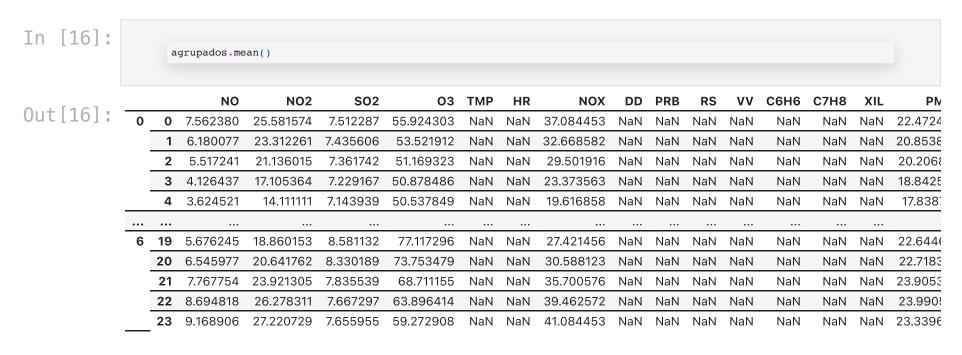
```
In [15]:
    agrupados = mompean.groupby([lambda x: x.dayofweek, lambda x: x.hour])
```

Calculamos la media desglosada por grupo:

Pasamos una lista de funciones

```
In [15]:
    agrupados = mompean.groupby([lambda x: x.dayofweek, lambda x: x.hour])
```

Calculamos la media desglosada por grupo:



168 rows × 16 columns