

# 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 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 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 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
```

In [2]:

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

Consideremos el siguiente DataFrame:

# Consideremos el siguiente DataFrame:

In [3]:

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

Out[3]:

	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

# Consideremos el siguiente DataFrame:

In [3]:

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

Out [3]:

	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

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

	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`

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

In [6]:

```
agrupados.mean()
```

Out[6]:

	Y	Z
<b>x</b>		
<b>a</b>	1.5	9.5
<b>b</b>	4.5	12.5
<b>c</b>	6.5	14.5

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

In [6]:

```
agrupados.mean()
```

Out[6]:

	Y	Z
x		
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.

In [7]:

```
agrupados.sum()
```

Out[7]:

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

In [8]:

```
agrupados.describe()
```

Out[8]:

	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.



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]:

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

Out [9] :

[illegible]



	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

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]:

[illegible]

	NO	NO2	SO2	O3	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 `datetime`

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 `datetime`

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 `datetime`

In [13]:

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

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



# Podemos aplicar más de una función en `groupby`

Pasamos una lista de funciones





# Podemos aplicar más de una función en `groupby`

Pasamos una lista de funciones

In [15]:

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



# Podemos aplicar más de una función en `groupby`

Pasamos una lista de funciones

In [15]:

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

Calculamos la media desglosada por grupo:



# Podemos aplicar más de una función en `groupby`

Pasamos una lista de funciones

In [15]:

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

Calculamos la media desglosada por grupo:

In [16]:

```
agrupados.mean()
```

Out[16]:

		NO	NO2	SO2	O3	TMP	HR	NOX	DD	PRB	RS	VV	C6H6	C7H8	XIL	PM
0	0	7.562380	25.581574	7.512287	55.924303	NaN	NaN	37.084453	NaN	NaN	NaN	NaN	NaN	NaN	NaN	22.4724
	1	6.180077	23.312261	7.435606	53.521912	NaN	NaN	32.668582	NaN	NaN	NaN	NaN	NaN	NaN	NaN	20.8538
	2	5.517241	21.136015	7.361742	51.169323	NaN	NaN	29.501916	NaN	NaN	NaN	NaN	NaN	NaN	NaN	20.2068
	3	4.126437	17.105364	7.229167	50.878486	NaN	NaN	23.373563	NaN	NaN	NaN	NaN	NaN	NaN	NaN	18.8425
	4	3.624521	14.111111	7.143939	50.537849	NaN	NaN	19.616858	NaN	NaN	NaN	NaN	NaN	NaN	NaN	17.8387
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6	19	5.676245	18.860153	8.581132	77.117296	NaN	NaN	27.421456	NaN	NaN	NaN	NaN	NaN	NaN	NaN	22.6446
	20	6.545977	20.641762	8.330189	73.753479	NaN	NaN	30.588123	NaN	NaN	NaN	NaN	NaN	NaN	NaN	22.7183
	21	7.767754	23.921305	7.835539	68.711155	NaN	NaN	35.700576	NaN	NaN	NaN	NaN	NaN	NaN	NaN	23.9053
	22	8.694818	26.278311	7.667297	63.896414	NaN	NaN	39.462572	NaN	NaN	NaN	NaN	NaN	NaN	NaN	23.9901
	23	9.168906	27.220729	7.655955	59.272908	NaN	NaN	41.084453	NaN	NaN	NaN	NaN	NaN	NaN	NaN	23.3396

168 rows × 16 columns

