Usar `transform` sobre `GroupedDataFrame`s

El método `transform` permite 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.

Preliminares

In [1]:

```
import pandas as pd
import numpy as np
```

Consideramos el DataFrame:

In [2]:

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

Out[2]:

	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`. En el vídeo anterior vimos cómo aplicar el método `agg`, pasándole la función para calcular el indicador.

In [3]:

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

Out[3]:

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

Hagamos lo mismo pero usando `transform`

In [4]:

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

Out[4]:

	Y	Z
0	1.5	9.5
1	1.5	9.5
2	1.5	9.5
3	1.5	9.5
4	4.5	12.5
5	4.5	12.5
6	6.5	14.5
7	6.5	14.5

Se puede añadir las columnas proporcionadas por `transform` a nuestro DataFrame inicial

In [5]:

```
# Creamos una copia del DataFrame original
df_extended = df.copy()
```

Añadimos las columnas que contienen las medias de Y y Z desglosadas por grupo

In [6]:

```
df_extended[['media_grupo_Y', 'media_grupo_Z']] = (
    df.groupby('X').transform(np.mean)
)
df_extended
```

Out[6]:

	X	Y	Z	media_grupo_Y	media_grupo_Z
0	a	0	8	1.5	9.5
1	a	1	9	1.5	9.5
2	a	2	10	1.5	9.5
3	a	3	11	1.5	9.5
4	b	4	12	4.5	12.5
5	b	5	13	4.5	12.5
6	c	6	14	6.5	14.5
7	c	7	15	6.5	14.5

Ejemplo de uso: normalización de columnas

Vamos ahora a llevar a cabo la normalización de los valores de Y y Z, teniendo en cuenta los grupos formados por los valores de X.

In [7]:

```
# Creamos una copia del DataFrame original
df_normalizado = df.copy()
```

In [8]:

```
(df_normalizado
 .groupby('X')
 .transform(lambda x: (x - x.mean()) / x.std())
```

Out[8]:

	Y	Z
0	-1.161895	-1.161895
1	-0.387298	-0.387298
2	0.387298	0.387298
3	1.161895	1.161895
4	-0.707107	-0.707107
5	0.707107	0.707107
6	-0.707107	-0.707107
7	0.707107	0.707107

Podemos sustituir las columnas Y y Z por sus equivalentes normalizados, por grupo.

In [9]:

```
df_normalizado[['Y', 'Z']] = df_normalizado.groupby('X').transform(
    lambda x: (x - x.mean()) / x.std()
)
df_normalizado
```

Out[9]:

	X	Y	Z
0	a	-1.161895	-1.161895
1	a	-0.387298	-0.387298
2	a	0.387298	0.387298
3	a	1.161895	1.161895
4	b	-0.707107	-0.707107
5	b	0.707107	0.707107
6	c	-0.707107	-0.707107
7	c	0.707107	0.707107

Sustitución de valores faltantes por la media de su grupo

Modificamos `df` para introducir valores faltantes en sus columnas:

In [10]:

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

Out[10]:

	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

Vamos a pasar a `transform` una función que tomando una columna, recorra sus elementos, si el elemento no falta, lo deja intacto. Si el elemento falta, lo sustituye por la media de la columna del grupo al que pertenece.

Para ello, vamos a usar el método `where`, ver **referencia**

> *`where` admite como parámetro un vector booleano y un vector `other`. Si el elemento del vector booleano es `True`, deja intacto el elemento de la columna a la que se aplica. Si es `False`, usa el elemento correspondiente del vector `other`.*

Ejemplo: aplicamos el método `where` a la columna Y, para sustituir los valores faltantes por el valor 1000.

In [11]:

```
# Usamos ~ para negar un vector booleano, True se vuelve False y viceversa
df['Y'].where( ~df['Y'].isna(), 1000)
```

Out[11]:

```
0    1000.0
1       1.0
2    1000.0
3       3.0
4       4.0
5    1000.0
6       6.0
7       7.0
Name: Y, dtype: float64
```

Ahora podemos sustituir los valores faltantes de Y y Z por la media de su grupo

In [12]:

```
df.groupby('X').transform(lambda x: x.where(~x.isna(), x.mean()))
```

Out[12]:

	Y	Z
0	2.0	8.0
1	1.0	9.0
2	2.0	10.0
3	3.0	11.0
4	4.0	12.0
5	4.0	13.0
6	6.0	15.0
7	7.0	15.0