PROPUESTA DE SOLUCIÓN BACKEND

Laura Salguero Vidal

2023

He realizado el código referente a los ejercicios del perfil backend. Antes de comenzar a trabajar con los reportes, he realizado un pequeño esquema explicativo de qué relación hay entre los atributos de cada fichero de datos, como se muestra en la imagen de la Figura 1.

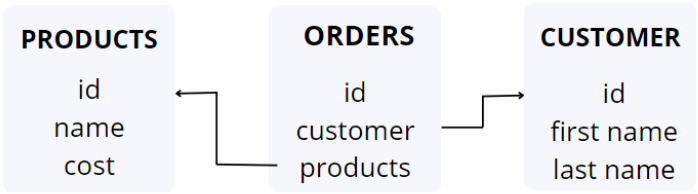

Figura 1

Una vez aclaradas las relaciones que conectan los datos, es hora de trabajar para obtener los tres reportes.

REPORT 1

El objetivo es conseguir como resultado final un fichero de datos que contenga información sobre el precio total de cada pedido junto con su id correspondiente. Para ello, he trabajado con los datos de “products.csv” y “orders.csv”.

He recorrido las filas de este último fichero, dividiendo el contenido de la columna *products* que nos indica qué productos han comprado y su cantidad. Para el cálculo, he tenido que crear un diccionario llamado *quanti* con el recuento de cada producto para así multiplicarlo por el precio y obtener el coste final. Por ejemplo, para el pedido con id = 0, su diccionario *quanti* asociado sería: `{1: 2, 0: 2}`

Una vez finalizado este cálculo, se añade al diccionario vacío *dic_order_prices* tanto el id de ese pedido como el resultado, de manera que al convertirlo en DataFrame obtengamos el fichero que deseamos.

REPORT 2

Para el segundo reporte queremos saber qué cliente ha comprado cada producto. Esto significa que utilizaremos la información de “products.csv” y “orders.csv”.

Primero generamos un diccionario que de entrada ya tiene como claves los productos y como valor asociado a cada producto una lista vacía [1]. Observamos dentro de “orders.csv” qué consumidor ha comprado cada producto, utilizando para ello un conjunto (*set*) en el atributo *products* ya que tenemos elementos repetidos que no nos dan información adicional relevante y así se agiliza el proceso. El contenido final de este diccionario es una lista de identificadores de clientes asociados a cada producto, como se puede ver en la Figura 2.

```
0 [0, 22, 20, 28, 40, 32, 5, 45, 37, 38, 6, 44, 50, 24, 54, 59, 15, 21, 34, 5, 34, 19, 47, 48, 46, 24, 10, 17, 22, 29, 44]
1 [0, 22, 40, 32, 45, 38, 51, 6, 44, 34, 3, 50, 24, 15, 34, 5, 45, 41, 47, 46, 32, 35, 29, 24, 10, 17, 9, 22, 5, 44, 58]
2 [22, 57, 20, 40, 5, 45, 37, 51, 6, 44, 54, 8, 15, 21, 5, 15, 41, 48, 46, 29, 44, 24, 17, 9, 29, 5, 44]
3 [22, 20, 28, 32, 5, 38, 51, 34, 50, 24, 54, 21, 34, 34, 41, 47, 46, 44, 32, 29, 24, 17, 9, 22, 29, 44]
4 [22, 57, 20, 28, 51, 24, 59, 36, 8, 15, 21, 5, 34, 19, 41, 44, 32, 35, 44, 24, 10, 17, 9, 22]
5 [22, 57, 28, 32, 5, 37, 38, 6, 44, 34, 3, 50, 24, 54, 59, 15, 25, 21, 34, 5, 34, 45, 19, 41, 47, 44, 38, 32, 44, 10, 17, 9, 2
2, 29, 44]
```

Figura 2

La lista ids contiene identificadores de los productos, pero para que se correspondan en el DataFrame con los clientes correspondientes, he debido añadir el id de cada producto tantas veces como clientes tiene. Con la lista customer_ids únicamente he unido todas las listas para que estén una seguida detrás de otra y a la hora de crear el DataFrame se correspondan con su clave (id de producto).

Cabe destacar que para este último paso debemos tener en cuenta que los diccionarios no tienen por qué estar ordenados según el orden de creación o cualquier otro criterio como el orden alfabético o numérico. Esto quiere decir que no se generaría de forma correcta el DataFrame ya que, al concatenar las listas, que son los valores del diccionario, no tienen por qué corresponderse con el orden original en el que se creó. Sin embargo, consultando las actualizaciones de Python, a partir de la versión 3.7, los diccionarios guardan el orden en el que se insertaron los elementos, y como en mi caso he creado un diccionario con las claves predeterminadas, este orden se conservará a la hora de realizar estos últimos bucles sobre las claves y los valores del diccionario. [2]

Para finalizar con este ejercicio, añadimos cada lista a un diccionario llamado *dic_order_prices* para posteriormente crear un DataFrame.

REPORT 3

Para este último reporte necesitamos información de “orders.csv”, “customers.csv” y “order_prices.csv”, que es el fichero que hemos creado en el primer reporte. He realizado un *merge* [3] sobre estos dos ficheros utilizando como clave los id de los pedidos y he eliminado los atributos que no me interesaba utilizar para el reporte. El contenido de “orders_with_prices” son dos columnas, una con los ids del cliente y otra con el total que ha gastado en un pedido. Cabe destacar que en este DataFrame hay claves repetidas ya que un mismo cliente puede haber realizado distintos pedidos y en ese caso su id aparecerá más de una vez con su gasto total en dicho pedido.

Para obtener el gasto total en todos los pedidos de un mismo cliente, hemos agrupado por el id de cliente y aplicado el método *sum*. Además, a la hora de generar el nuevo DataFrame con esta información, Python utiliza el id como índice de las filas. Para que esto no ocurra, he utilizado el método *reset_index* [4], ya que de lo contrario no podría trabajar con los ids como si fueran un atributo más.

Para finalizar, hacemos un *merge* con “customers.csv” para poder obtener el nombre y apellido de los clientes y ordenamos de forma descendente para conseguir el ranking deseado [5].

Como podemos observar, hay pedidos de los cuales no podemos obtener cuál ha sido el cliente que ha comprado dichos productos ya que en orders no tenemos dicha información. Un ejemplo puede ser el cliente con id (columna customer) = 2, que no figura en ninguno de los pedidos en orders como se puede observar en la Figura 3.

```
orders.sort_values('customer').head()
```

	id	customer	products
0	0	0	1 0 1 0
15	15	3	1 1 5 5 1
47	47	5	2 1
7	7	5	5 0 3 2
26	26	5	2 0 4 4 5 4 2 1 2 1

Figura 3

Para finalizar, propongo como mejora el siguiente código, que es para que el usuario seleccione desde el mismo programa cuál es el csv que desea abrir a la hora de ejecutar el código.

```
import pandas as pd
input_products = input("Introduzca el csv con informacion referente a los productos: ")
products = pd.read_csv(input_products, header=0)

input_customers = input("Introduzca el csv con informacion referente a los clientes: ")
customers = pd.read_csv(input_customers, header=0)

input_orders = input("Introduzca el csv con informacion referente a los pedidos: ")
orders = pd.read_csv(input_orders, header=0)
```

He decidido no incluirlo como solución final ya que en caso de que por algún motivo los nombres de las variables no se correspondan con los ficheros para los que se ha creado el código, este puede no funcionar correctamente. En el código propuesto el usuario debe introducir el nombre del fichero con la extensión .csv, de lo contrario, no funcionará.

```
Introduzca el csv con informacion referente a los productos: products.csv
```

El código inicialmente fue creado desde Jupyter ya que para mí era más cómo trabajar desde un notebook donde realizar pruebas y solucionar los errores es visualmente más fácil. Sin embargo, he entregado el código de forma que se pueda ejecutar todo de una en Spyder ya que en el notebook hay que ir celda a celda y es más tedioso. El código final solo requiere tener los tres ficheros de datos “customers.csv”, “orders.csv” y “products.csv”.

Bibliografía

[1] «Compresión de diccionarios,» [En línea]. Available: https://www.freecodecamp.org/espanol/news/compresion-de-diccionario-en-python-explicado-con-ejemplos/.

[2] «Orden de los diccionarios,» [En línea]. Available: https://tutorialpython.com/listas-en-python/#:~:text=A%20partir%20de%20la%20versi%C3%B3n,%C2%BB%2C%3E%3E%3E%7D..

[3] «Combinar dataframes,» [En línea]. Available: https://www.analyticslane.com/2018/09/10/unir-y-combinar-dataframes-con-pandas-en-python/.

[4] «Método reset_index,» [En línea]. Available: https://statologos.com/pandas-groupby-agrega-multiples-columnas/.

[5] «Ordenar DataFrame,» [En línea]. Available: https://www.analyticslane.com/2019/04/29/diferentes-formas-de-ordenar-dataframes-en-pandas/.