

Fastbook 07

Programación en R

Trabajando con dataframes (II)



07. Trabajando con dataframes (II)



Poco a poco nos vamos acercando al final de la asignatura. En el presente fastbook, que se define como una continuación del anterior, vamos a seguir explorando diferentes transformaciones de dataframes que el paquete **dplyr** nos permite llevar a cabo.

En concreto, vamos a abordar dos tareas que presentan gran importancia a la hora de trabajar con datos: la agregación de información y el cruce de diferentes tablas.

¡Allá vamos!

Autor: Juan Jiménez García

[Agregados de datos](#)

[Unión de dataframes](#)

[Conclusiones](#)

Agregados de datos

X Edix Educación

Vamos a empezar viendo qué es lo que se entiende por **nivel de agregación del dato**.

El nivel de agregación de una información hace referencia al nivel de detalle y granularidad que presenta la definición de cada muestra.

Pongamos como ejemplo una situación en la que trabajamos con el número de clicks que genera un determinado anuncio online. Podríamos encontrar la contabilización con distintas granularidades temporales (por minuto, por hora, por día, por mes, etc.). Cada uno de esos casos se corresponde con un nivel de agregación de la información distinto.

Supongamos ahora que el nivel de granularidad es por minuto, mientras que a nosotros nos interesa el número de clicks que se generan cada día. Para llegar a esa información (pasar de minuto a día) debemos **agregar el dato**.

Agregar un determinado dato o información consiste en **reducir el nivel de detalle y granularidad** que presenta la definición de cada muestra.

Con el ejemplo que estamos tratando se puede entender muy bien. Pasamos de tener el número de clicks que se generan cada minuto a tener el número de clicks que se generan cada día.

Para ayudar con el aterrizaje de este concepto, vamos a presentar de forma visual algunos **ejemplos de agregación de información**.

Ejemplo 1

Partimos de un experimento en el que **se compara la eficacia de dos anuncios en formato online**. El primero de ellos es el tradicional, el que ha sido usado por la empresa desde sus inicios. El segundo de ellos es una versión mejorada que se ha creado recientemente.

Para ello, tenemos el dato de clicks que ha generado diariamente cada una de las versiones entre los meses de enero y marzo de 2021.

Identificación		Atributos
Fecha	Version	Clicks
01/01/2021	Antigua	347
01/01/2021	Actualizada	423
02/01/2021	Antigua	335
02/01/2021	Actualizada	465
⋮	⋮	⋮
31/03/2021	Antigua	321
31/03/2021	Actualizada	578

Si nuestro objetivo es conocer el número de clicks totales que ha generado cada una de las versiones, debemos agregar la información a través de la columna *Version* y sumar los valores de la columna *Clicks*, obteniendo el siguiente resultado.

Identificación		Atributos	
Version		Clicks	
Antigua		36,341	
Actualizada		40,521	

Ejemplo 2

Partimos del dato del mismo experimento. Imaginemos ahora que nos interesase observar cuál ha sido el número de clicks que se ha generado en cada mes teniendo en cuenta los dos anuncios.

En primer lugar, tal y como se observa en la siguiente imagen, necesitamos construir (a partir de la fecha) una columna que nos identifique el mes de cada muestra.

Identificación		Atributos	
Fecha	Mes	Version	Clicks
01/01/2021	01/01/2021	Antigua	347
01/01/2021	01/01/2021	Actualizada	423
02/01/2021	01/01/2021	Antigua	335
02/01/2021	01/01/2021	Actualizada	465
⋮	⋮	⋮	⋮
31/03/2021	01/03/2021	Antigua	321
31/03/2021	01/03/2021	Actualizada	578

Tras ello, agregamos por la columna Mes sumando los valores de la columna Clicks.
Obtenemos el siguiente resultado.

Identificación		Atributos	
Mes		Clicks	
01/01/2021		27,136	
01/02/2021		24,110	
01/03/2021		25,616	

Ejemplo 3

Supongamos, ahora, que nos interesa observar cuál ha sido el número de clicks que se ha generado en cada mes y en cada anuncio (queremos reducir la granularidad temporal de diario a mensual).

Al igual que en el ejemplo anterior, necesitamos construir (a partir de la fecha) una columna que nos identifique el mes de cada muestra.

Identificación			Atributos	
Fecha	Mes	Version	Clicks	
01/01/2021	01/01/2021	Antigua	347	
01/01/2021	01/01/2021	Actualizada	423	
02/01/2021	01/01/2021	Antigua	335	
02/01/2021	01/01/2021	Actualizada	465	
⋮	⋮	⋮	⋮	
31/03/2021	01/03/2021	Antigua	321	
31/03/2021	01/03/2021	Actualizada	578	

Tras ello, agregamos por las columnas Mes y Version sumando los valores de la columna Clicks. Obtenemos el siguiente resultado.

Identificación		Atributos
Mes	Version	Clicks
01/01/2021	Antigua	12,770
01/01/2021	Actualizada	14,366
01/02/2021	Antigua	11,387
01/02/2021	Actualizada	12,723
01/03/2021	Antigua	12,184
01/03/2021	Actualizada	13,432

Ejemplo 4

Hasta ahora solo se han calculado sumas a la hora de agregar, pero, ¿qué pasa si queremos calcular la media diaria de clicks de cada una de las versiones?

En ese caso, tenemos que agregar por la columna Version y calcular la media de la columna Clicks. Obtenemos el siguiente resultado.

Identificación	Atributos
Version	Clicks
Antigua	404
Actualizada	450

Dicho de una forma sencilla, la agregación de datos nos permite **obtener una visión resumida** (con menor nivel de detalle) de nuestra información de partida.

Como ya habéis observado, dicho resumen no tiene por qué corresponderse con una suma, puede ser una media, la mediana, el máximo o cualquier tipo de operación que podamos aplicar sobre un vector (un conjunto de elementos).

Tras habernos introducido en el concepto de agregación del dato desde una perspectiva teórica, es hora de pasar a R y entender cómo podemos llevar a cabo estos procedimientos desde la programación.

Como ya os he adelantado, vamos a hacer uso del paquete **dplyr**, y en concreto de sus funciones **group_by** y **summarise**.

La función **group_by()** crea **grupos de filas** (muestras) en base a los valores que presenten en la columna o columnas que se hayan seleccionado. Las muestras que tienen el mismo valor en esas columnas van en el mismo grupo.

Función *group_by()*

Parámetro	Clase	Definición
data	dataframe	Dataframe origen que queremos dividir en grupos.
...	vector	Columna o columnas que queremos usar para construir los grupos.

La función *summarise()* está diseñada para aplicar los cálculos, funciones y transformaciones que deseemos sobre cada uno de los grupos que hemos creado previamente con la función *group_by()*.

Entre sus paréntesis tendremos que definir qué columnas va a tener el dataframe de salida (métricas resumen) y cómo se construyen a partir de las columnas de entrada. En este aspecto, su sintaxis recuerda a la de la función *mutate()*.

Función *summarise()*

Parámetro	Clase	Definición
data	dataframe	Dataframe sobre el que aplicaremos las correspondientes funciones de forma aislada para cada uno de sus grupos.
...	...	Nombre y definición de las columnas que tendrá el nuevo dataframe (sintaxis similar a <i>mutate</i>).

A continuación, vamos a llevar a código las cuatro agregaciones que hemos visto en los ejemplos. En todos ellos partimos del dataframe df, que contiene los clicks diarios generados por cada versión del anuncio (tal y como muestra la siguiente imagen).

Identificación		Atributos
Fecha	Version	Clicks
01/01/2021	Antigua	347
01/01/2021	Actualizada	423
02/01/2021	Antigua	335
02/01/2021	Actualizada	465
⋮	⋮	⋮
31/03/2021	Antigua	321
31/03/2021	Actualizada	578

EJEMPLO 1	EJEMPLO 2	EJEMPLO 3	EJEMPLO 4

Si nuestro objetivo es conocer el número de clicks totales que ha generado cada una de las versiones debemos agrupar la información por la columna Version y sumar los valores de la columna Clicks.

```
df %>% group_by(Version) %>%
  summarise(Clicks = sum(Clicks))
```

EJEMPLO 1

EJEMPLO 2

EJEMPLO 3

EJEMPLO 4

Imaginemos ahora que nos interesase observar cuál ha sido el número de clicks que se han generado en cada mes teniendo en cuenta ambos anuncios.

En primer lugar creamos la columna Mes a partir de la columna Fecha haciendo uso de la función `floor_date()` del paquete lubridate. Tras ello agrupamos por Mes y sumamos Clicks.

```
df %>% mutate(Mes = floor_date(Fecha, unit="month")) %>%
  group_by(Mes) %>%
  summarise(Clicks = sum(Clicks))
```

EJEMPLO 1

EJEMPLO 2

EJEMPLO 3

EJEMPLO 4

Supongamos ahora que nos interesa observar cuál ha sido el número de clicks que se ha generado en cada mes y en cada anuncio (queremos reducir la granularidad temporal de diario a mensual). En primer lugar creamos la columna Mes a partir de la columna Fecha haciendo uso de la función `floor_date()` del paquete lubridate. Tras ello agrupamos por Mes y Version y sumamos Clicks.

```
df %>% mutate(Mes = floor_date(Fecha, unit="month")) %>%
  group_by(Mes, Version) %>%
  summarise(Clicks = sum(Clicks))
```

EJEMPLO 1	EJEMPLO 2	EJEMPLO 3	EJEMPLO 4
<p>Por último, calculamos la media diaria de clicks generados por cada una de las versiones. Agrupamos por la columna Version y calculamos la media de clicks.</p> <pre>df %>% group_by(Version) %>% summarise(Clicks = mean(Clicks))</pre>			

Como veis, en términos de código, realizar este tipo de transformaciones resulta muy sencillo.

Aquí os comarto un conjunto de funciones que resultan interesantes de cara a realizar agregaciones (algunas de ellas ya os sonarán). Podemos utilizarlas dentro del *summarise* tal y como hemos usado *sum()* y *mean()* en los ejemplos.

Tipo	Función	Descripción
Básica	sum()	Calcula la suma de un vector.
	mean()	Calcula la media de un vector.
	median()	Calcula la mediana de un vector.
Variabilidad	sd()	Calcula la desviación estándar (o típica) de un vector.
	IQR()	Calcula el rango intercuartílico de un vector.

Tipo	Función	Descripción
Rango	min()	Calcula el mínimo de un vector.
	max()	Calcula el máximo de un vector.
	quantile()	Calcula el quantil seleccionado de un vector.
Posición	first()	Devuelve el primer elemento del vector.
	last()	Devuelve el último elemento del vector.
	nth()	Devuelve el elemento del vector que se encuentra en la posición seleccionada.
Conteo	n()	Devuelve el número de elementos que contiene un vector (solo puede usarse dentro de <i>summarise</i> , <i>mutate</i> y <i>filter</i>).
	n_distinct()	Devuelve el número de elementos distintos que contiene el vector (solo puede usarse dentro de <i>summarise</i> , <i>mutate</i> y <i>filter</i>).

Unión de dataframes

 Edix Educación

Al inicio de la mayoría de los proyectos de analítica, las fuentes de datos no se encuentran unificadas. Normalmente la información a usar está distribuida en diferentes tablas, que deberemos unir y limpiar de forma adecuada para poder construir cualquier tipo de análisis o modelo.

Las operaciones de unión nos permiten combinar la información que se encuentra almacenada en dos tablas (en nuestro caso, dataframes).

Si hacéis memoria, recordaréis que en temas anteriores ya se abordaron dos funciones que nos permiten unir dataframes, *cbind()* y *rbind()*.

rbind()



Si tenemos dos dataframes que contienen distintas muestras (filas) con las mismas variables (columnas), podemos pegarlos uno encima del otro con esta función.

Resulta útil cuando queremos ampliar la muestra de nuestro dataframe (añadir nuevas observaciones).

cbind()

Si tenemos dos dataframes que contienen las mismas muestras (filas) con distintas variables (columnas), podemos pegarlos uno al lado del otro con esta función.

En la práctica, su uso se encuentra bastante limitado e incluso puede considerarse poco recomendable, debido a que la muestra de ambas tablas debe ser idéntica y estar ordenada. En la mayoría de los casos, esta condición no se cumple e implica mayor complejidad y problemas. Por esta razón, resulta adecuado moverse hacia otras soluciones.

La operación de *join* (también conocida como *merge*) es un tipo de unión que permite combinar con precisión y flexibilidad el dato que se encuentra almacenado en dos tablas separadas (en nuestro caso, dataframes). Para ello trabaja con ciertos parámetros que nos ayudan a definir cómo debe llevarse a cabo el cruce de información.

Sigamos con el ejemplo de experimentación que venimos trabajando. Tal y como muestra la siguiente imagen tenemos dos tablas:

1. Tabla de clicks: contiene los clicks diarios generados por cada versión del anuncio en los meses de enero, febrero y marzo de 2021.
2. Tabla de impresiones: contiene las impresiones diarias de cada versión del anuncio en los meses de febrero, marzo y abril de 2021.

TABLA DE CLICKS			TABLA DE IMPRESIONES		
Identificación		Atributos	Identificación		Atributos
Fecha	Version	Clicks	Fecha	Version	Impresiones
01/01/2021	Antigua	347	01/02/2021	Antigua	3541
01/01/2021	Actualizada	423	01/02/2021	Actualizada	3623
...
01/02/2021	Antigua	295	01/03/2021	Antigua	4576
01/02/2021	Actualizada	341	01/03/2021	Actualizada	4650
...
31/03/2021	Antigua	321	30/04/2021	Antigua	3212
31/03/2021	Actualizada	578	30/04/2021	Actualizada	3091

Como podéis observar, existe un **desalineamiento muestral** (en este caso, temporal) entre las dos tablas. Está claro que pegar una al lado de la otra usando `cbind()` no nos llevaría a una solución correcta.

Para hacerlo de forma adecuada, el cruce debe tener en cuenta las variables en común (*Fecha y Version*) para que la unión se realice cuando esos valores coincidan (los clicks de la versión antigua del 1 de febrero deben ir con las impresiones de la versión antigua del 1 de febrero, y así sucesivamente).

Teniendo claro este eje, estamos en disposición de definir los cuatro tipos de joins que existen.

1

Inner join

El *inner join* devuelve solo aquellas **filas en las que hay match**. Es decir, para que una muestra aparezca en la tabla de salida, debe existir en las dos tablas de entrada. En nuestro ejemplo, el resultado contendría únicamente las observaciones de febrero y marzo, tal y como muestra la siguiente imagen.

Identificación		Atributos	
Fecha	Version	Clicks	Impresiones
01/02/2021	Antigua	295	3541
01/02/2021	Actualizada	341	3623
...
31/03/2021	Antigua	321	4416
31/03/2021	Actualizada	578	4752

2

Left join

El *left join* devuelve solo aquellas **filas que aparecen en la tabla situada a la izquierda**, independientemente de que existan o no en la tabla de la derecha. Es decir, para que una muestra aparezca en la tabla de salida, debe existir en la tabla izquierda. En nuestro ejemplo, el resultado contendría únicamente las observaciones de enero, febrero y marzo, tal y como muestra la siguiente imagen.

Identificación		Atributos	
Fecha	Version	Clicks	Impresiones
01/01/2021	Antigua	347	NA
01/01/2021	Actualizada	423	NA
...
01/02/2021	Antigua	295	3541
01/02/2021	Actualizada	341	3623
...
31/03/2021	Antigua	321	4416
31/03/2021	Actualizada	578	4752

Las impresiones adquieren valor NA en aquellas muestras que no existen en la tabla derecha.

3

Right join

El *right join* devuelve solo aquellas **filas que aparecen en la tabla situada a la derecha**, independientemente de que existan o no en la tabla de la izquierda. Es decir, para que una muestra aparezca en la tabla de salida, debe existir en tabla derecha. En nuestro ejemplo, el resultado contendría únicamente las observaciones de febrero, marzo y abril, tal y como muestra la siguiente imagen.

Identificación		Atributos	
Fecha	Version	Clicks	Impresiones
01/02/2021	Antigua	295	3541
01/02/2021	Actualizada	341	3623
⋮	⋮	⋮	⋮
01/03/2021	Antigua	312	4576
01/03/2021	Actualizada	427	4650
⋮	⋮	⋮	⋮
30/04/2021	Antigua	NA	3212
30/04/2021	Actualizada	NA	3091

Los clicks adquieren valor NA en aquellas muestras que no existen en la tabla izquierda.

4

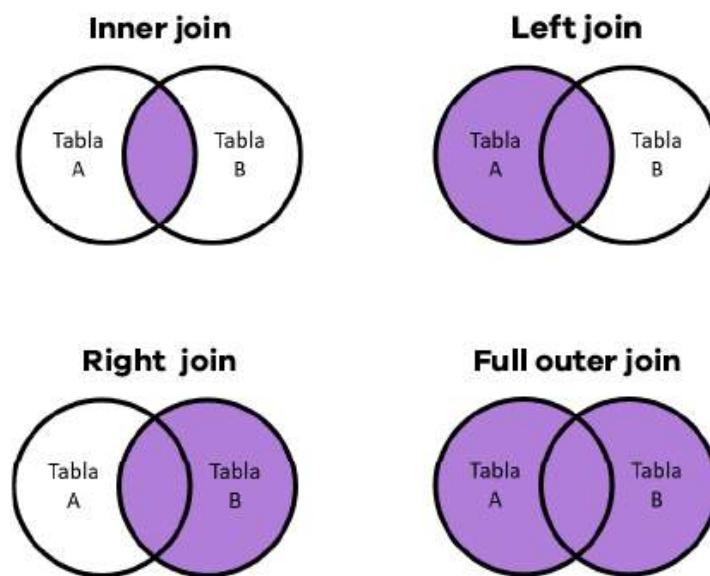
Full outer join

El *full outer join* devuelve **todas las filas que aparecen en alguna de las dos tablas**. Es decir, para que una muestra aparezca en la tabla de salida, basta con que exista en una de las dos tablas. En nuestro ejemplo, el resultado contendría las observaciones de enero, febrero, marzo y abril, tal y como muestra la siguiente imagen.

Identificación		Atributos	
Fecha	Versión	Clicks	Impresiones
01/01/2021	Antigua	347	NA
01/01/2021	Actualizada	423	NA
...
01/02/2021	Antigua	295	3541
01/02/2021	Actualizada	341	3623
...
31/03/2021	Antigua	321	4416
31/03/2021	Actualizada	578	4752
...
30/04/2021	Antigua	NA	3212
30/04/2021	Actualizada	NA	3091

Los clicks e impresiones adquieren valor NA en aquellas muestras para las que no existe su dato.

El siguiente gráfico representa estas ideas de una forma muy visual y clara:



Veamos ahora como todo esto se transforma en código R.

Los cuatro tipos de *joins* se encuentran disponibles en el paquete dplyr por medio de 4 funciones diferentes:

- 1 `inner_join()`
- 2 `left_join()`
- 3 `right_join()`
- 4 `full_join()`

Como las cuatro cuentan con la misma estructura de parámetros, podemos definirlas a la vez:

Funciones `inner_join()`, `left_join()`, `right_join()` y `full_join()`

Parámetro	Clase	Definición
x	dataframe	Dataframe a cruzar situada en la izquierda.
y	dataframe	Dataframe a cruzar situada en la derecha.
by	vector	Indica la variable o variables que hay que usar para realizar el cruce (si se deja a NULL, se utilizan las variables con el mismo nombre).

A continuación, llevamos a código el ejemplo de clicks e impresiones que hemos usado durante la explicación:

```
# df_clicks almacena los clicks diarios generados por cada versión del anuncio en los meses de enero, febrero y marzo de 2021
```

```
# df_impresiones almacena las impresiones diarias de cada versión del anuncio en los meses de febrero, marzo y abril de 2021.
```

```
# Realizamos un inner join
```

```
inner_join(df_clicks, df_impresiones, by = c("Fecha", "Version"))
```

```
# Realizamos un left join
```

```
left_join(df_clicks, df_impresiones, by = c("Fecha", "Version"))
```

```
# Realizamos un right join
```

```
right_join(df_clicks, df_impresiones, by = c("Fecha", "Version"))
```

```
# Realizamos un full outer join
```

```
full_join(df_clicks, df_impresiones, by = c("Fecha", "Version"))
```

Conclusiones

X Edix Educación

Como ya os he comentado más de una vez (y os cansaréis de oírlo), **la limpieza y el tratamiento de datos es una de las tareas que más tiempo consume en los proyectos de analítica**, de ahí su importancia dentro del programa de la asignatura.

Para que sirva como revisión final, entre el fastbook anterior y este hemos aprendido a:

- Realizar **transformaciones estructurales** (movernos entre los formatos wide y long gracias al paquete `tidyR`).
- Usar el operador **pipe** (`%>%`) de `dplyr`, así como los objetos tipo **tibble**.
- Seleccionar las columnas que nos interesen (función **select**).
- Filtrar muestras a partir de sus características (función **filter**).
- Ordenar en base a determinadas columnas (función **arrange**).
- Renombrar columnas (función **rename**).
- Modificar y crear nuevas variables (función **mutate**).
- Realizar agregaciones de datos (funciones **group_by** y **summarise**).
- Cruzar tablas por medio de joins (funciones **inner_join**, **left_join**, **right_join** y **full_join**).

Con este listado cubrimos las tareas y transformaciones más importantes, sin embargo, dplyr es un paquete muy grande y potente que sigue creciendo a día de hoy, por lo que existen otras funciones más específicas que podrían resultarnos útiles en ciertos escenarios.

Por eso, me gusta recomendar su [cheatsheet](#). En ella encontramos dos hojas que resumen todo el potencial de este paquete.

La pregunta que surge ahora es, ¿qué es lo que queda por abordar en los últimos dos fastbooks? La respuesta es la comunicación.

Como data scientist no sirve de nada realizar las transformaciones y modelizaciones más avanzadas y precisas si no somos capaces de obtener conclusiones y comunicarlas de una forma clara y eficiente. Los archivos RMarkdown son una gran herramienta de apoyo en aras de alcanzar ese objetivo, gracias a que combinan código R y texto formateado en un mismo documento.

En los siguientes dos fastbooks aprenderemos a construirlos y utilizarlos de forma inteligente.

¡Enhорabuena! Fastbook superado

edix

Creamos Digital Workers