Transformación de datos

Luciana Sastre

7/04/2021

Introducción

¿Qué pasa si no tengo los datos de la forma en que los necesito?

Bueno, la idea de este capítulo es aprender a como transformar los datos usando el paquete **dplyr**.

Para ello vamos a utilizar un conjunto de datos de salidas de vuelos en Nueva York en el año 2013, el cual está contenido en el paquete **datos**.

Vuelos

vuelos es un conjunto de datos que contiene los 336,776 vuelos que salieron desde Nueva York en el 2013.

```
vuelos[1:6,1:5]
```

```
## # A tibble: 6 x 5
##    anio   mes   dia horario_salida salida_programada
##   <int> <int> <int>          <int>             <int>
## 1  2013     1     1            517               515
## 2  2013     1     1            533               529
## 3  2013     1     1            542               540
## 4  2013     1     1            544               545
## 5  2013     1     1            554               600
## 6  2013     1     1            554               558
```

Observación

1) Notar que *vuelos* se imprime de forma diferente a lo que podríamos estar acostumbrados. En este caso se muestran solo las primeras filas y todas las columnas que caben en nuestra pantalla. Esto se debe a que es un **tibble**. Los tibble son data frames que están ajustados para que funcionen mejor en el tidyverse. Más adelante hablaremos más en detalle sobre ellos.
2) Notar que hay una fila compuesta únicamente de abreviaturas. Estas describen el tipo de cada variable. Las que aparecen en nuestro conjunto de datos son las siguientes:

- *int*: significa enteros
- *dbl* : significa dobles, o números reales
- *chr*: significa vectores de caracteres o cadenas
- *dttm*: significa fechas y horas (una fecha + una hora)

Funciones de dplyr

En este capítulo vamos a ver las 5 funciones básicas de dplyr. En esta presentación solo vamos a profundizar en las primeras 3 funciones.

- Filtrar (elegir) observaciones según sus valores (*filter*())
- Reordenar las filas (*arrange*())
- Seleccionar las variables por sus nombres (*select*())
- Crear nuevas variables transformando las variables existentes (*mutate*())
- Contraer muchos valores en un solo resumen (*summarise*())

Observación

Estas funciones las podemos combinar con group_by() que cambia el alcance de cada función. Es decir, en vez de que la función trabaje sobre todo el conjunto de datos, lo haga de grupo en grupo.

Filtrar filas con filter

filter() nos permite seleccionar un subconjunto de observaciones según sus valores.

Su primer argumento es el nombre del data frame y los siguientes son las expresiones que lo filtran.

Por ejemplo, podemos seleccionar todos los vuelos del 2 de enero:

```
filter(vuelos, mes==1, dia==2)[1:6, 1:4]
```

```
## # A tibble: 6 x 4
##    anio   mes   dia horario_salida
##   <int> <int> <int>          <int>
## 1  2013     1     2             42
## 2  2013     1     2            126
## 3  2013     1     2            458
## 4  2013     1     2            512
## 5  2013     1     2            535
## 6  2013     1     2            536
```

Observaciones

1) Las funciones de dplyr nunca modifican su input, es decir, que si queremos guardar el cambio realizado tenemos que asignarlo a una variable.

```
ene2 <- filter(vuelos, mes==1, dia==2)
```

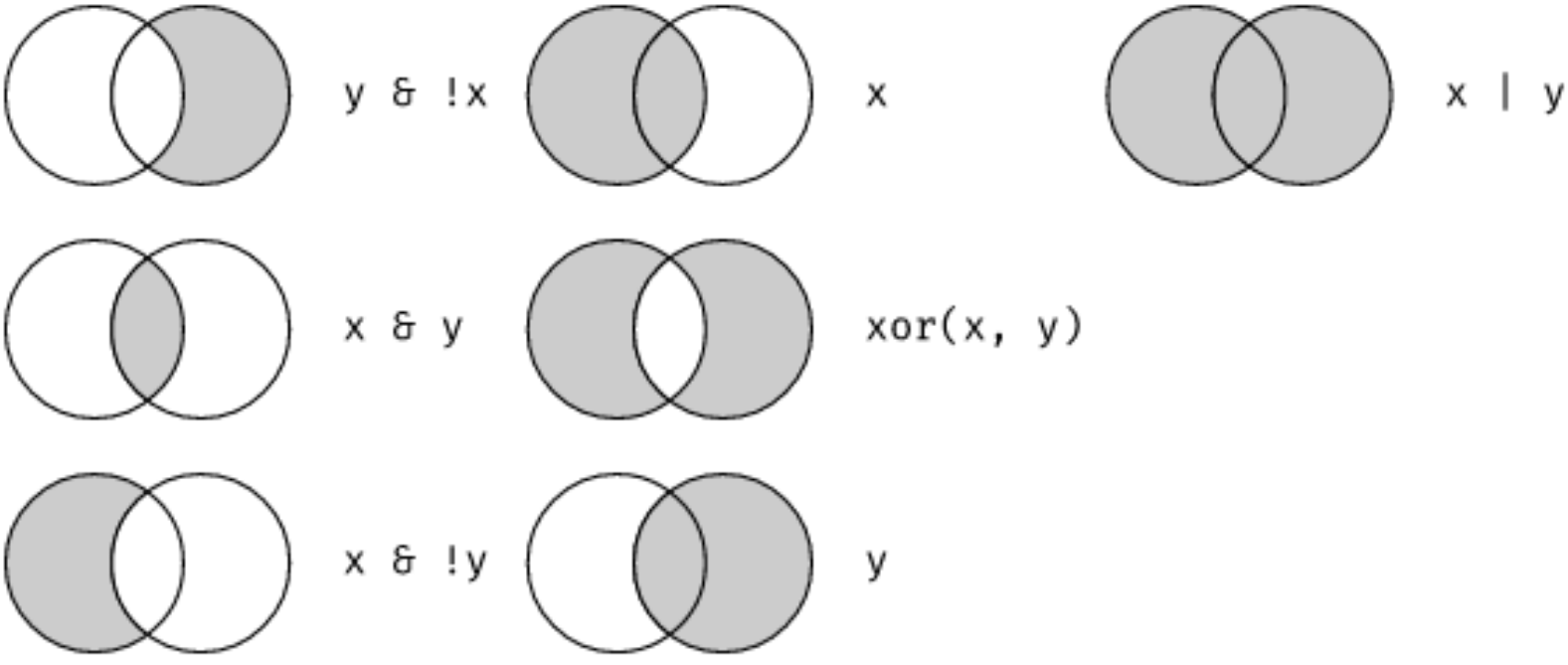
2) Esto genera que se guarden los cambios, si queremos que se guarden y se impriman a la vez hay que escribir la línea anterior en paréntesis.

```
(ene2 <- filter(vuelos, mes==1, dia==2))[1:4, 1:3]
```

```
## # A tibble: 4 x 3
##    anio   mes   dia
##   <int> <int> <int>
## 1  2013     1     2
## 2  2013     1     2
## 3  2013     1     2
## 4  2013     1     2
```

Operadores lógicos

- si queremos incluir varios argumentos en *filter*(), debemos hacerlo con "y". No obstante, las expresiones deben ser verdaderas para que las filas que queremos filtrar se incluyan en el output.
- si queremos incluir otro tipo de combinaciones tenemos que recurrir a los operadores Booleanos: & es "y", | es "o" y ! es "no".



Veamos algunos ejemplos

- Para ello vayamos a R y veámoslo ahí

Cuestiones utiles a tener en cuenta

- Podemos usar %in%, en donde si tenemos x %in% y (se lee: “x en y”) estamos seleccionado las filas en donde x tiene alguno de los valores de y
- Tener en cuenta las leyes De Morgan: !(x & z) = !x | !z, y !(x | z) = !x & !z

Valores falantes

Los valores faltantes o NA representan un valor que desconocemos. Tiene la característica de que cualquier operación que involucra un valor desconocido también será desconocida.

filter() solo incluye las filas en donde la condición es TRUE, por ende, los valores FALSE o NA son excluidos. Si nosotros los queremos conservar los valores desconocidos debemos solicitarlos explícitamente.

```
df <- tibble(x = c(1,NA,3))
filter(df, is.na(x) | x > 1)
```

```
## # A tibble: 2 x 1
##       x
##   <dbl>
## 1    NA
## 2     3
```

Ejemplo

Por ejemplo, si queremos filtrar los valores faltantes en horario_salida lo hacemos de la siguiente manera:

```
filter(vuelos, is.na(horario_salida))[1:6, 4:6]
```

```
## # A tibble: 6 x 3
##   horario_salida salida_programada atraso_salida
##            <int>             <int>         <dbl>
## 1             NA              1630            NA
## 2             NA              1935            NA
## 3             NA              1500            NA
## 4             NA               600            NA
## 5             NA              1540            NA
## 6             NA              1620            NA
```

Función útil

Una función útil para usar junto con filter es *between()*.

La misma nos permite seleccionar valores que se encuentren dentro de un intervalo (en otras palabras, entre dos números).

Sus argumentos son: between(x, left, right) donde:

- x: un vector de valores
- left, right: los extremos del intervalo (siempre deben ser escalares)

```
filter(vuelos, between(atraso_llegada, 0,10))[1:4, 7:9]
```

```
## # A tibble: 4 x 3
##   horario_llegada llegada_programada atraso_llegada
##             <int>              <int>          <dbl>
## 1             753                745              8
## 2             924                917              7
## 3             925                921              4
## 4            1017               1014              3
```

Reordenar las filas con arrange()

arrange() es muy parecida a filter() en su funcionamiento, en vez de filtrar las filas, les cambia el orden.

arrange() toma un data frame y un conjunto de nombres de columnas para ordenar según ellas. Cada columna adicional que se agrega servirá para romper empates en los valores de las columnas anteriores.

```
arrange(vuelos, horario_salida, salida_programada)[1000:1008, 3:7]
```

```
## # A tibble: 9 x 5
##     dia horario_salida salida_programada atraso_salida horario_llegada
##   <int>          <int>              <int>          <dbl>           <int>
## 1    25            123               2000            323             229
## 2    25            123               2029            294             215
## 3    22            123               2159            204             240
## 4     2            123               2359             84             511
## 5     2            124               2059            265             311
## 6     2            124               2225            179             226
## 7    28            125               1705            500             316
## 8     8            125               2138            227             356
## 9    30            125               2155            210             420
```

Observación

Por defecto, nos ordena la columnas de menor a mayor. Si queremos que el orden sea descendiente debemos explicitarlo con *desc*() ¿Qué sucede cuando queremos agregar más columnas?

```
arrange(vuelos, desc(horario_salida))[1:4, 3:5]
```

```
## # A tibble: 4 x 3
##     dia horario_salida salida_programada
##   <int>          <int>              <int>
## 1    30           2400               2359
## 2    27           2400               2359
## 3     5           2400               2359
## 4     9           2400               2359
```

Observación

Por defecto los valores NA (valores faltantes) siempre se ordenan al final. ¿Qué pasa si los queremos al principio?

```
df <- tibble(x = c(NA, 2, 5))
arrange(df, x)
```

```
## # A tibble: 3 x 1
##       x
##   <dbl>
## 1     2
## 2     5
## 3    NA
```

Seleccionar columnas con select()

La función *select*() nos permite seleccionar un subconjunto de variables utilizando operaciones basadas en los nombres de las mismas.

¿Cómo usar select?

- podemos seleccionar columnas por sus nombres

```
select(vuelos, anio, mes, dia)[1:6,]
```

```
## # A tibble: 6 x 3
##    anio   mes   dia
##   <int> <int> <int>
## 1  2013     1     1
## 2  2013     1     1
## 3  2013     1     1
## 4  2013     1     1
## 5  2013     1     1
## 6  2013     1     1
```

-podemos seleccionar todas las columnas entre dos especificas (incluyente)

```
select(vuelos, anio:dia)[1:6, ]
```

```
## # A tibble: 6 x 3
##    anio   mes   dia
##   <int> <int> <int>
## 1  2013     1     1
## 2  2013     1     1
## 3  2013     1     1
## 4  2013     1     1
## 5  2013     1     1
## 6  2013     1     1
```

- podemos seleccionar todas la alumnas excepto aquellas entre dos especificas (excluyente)

```
select(vuelos, -(anio:dia))[1:6, 1:3]
```

```
## # A tibble: 6 x 3
##   horario_salida salida_programada atraso_salida
##            <int>             <int>         <dbl>
## 1            517               515             2
## 2            533               529             4
## 3            542               540             2
## 4            544               545            -1
## 5            554               600            -6
## 6            554               558            -4
```

Funciones auxiliares

Esta es una lista de funciones auxiliares que podemos usar dentro de select():

- starts_with("abc") : selecciona las que tienen nombre que comienza con "abc"
- ends_with("abc"): selecciona las que tienen nombre que termina con "abc"
- contains("abc"): selecciona las que tienen nombre que contienen "abc"
- matches("(.) \ 1") : Selecciona variables que coincidan con una expresión regular (es una secuencia de caracteres que forma un patrón de búsqueda, principalmente utilizada para la búsqueda de patrones de cadenas de caracteres u operaciones de sustituciones.)
- num_range("x", 1:3) : selecciona las variables que coinciden con x1, x2, x3.

Veamos en R un par de ejemplos.

Comentarios sobre select() y las funciones auxiliares

- Cuando seleccionamos una columna por su nombre, es importante que coincida con el nombre de la columna en el data frame, es decir, las minúsculas y mayúsculas se tienen que respetar.
- Por defecto, cuando usamos contains, starts_with, ends_with no es necesario respetar el uso de las mayúsculas y minúsculas, es decir, las toma por igual. Si queremos que haga diferencia debemos incluir en sus argumentos ignore.case = FALSE.

Veamos un par de ejemplos en R para entender mejor

Funciones auxiliares

Podemos usar *select*() para cambiar el nombre a las variables pero no se recomienda pues descarta todas las variables que no se mencionan explícitamente. En vez, podemos utilizar *rename*(), que si mantiene todas las variables que no se mencionan explícitamente.

Otra función auxiliar que podemos usar junto a select() es everything(). Esto es particularmente útil cuando queremos mover variables al principio del data frame

```
select(vuelos, horario_salida, horario_llegada, everything())[1:6, 1:6]
```

```
## # A tibble: 6 x 6
##   horario_salida horario_llegada  anio   mes   dia salida_programada
##            <int>           <int> <int> <int> <int>             <int>
## 1            517             830  2013     1     1               515
## 2            533             850  2013     1     1               529
## 3            542             923  2013     1     1               540
## 4            544            1004  2013     1     1               545
## 5            554             812  2013     1     1               600
## 6            554             740  2013     1     1               558
```

También podemos usar la función auxiliar any_of(). Suele ser útil cuando queremos asegurarnos de que una variable haya sido removida, por ejemplo.

```
variables <- c("mes", "anio", "dia", "atraso_llegada", "accidentes")
select(vuelos, any_of(variables))[1:4,]
```

```
## # A tibble: 4 x 4
##     mes  anio   dia atraso_llegada
##   <int> <int> <int>           <dbl>
## 1     1  2013     1              11
## 2     1  2013     1              20
## 3     1  2013     1              33
## 4     1  2013     1             -18
```