4.Manipulacion_Datos

Paula Vargas Pellicer

15/02/2022

Subconjuntos, extracción y modificación de datos

Las bases de datos son objetos hechos de filas y columnas que contienen observaciones de diferentes variables: a menudo importarás tus datos de esa manera. A veces hay algunos errores después de importar, o tal vez necesitas cambiar el nombre de una variable o mantener solo un subconjunto de los datos que cumple con algunas condiciones, (como ya lo hemos hecho en la primera parte).

Profundicemos con en el conjunto de datos `EmpetrumElongation.csv` que puedes descargar del repositorio.

Crea un nuevo "script" en blanco y agrega información en la parte superior, por ejemplo, el título de la clase de hoy, tu nombre y la fecha (recuerda usar # para comentar y anotar tu "script").

Este conjunto de datos representa los incrementos anuales en el crecimiento del tallo en arbustos de Empetrum nigrum en un sistema de dunas de arena. La columna `Zone` corresponde a distintas zonas que van desde la más cercana (2) a la más lejana (7) del mar.

Ya hemos visto que podemos acceder a las variables en R usando el signo de peso $. Esta ya es una forma de crear subconjuntos, ya que esencialmente reduce una base de datos (2 dimensiones) a un vector (1 dimensión). También puedes acceder a partes de una base de datos usando corchetes [,]. El primer número que pongas obtendrá el número de fila, y el segundo el de columna. Puedes dejar uno en blanco para mantener todas las filas o todas las columnas.

```r
# Establece tu directorio de trabajo
setwd("ruta-de-archivo") #OJO! pon el tuyo

# Carga la base de datos
elongation <- read.csv("EmpetrumElongation.csv", header = TRUE)

# Checa que todo este bien
head(elongation)  #

str(elongation)   #

# Extraigamos informacion
elongation$Indiv   #

length(unique(elongation$Indiv))   #
```

```r
# Así conocemos el valor de la segunda fila en la quinta columna
elongation[2,5]

# Así conocemos la información de la sexta fila
elongation[6, ]

#
elongation[6, ]$Indiv
```

Crear subconjuntos con corchetes usando números de fila y columna puede ser bastante tedioso si tienes un conjunto de datos grande y no sabes dónde se encuentran las observaciones que estás buscando. En general, no se recomienda usarlos de todos modos para hacer subconuntos, porque si codificas un número en tu secuencia de comandos y agregas algunas filas más adelante, es posible que ya no selecciones las mismas observaciones. Es por eso, que podemos usar operaciones lógicas para acceder a partes específicas de los datos que coinciden con nuestra especificación.

```r
# Podemos acceder a los valores del individuo 603
elongation[elongation$Indiv == 603, ]
```

¡Hay mucho que desempacar en este pedazo de código! Estamos diciendo: "Toma esta base de datos (elongation), haz un subconjunto ([,]) para mantener las filas (escribiendo la expresión a la izquierda de la coma) para las cuales el valor en la columna Indiv es exactamente(==)603".

Nota: La expresión lógica funciona aquí porque la columna Indiv contiene valores numéricos: para acceder a datos que son de tipo carácter o factor, se usarían comillas: elongation$Indiv == "seiscientos tres".

Operadores para operaciones lógicas

A continuación, pongo algunos de los operadores más utilizados para manipular datos. Cuando los uses para crear una condición de subconjunto, R evaluará la expresión y devolverá solo las observaciones para las que se cumple la condición.

== es exactamente igual

<, <= es menor que, es menor que o igual a

>, >= es mayor que, es mayor que o igual a

!= no igual a

%in% pertenece a uno de los siguientes (normalmente seguido de un vector de valores posibles)

& operador Y, permite encadenar dos condiciones que deben cumplirse

| operador O, para encadenar dos condiciones cuando al menos una debe cumplirse

! operador NO, para especificar cosas que deben omitirse

```r
# Hacer subconjuntos con una condición
elongation[elongation$Zone < 4, ]    #

elongation[elongation$Zone <= 4, ]   #

# O, de manera equivalente
elongation[!elongation$Zone >= 5, ]   #

# Hacer subconjuntos con dos condiciones
elongation[elongation$Zone == 2 | elongation$Zone == 7, ]    #

elongation[elongation$Zone == 2 & elongation$Indiv %in% c(300:400), ]   #
```

Como puedes ver, cuanto más exigente eres con las condiciones, más complejo se vuelve el código. Luego aprenderemos algunas funciones que realizan estas acciones de una manera más limpia y minimalista, pero a veces no podrás escapar usando la base R (especialmente cuando se trata de objetos que no son bases de datos), por lo que es bueno comprender estos notaciones

Otros generadores de secuencias de vectores útiles son:

- seq() para crear una secuencia, aumentando en cualquier cantidad especificada. P.ej. intenta secuencia (300, 400, 10)
- rep() para crear repeticiones de elementos. P.ej. rep(c(1,2), 3) dará 1 2 1 2 1 2.
- ¡Puedes mezclar y combinar! ¿Qué daría rep(seq(0, 30, 10), 4)?

Y finalmente, supongamos que necesitas modificar algunos valores o niveles de factores, o deseas crear una nueva columna. Ahora que sabes cómo acceder a partes de una base de datos, puedes hacer todo eso. Solo necesitas una herramienta adicional: la flecha de asignación <- para sobrescribir datos.

```r
## Cambiar nombres de variables

# Vamos a hacer primero una copia de la BD (base de datos) original
elong2 <- elongation

# Cambiemos el nombre de una columna

names(elong2)                  # Te da el nombre de las columnas

names(elong2)[1] <- "zone"     # Asigna un nuevo valor, en este caso, el nombr
e

names(elong2)[2] <- "ID"       #

# Supongamos que hay un error en la captura de los datos, y el valor 5.1 del
individuo 373 del año 2008m debería de ser 5.7

## - opción 1
elong2[1,4] <- 5.7
```

```
## - opción 2
elong2[elong2$ID == 373, ]$X2008 <- 5.7
```

👉 ¿Puedes identificar los pros y los contras de las opciones 1 y 2 anteriores?

Usando las mismas técnicas, puedes especificar clases de variables, lo que será muy útil cuando lleguemos a diseñar modelos estadísticos y necesitemos agrupar variables como factores.

```
## Crear un factor

#
str(elong2)

# la columna zone, es en realidad un factor

elong2$zone <- as.factor(elong2$zone)        #

str(elong2)                                  #

## Cambiar su nombre

levels(elong2$zone)  #

levels(elong2$zone) <- c("A", "B", "C", "D", "E", "F")   #
```

¿Qué son los datos ordenados y cómo los logramos?

La forma en que registres la información en el campo o en el laboratorio es probablemente muy diferente a la forma en que deseas ingresar tus datos en R. En el campo, quieres tablas que idealmente puedas elaborar con anticipación y completar sobre la marcha, ir añadiendo notas y todo tipo de información además de los datos que quieras analizar. Por ejemplo, si monitoreas la altura de las plántulas durante un experimento factorial usando tratamientos de calentamiento y fertilización, podrías registrar tus datos de esta manera:

		Warm		Fertilised		Warm + Fertilised		Control	
	seedling	Species 1	Species 2	Species 1	Species 2	Species 1	Species 2	Species 1	Species 2
Week 1	1								
	2								
	3								
	4								
	5								
	6								
	7								
	8								
	9								
	10								
Week 2	1								
	2								
	3								
	4								
	5								
	6								
	7								
	8								
	9								
	10								

Week 3 etc

Supongamos que deseas realizar una prueba para determinar si el calentamiento y/o la fertilización afectaron el crecimiento de las plántulas. Por el momento, con 8 medidas por fila (combinación de todos los tratamientos y especies para una réplica o bloque), no se puede ejecutar un análisis. Por el contrario, los conjuntos de datos ordenados se organizan de modo que cada fila represente una observación y cada columna represente una variable. En nuestro caso, esto se vería así:

Week	Species	Block	Treatment	Length (cm)
1	1	1	Warmed	
1	1	1	Fertilised	
1	1	1	W + F	
1	1	1	Control	
1	2	1	Warmed	
1	2	1	Fertilised	
1	2	1	W + F	
1	2	1	Control	
1	1	2	Warmed	
1	1	2	Fertilised	
1	1	2	W + F	
1	1	2	Control	
1	2	2	Warmed	
1	2	2	Fertilised	
1	2	2	W + F	
1	2	2	Control	

(etc) (through to 10)

Esto hace que la base de datos sea mucho más larga en filas, por lo que esta forma a menudo se denomina formato largo. Ahora, si quisieras comparar entre grupos, tratamientos, especies, etc., R podría dividir la base de datos correctamente, ya que cada factor de agrupación tiene su propia columna.

Basado en esto, ¿notas algo que no está del todo ordenado con nuestra base de datos anterior (`elongation`)? Tenemos la observación de la misma variable, es decir, la longitud del tallo, distribuida en múltiples columnas que representan diferentes años.

La función de `gather()` del paquete tidyr nos permitirá convertir esta tabla de formato ancho en una base de datos ordenada. Queremos crear una sola columna *year* (año) que tenga los años (2007-2012) repetidos para cada individuo. A partir de esto, deberías poder calcular que la base de datos será seis veces más larga que la original. También queremos una columna `lenght` donde irán todos los datos de crecimiento asociados a cada año e individuo.

Nota: Esta función es un poco inusual ya que está creando sus propios nombres de columna en el segundo (key=clave) y tercer (value=valor) argumentos, en lugar de pasarles objetos o valores predefinidos como la mayoría de las funciones de R. Aquí, el año es nuestra clave y longitud es nuestro valor.

```r
install.packages("tidyr")  #

library(tidyr)             #


# orden: base de datos, clave, valor
elongation_long <- gather(elongation, Year, Length,
c(X2007, X2008, X2009, X2010, X2011, X2012))  # Qué columnas debe reunir

# Usemos ahora el reverso: `spread()`, para ir, de formato largo a ancho
elongation_wide <- spread(elongation_long, Year, Length)
```

Observa cómo usamos los nombres de las columnas para indicar a la función de `gather()` qué columnas remodelar. Esto es útil si solo tiene unas pocas, y si las columnas cambian de orden eventualmente. Sin embargo, si tienes un conjunto de datos con columnas para 100 genes, por ejemplo, es mejor que especifiques los números de columna:

```r
elongation_long2 <- gather(elongation, Year, Length, c(3:8))
```

Estas funciones tienen limitaciones y no funcionarán en todas las estructuras de datos. Esta es la razón por la cual pensar un poco en la estructura de tus datos antes de digitalizarlos te puede ahorrar mucha frustración más adelante

Una vez que tengas los datos en el formato correcto, es mucho más fácil analizarlos y visualizar los resultados. Por ejemplo, si queremos saber si hay variación interanual en el crecimiento de *Empetrum hermaphroditum*, podemos hacer rápidamente una gráfica de caja:

```
boxplot(Length ~ Year, data = elongation_long,
        xlab = "Year", ylab = "Elongation (cm)",
    main = "Crecimiento anual de Empetrum hermaphroditum")
```

Al mirar la grafica, hay una superposición bastante grande entre el crecimiento anual de cada año, en realidad no hay mucho que ver aquí. (Aprenderemos a hacer gráficas más bonitas e interesantes más adelante).

Explorar las funciones más comunes y útiles de dplyr

El paquete dplyr es un paquete fantástico de funciones intuitivas para la manipulación de datos, que lleva el nombre de la acción que realizan. Una gran ventaja de estas funciones es que toman la base de datos como primer argumento, de modo que puedes referirte a las columnas sin tener que referirse explícitamente al objeto completo (¡así que podemos eliminar los signos $!). Conozcamos las funciones más comunes y útiles trabajando en el objeto de formato largo que acabamos de crear, elongation_long.

Primero, instala y carga el paquete.

```
install.packages("dplyr")  #

library(dplyr)              #
```

1. rename() las variables

Esto te permite cambiar los nombres de una columna o varias columnas.

```
elongation_long <- rename(elongation_long, zone = Zone, indiv = Indiv, year =
Year, length = Length)     #

# Asi seria en la base de R
names(elongation_long) <- c("zone", "indiv", "year", "length")
```

2. filter() filas y select() columnas

Estas son algunas de las funciones más rutinarias que permiten reducir la base de datos a las filas y columnas que necesitas. La función `filter()` funciona muy bien para crear subconjuntos de filas con operaciones lógicas. La función `select()` permite especificar qué columnas conservar.

Nota: la función `select()` a menudo se usa en otras funciones del mismo nombre en otros paquetes, y por esa razón se recomienda usar `dplyr::select()` cada vez que se use

```r
# Filtrar observaciones

# mantener observaciones solo de las zonas 2 y 3, y del 2009 al 2011

elong_subset <- filter(elongation_long, zone %in% c(2, 3), year %in% c("X2009
", "X2010", "X2011")) #

# Así seria en la base de R
elongation_long[elongation_long$zone %in% c(2,3) & elongation_long$year %in%
c("X2009", "X2010", "X2011"), ]
```

Ten en cuenta que aquí usamos %in% como operador lógico porque buscamos hacer coincidir una lista de valores exactos (caracteres). Si deseas mantener las observaciones dentro de un rango de valores numéricos, necesitas dos declaraciones lógicas en la función `filter()`, por ejemplo, `length > 4 & length <= 6.5` o puedes usar la conveniente función `between()`, por ejemplo `between(length, 4, 6.5)`.

Aquí empezarás a ver cómo `dplyr` evita la repetición llamando directamente a los nombres de las columnas sin necesidad de llamar al objeto cada vez

USO DE LAS COMILLAS “”

¿Citar o no citar? Es posible que hayas notado cómo a veces llamamos valores entre comillas "", y otras veces no. Esto depende de: Ya sea que el valor que estás llamando sea un carácter o un valor numérico: en el ejemplo de arriba, `zone` es de clase entera (un número), por lo que no necesitamos comillas alrededor de los valores que toma, pero el año es un carácter (letras), así que las necesita.

- Por otro lado, si estás llamando a un objeto existente o refiriéndote a un valor que R aún no conoce, entonces puede pasar esto:
  nuevo.objeto <- elongation_long
  nuevo.objeto <- "elongation_long"
  El primero crea un duplicado de nuestro objeto, porque R reconoce el nombre como un objeto en nuestro entorno. En el segundo caso, está creando un objeto que consta de un valor de carácter.

Hagamos un subconjunto de columnas

```r
# Selecciona las columnas

# Nos deshacemos de la columna zone

elong_no.zone <- dplyr::select(elongation_long, indiv, year, length)   # ó

elong_no.zone <- dplyr::select(elongation_long, -zone) # el símbolo - elimina
columna

# Así sería en R base:
elongation_long[ , -1]  # quita primera columna

#
elong_no.zone <- dplyr::select(elongation_long, Year = year, Shrub.ID = indiv
, Growth = length)
```

3. mutate() tu conjunto de datos creando nuevas columnas

Un comando útil para crear una nueva columna. Si deseas realizar una operación en varias columnas, o quizás reclasificar un factor. La función `mutate()` hace exactamente eso y también permite definir el nombre de la columna. Aquí vamos a usar nuestra primera base de datos `elongation` de formato ancho y crear una columna que represente el crecimiento total para el período 2007-2012:

```
# Crear una nueva columna

elong_total <- mutate(elongation, total.growth = X2007 + X2008 + X2009 + X201
0 + X2011 + X2012)
```

Ahora, veamos cómo podríamos lograr lo mismo en nuestros datos de formato largo elongation_long mediante el uso de dos funciones que combinan muy bien juntas: `group_by()` y `summarise()`.

4. group_by() ciertos para hacer operaciones

Lo más importante que hay que comprender acerca de esta función es que no verás ningún cambio visible en tu base de datos. Crea una estructura de agrupación interna, lo que significa que cada función subsiguiente que ejecute utilizará estos grupos, y no todo el conjunto de datos, como entrada. Es muy útil cuando deseas calcular estadísticas de resumen para diferentes sitios, tratamientos, especies, etc.

```
# Agrupar datos

elong_grouped <- group_by(elongation_long, indiv)   # Agrupar por individuo
```

Compara elong_grouped y elongation_long: deberían verse exactamente iguales. Pero ahora, usemos summarise() para calcular el crecimiento total de cada individuo a lo largo de los años.

5. summarise() datos con estadística

Esta función siempre agregará la base de datos (BD) original, es decir, la BD de salida será más corta que la entrada. Aquí, vamos a contrastar la suma de los incrementos de crecimiento durante el período de estudio en el conjunto de datos original frente a nuestro nuevo conjunto de datos agrupados.

```
summary1 <- summarise(elongation_long, total.growth = sum(length))

summary2 <- summarise(elong_grouped, total.growth = sum(length))
```

El primer resumen corresponde a la suma de todos los incrementos de crecimiento en el conjunto de datos (todos los individuos y años). El segundo nos da un desglose del crecimiento total por individuo, nuestra variable de agrupación.

```r
summary3 <- summarise(elong_grouped, total.growth = sum(length),
                                     mean.growth = mean(length),
                                     sd.growth = sd(length))
```

6. ..._join() BD basadas en sus atributos

A veces, tienes varios archivos de datos relacionados con un mismo proyecto: uno para las mediciones tomadas en varios sitios, otros con datos climáticos en estos sitios y quizás algunos metadatos sobre un experimento. Dependiendo de tus necesidades analíticas, puede ser muy útil tener toda la información en una tabla. Aquí es donde la fusión o unión de conjuntos de datos resulta útil.

Imaginemos que los datos de crecimiento con los que hemos estado trabajando en realidad provienen de un experimento en el que algunas plantas se calentaron con invernaderos portátiles (W), otras se fertilizaron (F), algunas recibieron ambos tratamientos (WF) y algunas fueron plantas de control (C) . Importaremos estos datos del archivo `EmpetrumTreatments.csv`, que contiene los detalles de qué individuos recibieron qué tratamientos, y los uniremos con nuestro conjunto de datos principal `elongation_long`.

Importante: Podemos hacer esto porque ambos conjuntos de datos tienen una columna que representa el ID de cada planta: esto es lo que fusionaremos.

Hay muchos tipos de uniones que puede realizar, se diferencian en la forma en que manejan los datos que no comparten ambas tablas, así que siempre hay que preguntarse qué observaciones necesitas conservar y cuáles quieres descartar, (en caso de duda, `full_join()` conservará todo). En el siguiente ejemplo, queremos mantener toda la información en `elong_long` y repetir el código de tratamiento para las cinco ocurrencias de cada individuo, por lo que usaremos `left_join()`.

```r
# Cargar los datos de tratamiento asociados a cada individuo

treatments <- read.csv("EmpetrumTreatments.csv", header = TRUE, sep = ";")

head(treatments)

# Unir dos bases de datos por ID. Las dos columnas están escritas distintas,
por lo que debemos indicar qué columnas coinciden: zone y ID.

experiment <- left_join(elongation_long, treatments, by = c("indiv" = "Indiv"
, "zone" = "Zone"))

# El nuevo objeto tiene la misma longitud que la primera base de datos, que
es lo queremos. y se ha añadido la columna de tratamientos para cada planta
```

La función base R equivalente es merge() y en realidad también funciona muy bien:

```r
experiment2 <- merge(elongation_long, treatments, by.x = c("zone", "indiv"),
by.y = c("Zone", "Indiv"))

# Mismo resultado
```

Ahora podemos verificar si afectan el crecimiento dibujando otra gráfica de cajas.

```
boxplot(length ~ Treatment, data = experiment)
```

¿Son estas diferencias estadísticamente significativas? ¡Descubriremos cómo probar esto en la clase de modelado!

Ejercicio

En el repositorio, encontrarás el archivo `dragons.csv`, que proporciona la longitud (en cm) de las columnas de fuego que respiran los dragones de diferentes especies cuando se alimentan con diferentes especias.

Tienes que es ordenar los datos (formato largo) y crear una gráfica de caja para cada especie que muestre el efecto de las especias en el tamaño de la columna de fuego, de modo que pueda responder las preguntas: ¿Qué especia desencadena la reacción más intensa? ¿Y lo mínimo?

Sin embargo, vas a descubrir que tu asistente de campo fue un poco descuidado durante la recopilación de datos y cometió muchos errores que debes corregir:

** El cuarto tratamiento no fue pimentón, fue cúrcuma.

** Hubo un error de calibración con el dispositivo de medición para la prueba de salsa Tabasco, pero solo para la especie "*Hungarian Horntail*". Todas las medidas son 30 cm más altas de lo que deberían ser.

** Las longitudes se dan en centímetros, pero tiene más sentido reportarlas en metros.