BIO 4022. Manipulación de datos e investigación reproducible en R

Derek Corcoran

2018-08-27

Contents

Requerimientos

Para comenzar el trabajo se necesita la última versión de R y RStudio (**?**).También se requiere de los paquetes *pacman*, *rmarkdown*, *tidyverse* y *tinytex*. Si no se ha usado R o RStudio anteriormente, el siguiente video muestra cómo instalar ambos programas y los paquetes necesarios para este curso en el siguiente link.

El código para la instalación de esos paquetes es el siguiente:

```
install.packages("pacman", "rmarkdown", "tidyverse", "tinytex")
```

En caso de necesitar ayuda para la instalación, contactarse con el instructor del curso.

0.1 Antes de comenzar

Si nunca se ha trabajado con `R` antes de este curso, una buena herramienta es provista por el paquete Swirl (**?**). Para comenzar la práctica, realizar los primeros 7 modulos del programa *R Programming: The basics of programming in R* que incluye:

- Basic Building Blocks
- Workspace and Files
- Sequences of Numbers
- Vectors
- Missing Values
- Subsetting Vectors
- Matrices and Data Frames

El siguiente link muestra un video explicativo de cómo usar el paquete swirl Video

0.2 Descripción del curso

Este curso está enfocado en entregar principios básicos de investigación reproducible en R, con énfasis en la recopilación y/o lectura de datos de forma reproducible y automatizada. Para esto se trabajará con bases de datos complejas, las cuales deberán ser transformadas y organizadas para optimizar su análisis. Se generarán documentos reproducibles integrando en un documento: código, bibliografía, exploración y análisis de datos. Se culminará el curso con la generación de un manuscrito, una presentación y/o un documento interactivo reproducible.

0.3 Objetivos del curso

1. Conocer y entender el concepto de investigación reproducible como una forma y filosofía de trabajo que permite que las investigaciones sean más ordenadas y replicables, desde la toma de datos hasta la escritura de resultados.

2. Conocer y aplicar el concepto de pipeline, el cual permite generar una modularidad desde la toma de datos hasta la escritura de resultados, donde la corrección independiente de un paso tiene un efecto cascada sobre el resultado final.

3. Aprender buenas prácticas de recolección y estandarización de bases de datos, con la finalidad de optimizar el análisis de datos y la revisión de éstas por pares.

4. Realizar análisis críticos de la naturaleza de los datos al realizar análisis exploratorios, que permitirán determinar la mejor forma de comprobar hipótesis asociadas a estas bases de datos.

0.4 Contenidos

- Capítulo **??** *Tidy Data*: En este capítulo se aprenderá a cómo optimizar una de base de datos, sobre la limpieza y transformación de bases de datos, qué es una base de datos *tidy* y cómo manipular estas bases de datos con el paquete *dplyr* (**?**).

- Capítulo **??** *Investigación reproducible*: En este capítulo se trabajará en la confección de un documento que combine códigos de `R` y texto para generar documentos reproducibles utilizando el paquete *rmarkdown* (**?**). Además, se verá cómo al usar RStudio se pueden guardar los proyectos en un repositorio de github.

- Capítulo **??** *El tidyverso* y el concepto de pipeline:En este capítulo se aprenderá sobre la limpieza de datos complejos.

- Capítulo **??** *Visualización de datos* visualizar datos vs. visualizar modelos. Insertar gráficos con leyenda en un documento Rmd

- Capítulo **??** *Modelos en R* Aprender a generar modelos en R, desde ANOVA a GLM.

- Capítulo **??** *Loops*. Generación de funciones propias en R y loops

6. Escritura de manuscritos en R, transformación de documentos Rmd en un manuscrito

7. Presentaciones en R y generar documentos interactivos. Transformación de datos en una presentación o en una Shiny app. Realizar una presentación o aplicación en R.

0.5 Metodología

Todas las clases estarán divididas en dos partes: I. Clases expositivas de principios y herramientas, donde se presentarán los principios de investigación reproducible y tidy data, junto con las herramientas actuales más utilizadas, y II. Clases prácticas donde cada estudiante trabajará con datos propios para desarrollar un documento reproducible. Los estudiantes que no cuenten con datos propios podrán acceder a sets de datos para su trabajo o podrán simularlos, dependiendo del caso.

Además, se deberán generar informes y presentaciones siguiendo los principios de investigación reproducible, en base al trabajo con sus datos. Se realizará un informe final, en el cual se espera un trabajo que compile los conociminetos adquiridos durante el curso.

0.6 Evaluación

- Evaluación 1: Informe exploratorio de base de datos 25%
- Evaluación 2: Presentación 25%
- Evaluación 3: Informe final 50%

0.7 Libros de consulta

Los principios de este curso están explicados en los siguientes libros gratuitos.

- Gandrud, Christopher. Reproducible Research with R and R Studio. CRC Press, 2013. Available for free in the following link
- Stodden, Victoria, Friedrich Leisch, and Roger D. Peng, eds. Implementing reproducible research. CRC Press, 2014. Available for free in the following link

0.8 Bibliografía

Chapter 1

Tidy Data y manipulación de datos

1.1 Paquetes necesarios para este capítulo

Para este capitulo necesitas tener instalado el paquete *tidyverse*

En este capítulo se explicará qué es una base de datos *tidy* (**?**) y se aprenderá a usar funciones del paquete *dplyr* (**?**) para manipular datos.

Dado que este libro es un apoyo para el curso BIO4022, esta clase del curso puede también ser seguida en este link. El video de la clase se encuentra disponible en este link.

1.2 Tidy data

Una base de datos tidy es una base de datos en la cuál (modificado de (**?**)):

- Cada vararible que se medida debe estar en una columna.
- Cada observación distinta de esa variable debe estar en una fila diferente.

En general, la forma en que representaríamos una base de datos *tidy* en `R` es usando un *data frame*.

1.3 dplyr

El paquete *dplyr* es definido por sus autores como una gramática para la manipulación de datos. De este modo sus funciones son conocidas como verbos. Un resumen útil de muchas de estas funciones se encuentra en este link.

Este paquete tiene un gran número de verbos y sería difícil ver todos en una clase, en este capítulo nos enfocaremos en sus funciones más utilizadas, las cuales son:

- *group_by* (agrupa datos)
- *summarize* (resume datos agrupados)
- *mutate* (genera variables nuevas)
- *%>%* (pipeline)
- *filter* (encuentra filas con ciertas condiciones)
- *select* junto a *starts_with*, *ends_with* o *contains*

Table 1.1: una tabla con 10 filas de la base de datos iris.

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.8	4.0	1.2	0.2	setosa
4.7	3.2	1.6	0.2	setosa
5.1	3.8	1.9	0.4	setosa
5.2	2.7	3.9	1.4	versicolor
6.4	2.9	4.3	1.3	versicolor
5.5	2.5	4.0	1.3	versicolor
6.5	3.0	5.8	2.2	virginica
6.0	2.2	5.0	1.5	virginica
6.1	2.6	5.6	1.4	virginica
5.9	3.0	5.1	1.8	virginica

Table 1.2: Resumen del promedio y desviación estándar del largo de pétalo de las flores del generi Iris.

Mean.Petal.Length	SD.Petal.Length
3.758	1.765298

1.3.1 summarize

La función `summarize` toma los datos de un data frame y los resume. Para usar esta función, el primer argumento que tomaríamos sería un data frame, se continúa del nombre que queremos darle a una variable resumen, seguida del signo = y luego la fórmula a aplicar a una o mas columnas. COmo un ejemplo se utilizará la base de datos `iris` (**?**) que viene en `R` y de las cual podemos ver parte de sus datos en la tabla **??**

Si quisieramos resumir esa tabla y generar un par de variables que fueran la media y la desviación estándar del largo del pétalo, lo haríamos con el siguiente código:

```
library(tidyverse)
Summary.Petal <- summarize(iris, Mean.Petal.Length = mean(Petal.Length),
    SD.Petal.Length = sd(Petal.Length))
```

El resultado se puedde ver en la tabla **??**, en el cuál se obtienen los promedios y desviaciones estándar de los largos de los pétalos. Es importante notar que al usar summarize, todas las otras variables desapareceran de la tabla.

1.3.2 group_by

La función `group_by` por si sola no genera cambios visibles en las bases de datos. Sin embargo, al ser utilizada en conjunto con `summarize` permite resumir una variable agrupada (usualmente) basada en una o más variables categóricas.

Se puede ver que para el ejemplo con el caso de las plantas del género *Iris*, el resumen que se obtiene en el caso de la tabla **??** no es tan útil considerando que tenemos tres especies presentes. Si se quiere ver el promedio del largo del pétalo por especie, se debe ocupar la función `group_by` de la siguiente forma:

Table 1.3: Resumen del promedio y desviación estándar del largo de pétalo de las flores del generi Iris.

Species	Mean.Petal.Length	SD.Petal.Length
setosa	1.462	0.1736640
versicolor	4.260	0.4699110
virginica	5.552	0.5518947

Table 1.4: Millas por galón promedio en vehiculos automáticos (am = 0) y manuales (am = 1), con los distintos tipos de cilindros

cyl	am	Eficiencia
4	0	22.90000
4	1	28.07500
6	0	19.12500
6	1	20.56667
8	0	15.05000
8	1	15.40000

```
BySpecies <- group_by(iris, Species)
Summary.Byspecies <- summarize(BySpecies, Mean.Petal.Length = mean(Petal.Length),
    SD.Petal.Length = sd(Petal.Length))
```

Esto dá como resultado la tabla **??**, con la cuál se puede ver que *Iris setosa* tiene pétalos mucho más cortos que las otras dos especies del mismo género.

1.3.2.1 group_by en más de una variable

Se puede usar la función `group_by` en más de una variable, y esto generaría un resumen anidado. Como ejemplo se usará la base de datos `mtcars` presente en R (**?**). Esta base de datos presenta una variable llamada *mpg* (miles per gallon) y una medida de eficiencia de combustible. Se resumirá la información en base a la variable *am* (que se refiere al tipo de transmisión, donde 0 es automático y 1 es manual) y al número de cilindros del motor. Para eso se utilizará el siguiente código:

```
Grouped <- group_by(mtcars, cyl, am)
Eficiencia <- summarize(Grouped, Eficiencia = mean(mpg))
```

Como puede verse en la tabla **??**, en todos los casos los autos con cambios manuales tienen mejor eficiencia de combustible. Se podría probar el cambiar el orden de las variables con las cuales agrupar y observar los distintos resultados que se pueden obtener.

1.3.3 mutate

Esta función tiene como objetivo crear variables nuevas basadas en otras variables. Es muy facil de usar, como argumento se usa el nombre de la variable nueva que se quiere crear y se realiza una operación con variables que ya estan ahí. Por ejemplo, si se continúa el trabajo con la base de datos *Iris*, al crear una nueva variable que sea la razón entre el largo del pétalo y el del sépalo, resulta lo siguiente:

Table 1.5: Tabla con diez de las observaciones de la nueva base de datos con la variable nueva creada con mutate

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	Petal.Sepal.Ratio
5.8	4.0	1.2	0.2	setosa	0.21
4.7	3.2	1.6	0.2	setosa	0.34
5.1	3.8	1.9	0.4	setosa	0.37
5.2	2.7	3.9	1.4	versicolor	0.75
6.4	2.9	4.3	1.3	versicolor	0.67
5.5	2.5	4.0	1.3	versicolor	0.73
6.5	3.0	5.8	2.2	virginica	0.89
6.0	2.2	5.0	1.5	virginica	0.83
6.1	2.6	5.6	1.4	virginica	0.92
5.9	3.0	5.1	1.8	virginica	0.86

```
DF <- mutate(iris, Petal.Sepal.Ratio = Petal.Length/Sepal.Length)
```

El resultado de esta operación es la tabla **??**. Siempre la variable que se acaba de crear aparecerá al final del data frame.

1.3.4 Pipeline (%>%)

El pipeline es un simbolo operatorio **%>%** que sirve para realizar varias operaciones de forma secuencial sin recurrir a parentesis anidados o a sobrescribir muúltiples bases de datos.

Para ver como funciona esto como un vector, supongamos que se tiene una variable a la cual se quiere primero obtener su logaritmo, luego su raíz cuadrada y finalmente su promedio con dos cifras significativas. Para realizar esto se debe seguir lo siguiente:

```
x <- c(1, 4, 6, 8)
y <- round(mean(sqrt(log(x))), 2)
```

Si se utiliza pipeline, el código sería mucho más ordenado. En ese caso, se partiría por el objeto a procesar y luego cada una de las funciones con sus argumentos si es necesario:

```
x <- c(1, 4, 6, 8)
y <- x %>% log() %>% sqrt() %>% mean() %>% round(2)
```

```
## [1] 0.99
```

El código con pipeline es mucho más fácil de interpretar a primera vista ya que se lee de izquierda a derecha y no de adentro hacia afuera. EL uso de pipeli se hace aun más importante cuando se usa con un *Data frame*, como se ve en el siguiente ejemplo:

1.3.4.1 El pipeline en data frames

POr ejemplo se quiere resumir la variable recien creada de la razón entre el sépalo y el petalo. Para hacer esto, si se partiera desde la base de datos original, tomaría varias líneas de código y la creación de múltiples bases de datos intermedias

Table 1.6: Razón pétalo sépalo promedio para las tres especies de Iris

Species	MEAN	SD
setosa	0.2927557	0.0347958
versicolor	0.7177285	0.0536255
virginica	0.8437495	0.0438064

Table 1.7: Símbolos lógicos de R y su significado

simbolo	significado	simbolo_cont	significado_cont
>	Mayor que	!=	distinto a
<	Menor que	%in%	dentro del grupo
==	Igual a	is.na	es NA
>=	mayor o igual a	!is.na	no es NA
<=	menor o igual a	| &	o, y

```
DF <- mutate(iris, Petal.Sepal.Ratio = Petal.Length/Sepal.Length)
BySpecies <- group_by(DF, Species)
Summary.Byspecies <- summarize(BySpecies, MEAN = mean(Petal.Sepal.Ratio),
    SD = sd(Petal.Sepal.Ratio))
```

Otra opción es usar paréntesis anidados, lo que se traduce en el siguiente código:

```
Summary.Byspecies <- summarize(group_by(mutate(iris, Petal.Sepal.Ratio = Petal.Length/Sepal.Length),
    Species), MEAN = mean(Petal.Sepal.Ratio), SD = sd(Petal.Sepal.Ratio))
```

Esto se simplifica mucho más al usar el pipeline, lo cual permite partir en un *Data Frame* y luego usar el pipeline. Esto permite obtener el mismo resultado que en las operaciones anteriores con el siguiente código:

```
Summary.Byspecies <- iris %>% mutate(Petal.Sepal.Ratio = Petal.Length/Sepal.Length) %>%
    group_by(Species) %>% summarize(MEAN = mean(Petal.Sepal.Ratio),
    SD = sd(Petal.Sepal.Ratio))
```

Estos tres códigos son correctos (tabla **??**), pero definitivamente el uso del pipeline da el código más conciso y fácil de interpretar sin pasos intermedios.

1.3.5 filter

Esta función permite seleccionar filas que cumplen con ciertas condiciones, como tener un valor mayor a un umbral o pertenecer a cierta clase Los símbolos más típicos a usar en este caso son los que se ven en la tabla **??**.

Por ejemplo si se quiere estudiar las características florales de las plantas del género *Iris*, pero no tomar en cuenta a la especie *Iris versicolor* se deberá usar el siguiente código:

```
data("iris")
DF <- iris %>% filter(Species != "versicolor") %>% group_by(Species) %>%
    summarise_all(mean)
```

Table 1.8: Resumen de la media de todas las características florales de las especies Iris setosa e Iris virginica

Species	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
setosa	5.006	3.428	1.462	0.246
virginica	6.588	2.974	5.552	2.026

Table 1.9: Número de plantas de cada especie con un largo de pétalo mayor a 4 y un largo de sépalo mayor a 5 centímetros

Species	N
versicolor	39
virginica	49

De esta forma se obtiene como resultado la tabla **??**. En este caso se introduce la función `summarize_all` de `summarize`, la cual aplica la función que se le da como argumento a todas las variables de la base de datos.

Por otro lado si se quiere estudiar cuántas plantas de cada especie tienen un largo de pétalo mayor a 4 y un largo de sépalo mayor a 5 se deberá usar el siguiente código:

```
DF <- iris %>% filter(Petal.Length >= 4 & Sepal.Length >= 5) %>%
    group_by(Species) %>% summarise(N = n())
```

En la tabla tabla **??** se ve que con este filtro desaparecen de la base de datos todas las plantas de *Iris setosa* y que todas menos una planta de *Iris virginica* tienen ambas características.

1.3.6 select

Esta función permite seleccionar las variables a utilizar dado que en muchos casos nos encontraremos con bases de datos con demasiadas variables y por lo tanto, se querrá reducirlas para solo trabajar en una tabla con las variables necesarias.

Con select hay varias formas de trabajar, por un lado se puede escribir las variables que se utilizarán, o restar las que no. En ese sentido estos cuatro códigos dan exactamente el mismo resultado. Esto se puede ver en la tabla **??**

```
iris %>% group_by(Species) %>% select(Petal.Length, Petal.Width) %>%
    summarize_all(mean)
```

```
iris %>% group_by(Species) %>% select(-Sepal.Length, -Sepal.Width) %>%
    summarize_all(mean)
```

```
iris %>% group_by(Species) %>% select(contains("Petal")) %>%
    summarize_all(mean)
```

```
iris %>% group_by(Species) %>% select(-contains("Sepal")) %>%
    summarize_all(mean)
```