R es una calculadora demasiado extraña

Introducción a R

Rodrigo Zepeda-Tello

2022-10-03

Discutimos por qué vale la pena hacer un curso de R y cómo instalar tanto R como RStudio en todas las plataformas.

> **Warning**
>
> Si aún no cuentas con una instalación de R y RStudio ve a la sección de instalación

Primeros pasos con R

Cálculos numéricos

R sirve como calculadora para las operaciones usuales. En él puedes hacer sumas,

```
#Esto es una suma en R
12 + 31
```

```
[1] 43
```

restas,

Figure 1: Ada Lovelace (1815-1852), la primera en diseñar un algoritmo computacional ¡y sin tener computadoras!

```
#Esto es una resta en R
3 - 4
```

```
[1] -1
```

multiplicaciones,

```
#Esto es una multiplicación en R
7*8
```

```
[1] 56
```

divisiones,

```
#Esto es una división en R
4/2
```

```
[1] 2
```

sacar logaritmos naturales ln,

```
#Para sacar logaritmo usas el comando log
log(100)
```

```
[1] 4.60517
```

o bien logaritmos en cualquier base,[1]

```
#Puedes especificar la base del logaritmo con base
log(100, base = 10)
```

```
[1] 2
```

también puedes elevar a una potencia (por ejemplo hacer 6^3),

[1] Recuerda que un logaritmo base a te dice a qué potencia b tuve que elevar a para llegar a b. Por ejemplo $\log_{10}(100) = 2$ te dice que para llegar al 100 tuviste que hacer 10^2.

```
#Así se calculan potencias
6^3
```

```
[1] 216
```

calcular la exponencial e,

```
#Para exponenciales puedes usar exp
exp(1)
```

```
[1] 2.718282
```

o bien exponenciar cualquier variable e^{-3},

```
#O bien exponenciales específicas, e^-3
exp(-3)
```

```
[1] 0.04978707
```

también puedes usar el número π.

```
#Cálculo de pi
pi
```

```
[1] 3.141593
```

No olvides que `R` usa el orden de las operaciones de matemáticas. Siempre es de izquierda a derecha con las siguientes excepciones:

1. Primero se evalúa lo que está entre paréntesis.
2. En segundo lugar se calculan potencias.
3. Lo tercero en evaluarse son multiplicaciones y divisiones.
4. Finalmente, se realizan sumas y restas.

Por ejemplo, en la siguiente ecuación

$$2 - 2 \cdot \frac{(3^4 - 9)}{(5 + 4)}$$

se resuelven primero los paréntesis $(3^4 - 9) = 81 - 9 = 72$ y $(5 + 4) = 9$; luego se resuelve la división: $\frac{72}{9} = 8$, se multiplica por el 2: $2 \cdot 8$ y finalmente se hace la resta: $2 - 8 = -6$.

Ejercicio (operaciones sin contexto)

Determina, sin evaluar, los resultados de los siguientes segmentos de código:

```
#Primer ejercicio
(9 - 3)^2 * (2 - 1) - 6
```

```
#Segundo ejercicio
6 * 2 / (7 - 3) * 5
```

```
#Tercer ejercicio
2 * 3 ^ 2 * 2 / (5 - 4) * 1 / 10
```

Evalúa para comprobar tus respuestas.

Ejercicio (NNT)

El **número (de pacientes) que es necesario tratar (NNT)** se define como la cantidad total de pacientes a quienes es necesario darles tratamiento para evitar un resultado negativo (es decir a cuántos pacientes debo darles tratamiento para que al menos uno se beneficie).

> NNT perfecto
>
> El **NNT** perfecto es cuando $NNT = 1$. ¿Por qué es éste el mejor escenario?

Dada la siguiente tabla

		Resultado	
		Si	No
Tratamiento	Si	A	B
	No	C	D

para calcular el **NNT** es necesario calcular la tasa de los eventos en el grupo experimental **EER** y en el grupo control **CER** como sigue:

$$EER = \frac{A}{A+B} \quad \text{y} \quad CER = \frac{C}{C+D}$$

finalmente, el número necesario a tratar **NNT** se define como

$$NNT = \frac{1}{|EER - CER|}$$

El ejercicio consiste en calcular en `R` el **NNT** para la siguiente tabla:

		Resultado	
		Si	No
Tratamiento	Si	234	39
	No	981	1040

Respuestas

```
i NNT = 2.69
```

Ejercicio (círculos)

Calcula en `R` el área y el perímetro de un círculo de radio 5.

Recuerda que la fórmula del área es $\pi \cdot r^2$ donde r es el radio; mientras que la del perímetro es: $\pi \cdot d$ donde d es el diámetro (= dos veces el radio).

Respuestas

```
i Área = 78.54 y Perímetro = 31.42
```

Ejercicio (R0)

El R_0 se interpreta como el promedio de casos nuevos que genera un infectado en una enfermedad infecciosa. En su forma más sencilla se puede modelar como:

$$R_0 = \text{Transmisibilidad} \times \text{Promedio de tasa de contactos} \times \text{Duración del periodo infeccioso}$$

donde la **transmisibilidad** es la probabilidad de contagio si hay un contacto entre un individuo infectado y uno susceptible, el **promedio de contactos** es el promedio de interacciones entre un infectado y un susceptible que ocurran y la **duración** es la cantidad de tiempo que un individuo infectado dura contagioso.

> Calcula en R el R_0 para una enfermedad con 10 días de duración de periodo infeccioso, un promedio de 2 contactos por día y una transmisibilidad de 0.07.

Resultado

```
i R_0 = 1.4
```

Variables

`R` es un programa orientado a objetos; esto quiere decir que `R` almacena la información en un conjunto de variables que pueden tener diferentes `clases` y opera con ellos según su clase. Por ejemplo, un conjunto de caracteres, entre comillas, es un `Character` (`R` lo piensa como texto)

```
#Un conjunto de caracteres es un char
"Hola"
```

```
[1] "Hola"
```

Un número (por ejemplo `2` tiene clase `numeric`)[2]. Hay que tener mucho cuidado con combinar floats con `Strings`:

```
#Código que sí funciona porque ambos son números
2 + 4
```

```
[1] 6
```

```
#Código que no funciona porque uno es caracter
2 + "4"
```

```
Error in 2 + "4": non-numeric argument to binary operator
```

Si lo piensas, este último error ¡tiene todo el sentido! no puedes sumar un número a un texto. ¿O qué significaría `'Felices' * 4` ?

La magia de `R` comienza con que puedes almacenar valores en variables. Por ejemplo, podemos asignar un valor a una variable:

```
#Asignamos x = 10
x <- 10
```

Hay dos formas de asignar valores, una es con la flecha de asignación ← y otra con el signo de igual:

```
#Podemos asignar valores con el signo de =
y = 6
```

Nota que, cuando realizamos operaciones, la asignación es la última que se realiza:

```
#Aquí z = 106
z <- y + x^2
```

[2] Puede ser `float`, `int`, `double` pero no nos preocuparemos por eso.

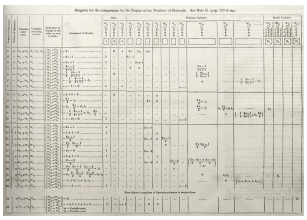


Figure 2: El algoritmo diseñado por Ada Lovelace.

Los valores que fueron asignados en las variables, `R` los recuerda y es posible calcular con ellos:

```
#Podemos realizar una suma
x + y
```

```
[1] 16
```

```
#O bien podemos realizar una multiplicación
3*y - x
```

```
[1] 8
```

Podemos preguntarnos por el valor de las variables numéricas mediante los operadores `==` (sí, son dos iguales), `!=` (que es un $\neq$) `>`, `>=`, `<=` y `<`:

```
#Podemos preguntarnos si x vale 4
x == 4
```

```
[1] FALSE
```

> **Warning**
>
> El operador de asignación también se puede utilizar al revés $2 \rightarrow x$ pero no lo hagas, por favor.

Nota que no estamos asignando el valor de `x`:

```
x
```

```
[1] 10
```

Podemos preguntarnos por diferencia:

```
x != 4
```

```
[1] TRUE
```

Así como por mayores, menores incluyendo posibles igualdades (*i.e.* los casos $\geq$ y $\leq$)

```
#Nos preguntamos si x > y
x > y
```

```
[1] TRUE
```

```
#Nos preguntamos si x >= 10
x >= 10
```

```
[1] TRUE
```

```
#Nos preguntamos si y < 6
y < 6
```

```
[1] FALSE
```

```
#O bien si y <= 6
y <= 6
```

```
[1] TRUE
```

En todos los casos los resultados han sido `TRUE` ó `FALSE`. La clase de variables que toma valores `TRUE` ó `FALSE` se conoce como booleana. Hay que tener mucho cuidado con ellas porque, puedes acabar con resultados muy extraños:

```
#MALAS PRÁCTICAS, NO HAGAS ESTO
#Cuando lo usas como número TRUE vale 1
100 + TRUE
```

```
[1] 101
```

```
#MALAS PRÁCTICAS, NO HAGAS ESTO
#Cuando lo usas como número FALSE vale 0
6*FALSE
```

```
[1] 0
```

> **Warning**
>
> Aquí puedes encontrar una lista de malas prácticas en computación a evitar.

Finalmente, nota que es posible reescribir una variable y cambiar su valor:

```
#Aquí x vale 10, como antes
x
```

```
[1] 10
```

```
#Aquí cambianos el valor de x y valdrá 0.5
x <- 0.5
x
```

```
[1] 0.5
```

Ejercicios de variables (para confundir)

Determina el valor que imprime `R` en cada caso, sin que corras los siguientes pedazos de código. Después, verifica tu respuesta con `R`:

R BÁSICO

```r
#Primer ejercicio
x <- 100
y <- 3
x > y
```

```r
#Segundo ejercicio
z <- (4 - 2)^3
z <- z + z + z
z
```

```r
#Tercer ejercicio
x <- 3
y <- 2
z <- x * y
x <- 5
y <- 10
z
```

```r
#Cuarto ejercicio
variable1 <- 1000
variable2 <- 100
variable3 <- variable1/variable2 <= 10
variable3
```

```r
#Quinto ejercicio
"2" - 2
```

```r
#Sexto ejercicio
(0.1 + 0.1 + 0.1) == 0.3
```

R INTERMEDIO

Determina, sin correr el programa, qué regresa la consola en este caso

```
x <- 2
x <- 5 + x -> y -> x
x <- x^2
x
```

```
x <- TRUE
if (x > FALSE){
  x <- x^(FALSE)^(TRUE)
} else {
  x <- x^(FALSE)^(TRUE)
}
x
```

Comprueba con la consola tus resultados; puede que encuentres respuestas poco intuitivas.

Observaciones sobre la aritmética de punto flotante

Si hiciste el penúltimo ejercicio (el cual, obviamente hiciste y comprobaste con la consola) podrás haber notado una trampa. Analicemos qué ocurre; quizá hicimos mal la suma

```
#Veamos si este lado está mal
(0.1 + 0.1 + 0.1)
```

```
[1] 0.3
```

```
#O si éste es el que tiene la trampa
0.3
```

```
[1] 0.3
```

Aparentemente no hay nada malo ¿qué rayos le pasa a `R`? La respuesta está en la aritmética de punto flotante. Podemos pedirle a `R` que nos muestre los primeros 100 dígitos de la suma `0.1 + 0.1 + 0.1`:

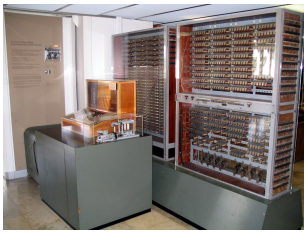


Figure 3: Réplica de la Z3, la primer computadora con punto flotante (1941).

```r
#Veamos qué pasa con la suma
options(digits = 22) #Cambiamos dígitos
(0.1 + 0.1 + 0.1)    #Sumamos
```

```
[1] 0.3000000000000000444089
```

> **Warning**
>
> El comando `options(digits = 22)` especifica que `R` debe imprimir en la consola 22 dígitos. No más. Si quieres devolverlo a como lo tenías haz `options(digits = 7)`.

¡Ahí está el detalle! `R` no sabe sumar. En general, ningún programa de computadora sabe hacerlo. Veamos otros ejemplos:

```r
4.1 - 0.1 #Debería dar 4
```

```
[1] 3.999999999999999555911
```

```r
3/10      #Debería ser 0.3
```

```
[1] 0.2999999999999999888978
```

```r
log(10^(12345), base = 10) #Debería dar 12345
```

```
[1] Inf
```

El problema está en cómo las computadoras representan los números. Ellas escriben los números en binario. Por ejemplo, 230 lo representan como 11100110 mientras que el 7 es: 111. El problema de las computadoras radica en que éstas tienen una memoria finita por lo que números muy grandes como: 12476573146709837265417 6 la computadora hace lo mejor por representarlos eligiendo el más cercano:

```r
#Nota la diferencia entre lo que le decimos a R
#y lo que resulta
x <- 12476573146709837265417 6
x
```

```
[1] 124765731467098377420800
```

> **Warning**
>
> Un error de punto flotante en la vida real ocasionó en los años noventa, la explosión del cohete `Ariane 5`. Moraleja: hay que tener cuidado y respeto al punto flotante.

No olvides cambiar la cantidad de dígitos que deseas que imprima `R` en su consola de vuelta:

```r
options(digits = 7) #Cambiamos dígitos
```

El mismo problema ocurre con números decimales cuya representación binaria es periódica; por ejemplo el $\frac{1}{10}$ en binario se representa como $0.0001100110011\overline{0011}\ldots$. Como es el cuento de nunca acabar con dicho número, `R` lo trunca y almacena sólo los primeros dígitos de ahí que, cada vez que escribes `0.1`, `R` en realidad almacene el 0.10000000000000000055511 que es *casi lo mismo* pero no es estrictamente igual. Hay que tener mucho cuidado con esta inexactitud de las computadoras (inexactitud estudiada por la rama de Análisis Numérico) pues puede generar varios resultados imprevistos.

¿Cómo checar un if?

En general lo que hacen las computadoras para comparar valores es que verifican que, en valor absoluto, el error sea pequeño. Recuerda que el valor absoluto de x, $|x|$, regresa siempre el positivo: $$ $|4| = 4$ y $|-8| = 8$

$$

Para verificar que algo es más o menos 0.3 suele usarse el valor absoluto[3] de la siguiente manera:

[3] En `R` el comando `abs` toma el valor absoluto.

```
abs( (0.1 + 0.1 + 0.1) - 0.3 ) < 1.e-6
```

```
[1] TRUE
```

donde `1.e-6` es notación corta para 0.000001 (también escrito como 1×10^{-6}). La pregunta que nos estamos haciendo es que si el error entre sumar $0.1 + 0.1 + 0.1$ y 0.3 es muy pequeño < 0.000001:

$$|(0.1 + 0.1 + 0.1) - 0.3| < 0.000001$$

Leer y almacenar variables en `R`

Para terminar esta sección, aprenderemos cómo guardar variables en `R`. Para eso, el concepto de directorio es uno de los más relevantes. En general, en computación, el directorio se refiere a la dirección en tu computadora donde estás trabajando. Por ejemplo, si estás en una carpeta en tu escritorio de nombre "Ejercicios_R" probablemente tu directorio sea '~/Desktop/Ejercicios_R/' (en Mac) o bien '~\Desktop\Ejercicios_R\' en Windows[4]. La forma de saber tu directorio (en general) es ir a la carpeta que te interesa y con clic derecho ver propiedades (o escribir `ls` en la terminal `Unix`).

[4] Windows usa backslash. Y hay toda una historia detrás de ello

`R` tiene un directorio `default` que quién sabe dónde está (depende de tu instalación, generalmente está donde tu `Usuario`). Usualmente lo mejor es elegir un directorio para cada uno de los proyectos que hagas. Para ello si estás en `RStudio` puedes utilizar `Shift+Ctrl+H` (`Shift+Cmd+H` en Mac) o bien ir a `Session > Set Working Directory > Choose Directory` y elegir el directorio donde deseas trabajar tu proyecto. Pensando que elegiste el escritorio (`Desktop` en mi computadora) notarás que en la consola aparece el comando `setwd("~/Desktop")` (o bien con '\' si eres Windows). Mi sugerencia es que copies ese comando en tu `Script` para

que, la próxima vez que lo corras ya tengas preestablecido el directorio.

```
#Si eres Mac/Linux
setwd("~/Desktop")

#Si eres Windows
setwd("C:\Users\Rodrigo\Desktop") #Rodrigo = Mi usuario
```

Podemos verificar el directorio elegido con `getwd()`:

```
getwd()
```

> **Warning**
>
> En general es buena práctica en `R` establecer, hasta arriba del `Script`, el comando de directorio. Esto con el propósito de que, cuando compartas un archivo, la persona a quien le fue compartido el archivo pueda rápidamente elegir su propio directorio en su computadora.

Probemos guardar unas variables en un archivo dentro de nuestro directorio. Para ello utilizaremos el comando `save`.

```
#Crear las variables
x <- 200
y <- 100

#Los archivos de variables de R son rda
save(x,y, file = "MisVariables.rda")
```

Si vas a tu directorio, notarás que el archivo `MisVariables.rda` acaba de ser creado. De esta forma `R` puede almacenar objetos creados en `R` que sólo `R` puede leer (más adelante veremos cómo exportar bases de datos y gráficas). Observa que en tu ambiente (si estás en `RStudio` puedes verlas en el panel 3) deben aparecer las variables que hemos usado hasta ahora:

```
Warning in rm(rojos, empty, pdata, scatter, xmin, ymin, hist_right, hist_top, :
object 'rojos' not found
```

```
Warning in rm(rojos, empty, pdata, scatter, xmin, ymin, hist_right, hist_top, :
object 'empty' not found

Warning in rm(rojos, empty, pdata, scatter, xmin, ymin, hist_right, hist_top, :
object 'pdata' not found

Warning in rm(rojos, empty, pdata, scatter, xmin, ymin, hist_right, hist_top, :
object 'scatter' not found

Warning in rm(rojos, empty, pdata, scatter, xmin, ymin, hist_right, hist_top, :
object 'xmin' not found

Warning in rm(rojos, empty, pdata, scatter, xmin, ymin, hist_right, hist_top, :
object 'ymin' not found

Warning in rm(rojos, empty, pdata, scatter, xmin, ymin, hist_right, hist_top, :
object 'hist_right' not found

Warning in rm(rojos, empty, pdata, scatter, xmin, ymin, hist_right, hist_top, :
object 'hist_top' not found

Warning in rm(rojos, empty, pdata, scatter, xmin, ymin, hist_right, hist_top, :
object 'verdes' not found

Warning in rm(rojos, empty, pdata, scatter, xmin, ymin, hist_right, hist_top, :
object 'bg' not found

Warning in rm(rojos, empty, pdata, scatter, xmin, ymin, hist_right, hist_top, :
object 'g' not found

Warning in rm(rojos, empty, pdata, scatter, xmin, ymin, hist_right, hist_top, :
object 'xhist' not found

Warning in rm(rojos, empty, pdata, scatter, xmin, ymin, hist_right, hist_top, :
object 'yhist' not found

[1] "x"     "y"     "z"     "eer"   "cer"   ".main"
```

Podemos probar sumar nuestras variables y todo funciona súper:

```
x + y #Funciona magnífico
```

```
[1] 300
```

Limpiemos el ambiente. El comando equivalente al `clear all` en `R` es un poco más complicado de memorizar:

```
#EL clear all de R
rm(list = ls())
```

Ahora, si vuelves a ver el ambiente, éste estará vacío: ¡hemos limpiado el historial! Nota que si intentamos operar con las variables, `R` ya no las recuerda:

```
x + y #Error
```

```
Error in eval(expr, envir, enclos): object 'x' not found
```

> **Warning**
>
> Así como hay que lavarse las manos antes de comer, es buen hábito limpiar todas las variables del ambiente de `R` antes de usarlo.

Podemos leer la base de datos usando `load`:

```
#Leemos las variables
load("MisVariables.rda")

#Una vez leídas podemos empezar a jugar con ellas
x + y #Ya funciona
```

```
[1] 300
```

Por último, es necesario resaltar la importancia del directorio. Para ello crea una nueva carpeta en tu escritorio de nombre `"Mi_curso_de_R"`. Mueve el archivo `"MisVariables.rda"` dentro de la carpeta. Borra todo e intenta leer de nuevo el archivo:

```
#Borramos todo
rm(list = ls())

#Intentamos leer el archivo de nuevo
load("MisVariables.rda")
```

Este error es porque `R` sigue pensando que nuestro directorio es el escritorio y está buscando el archivo ahí sin hallarlo. Para encontrarlo hay que cambiar el directorio a través de `RStudio` (ya sea `Ctrl+Shift+H` o `Session >Set Working Directory > Choose Directory`) o bien a través de comandos en `R`:

```
#Si eres Mac/Linux
setwd("~/Desktop/Mi_curso_de_R")

#Si eres Windows
setwd("C:\Users\Rodrigo\Desktop\Mi_curso_de_R") #Rodrigo = Mi usuario
```

```
Error in file.move(paste0(subdir, "/", "MisVariables.rda"), "MisVariables.rda"): could not find
```

```
#Aquí sí se puede leer
load("MisVariables.rda")
```

Ejercicio

Responde a las siguientes preguntas:

1. ¿Qué es el directorio y por qué es necesario establecerlo?
2. Si `R` me da el error `'No such file or directory'` ¿qué hice mal?

3. En `RStudio`, ¿qué hace `Session > Restart R`? ¿cuál es la diferencia con `rm(list = ls())`?
4. ¿Qué hace el comando `cat("\014")`? (*Ojo* puede que no haga nada). Si funciona, ¿cuál es la diferencia con `rm(list = ls())` y con `Restart R`?

Instalación de paquetes

Un paquete de `R` es un conjunto de funciones adicionales elaboradas por los usuarios, las cuales permiten hacer cosas adicionales en `R`. Para instalarlos requieres de una conexión a Internet (o bien puedes instalarlos a partir de un archivo, por ejemplo, mediante una `USB`). El comando de instalación es `install.packages` seguido del nombre del paquete. Por ejemplo (y por ocio) descarguemos el paquete `beepr` para hacer reproducir sonidos en la computadora[5]. Para ello:

```
install.packages("beepr")
```

```
[...]
* DONE (beepr)

The downloaded source packages are in
    '/algun/lugar/downloaded_packages'
```

Esto significa que el paquete ha sido instalado. Nos interesa usar la función `beep` que emite un sonido (`??beep` para ver la ayuda). Si la llamamos así tal cual, nos da error:

```
beep(3)
```

```
Error in beep(3): could not find function "beep"
```

`R` es incapaz de hallar la función porque aún no le hemos dicho dónde se encuentra. Para ello podemos llamar al paquete mediante la función `library` y decirle a `R` que incluya las funciones que se encuentran dentro de `beepr`:

[5] En los siguientes capítulos descargaremos paquetes más interesantes; pero no desprecies la utilidad de `beepr` yo lo he usado en múltiples ocasiones para que la computadora me avise que ya terminó de correr un código.

```
library(beepr)
beep(3) #Esto produce un sonido
```

El comando `library` le dice a `R` ¡hey, voy a usar unas funciones que creó alguien más y que están dentro del paquete `beepr`! De esta manera, al correr `beep(3)`, `R` ya sabe dónde hallar la función y por eso no arroja error.

Ejercicios

R BÁSICO

1. Instala el paquete `ggplot2` en `R` así como `ggformula`, `readr` y `readxl` (son 4 paquetes distintos).
2. Con `ggplot2` haz lo necesario para que el siguiente bloque de código te arroje una gráfica:

```
#Aquí tienes que hacer algo
#
# RELLENA AQUÍ
#

#Esto genera un histograma
set.seed(1364752)
mis.datos <- data.frame(x = rnorm(1000))

#Las siguientes funciones viven en el paquete ggplot2
ggplot(mis.datos, aes(x = x)) +
  geom_histogram(bins = 50, fill = "deepskyblue3") +
  ggtitle("Histograma generado por el código")
```

```
Error in ggplot(mis.datos, aes(x = x)): could not find function "ggplot"
```

R INTERMEDIO

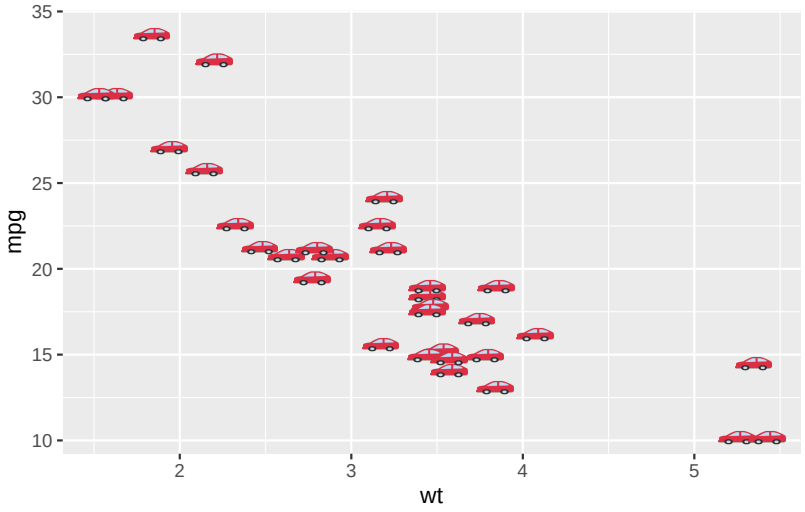
1. Instala el paquete `devtools` (para hacerlo probablemente necesites instalar más cosas en tu computadora; averigua cuáles)

2. Usa `devtools` para instalar el paquete `emoGG` desde Github.
3. Verifica que tu instalación fue correcta haciendo la siguiente gráfica:

```
library(emoGG)
```

```
Loading required package: ggplot2
```

```
ggplot(mtcars, aes(wt, mpg)) + geom_emoji(emoji="1f697")
```



Comentarios adicionales sobre el formato

Así como en el español existen reglas de gramática para ponernos todos de acuerdo y entendernos entre todos, en `R` también existen *sugerencias* a seguir para escribir tu código. Las sugerencias que aquí aparecen fueron adaptadas de las que utiliza el equipo de `Google`.

1. No escribas líneas de más de 80 caracteres (si se salió de tu pantalla, mejor continúa en el siguiente renglón).

2. Coloca espacios entre operadores `+,*,/,-,<-,=, <, <=, >, >=, ==` y usa paréntesis para agrupar:

```
#Esto no se ve muy bien
abs(3*5/(4-9)^2-60/100-888+0.1*8888-4/10*2) < 1.e-6

#Los espacios permiten distinguir el orden de las operaciones
abs( (3 * 5) / (4 - 9)^2 - 60 / 100 - 888
      + (0.1 * 8888) - (4 / 10) * 2 ) < 1.e-6
```

3. Intenta alinear la asignación de variables para legibilidad:

```
#Esto no tanto
altura <- 1.80
peso <- 80
edad <- 32

#Esto se ve bien
altura <- 1.80
peso   <- 80
edad   <- 32
```

4. Utiliza nombres que evoquen la variable que representas

```
#Cuando regreses a esto no sabrás ni qué
x <- 10
y <- 2
z <- 3.14
W <- z * x^y #¿Qué calculé?

#Es mejor especificar la variable
radio        <- 10
potencia     <- 2
pi_aprox     <- 3.14
area_circulo <- pi_aprox * radio^potencia
```

5. No utilices un nombre demasiado similar para cosas diferentes.

```r
#Aquí, seguro eventualmente te vas a equivocar
altura <- 10   #Altura del edificio
Altura <- 1.8  #Mi altura
ALTURA <- 2000 #La altitud de la CDMX

#Siempre elegir nombres claros, aunque largos
altura.edificio <- 10   #Altura del edificio
altura.Rodrigo  <- 1.8  #Mi altura
altura.CDMX     <- 2000 #La altitud de la CDMX
```

6. Comenta:

```r
#¿Qué hace esto?
x <- 168
x <- x/100
y <- 71.2
print(y/x^2)

#Es mejor así
altura <- 168        #en centímetros
altura <- altura/100 #en metros
peso   <- 71.2       #peso en kg
print(peso/altura^2) #índice masa corporal
```

Figure 4: Trad: Un periodista se acerca a un programador a preguntarle ¿qué hace que un código sea malo? - Sin comentarios.

7. Siempre pon las llamadas a los paquetes y el directorio al inicio de tu archivo para que otro usuario sepa qué necesita.

Código limpio y legible:

```r
#Asumiendo aquí inicia el archivo:
setwd("Mi directorio")

#Llamamos la librería
library(beepr)
library(tidyverse)

#Analizamos una base de datos de R
data(iris) #Base de datos de flores
```

```r
#Agrupamos la base por especie
iris.agrupada <- group_by(iris, Species)

#Obtenemos la media por longitud de sépalo
iris.media    <- summarise(iris.agrupada, SL.mean = mean(Sepal.Length))

#Avisa que ya terminó
beep(5)
```

es siempre preferible a código escrito *con prisas* :

```r
data(iris);setwd("Mi directorio")
library(tidyverse);x<-group_by(iris,Species  )
#Aquí hacemos esto
iris.means=summarise( x,SL.mean=mean(Sepal.Length));library(beepr);beep(5)#FIN
```

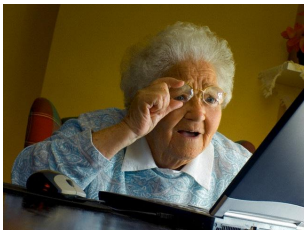


Figure 5: Yo, leyendo mi código no comentado y con mala edición 6 meses después de haberlo hecho.

Siempre escribe tu código pensando que alguien más (y ese alguien más puedes ser tú) va a leerlo. ¡No olvides comentar!

Continuación

¡Felicidades! Ya terminaste esta sección. Puedes ir a:

- Graficación con `ggplot2`.

Ejercicios de cierre de la sección

R Básico

1. El siguiente código crea una variable que se llama `enfermedad` a la cual se le asigna el valor (caracter) `"Polio"`. Sin embargo el código no corre. Arréglalo para que funcione

```r
enfermedad == POLIO
```

2. ¿Cuál es la diferencia entre `R` y `RStudio`?

3. Corre el siguiente código el cuál se encarga de generar una variable de nombre `viento`, crear un archivo `MIVIENTO.rda`, guardarlo en otro directorio y decirte en qué directorio lo guardó. Cambia el directorio con `setwd` y lee el archivo usando `load`

```
#Genera una variable viento y la guarda en un archivo rda. Córreme completo
set.seed(2734)
directorio <- tempdir()
viento     <- rexp(1)
save(viento, file = file.path(directorio, "MIVIENTO.rda"))
rm(viento)
message(paste("Guardé MIVIENTO.rda en el directorio;",directorio))
#Ahora usa setwd para ir al directorio y leer "MIVIENTO.rda"
```

4. El siguiente código se encarga de clasificar los resultados de un test de `A1C` (porcentaje) de una persona en tres: `diabetes` (si `AIC` es mayor o igual a 6.5%), `prediabetes` (5.7 a 6.4%) y `normal` (menor a 5.7%). Sin embargo el código tiene varios errores y sin importar la persona a todos les dice que tienen diabetes. Corrige el código para que funcione

```
#Aquí cambia el usuario cambia el nivel de AIC según el resultado de cada persona:
nivel_AIC <- 5.4

#Hacemos la clasificación:
clasif    <- "diabetes"
if (nivel_AIC > 6.5){
  clasif <- "diabetes"
} else if (nivel_AIC < 6.5 & nivel_AIC > 5.7){
  clasif1 <- "prediabetes"
} else {
  clasif2 <- "normal"
}

message(paste0("Sus niveles corresponden a ", clasif))
```

5. ¿Qué significan los signos de `+` en el siguiente input de la consola?

```
x <-
+  "Ho
+  la"
```

a. El primer signo de `+` indica que a lo que estuviera en `x` se le agregan los caracteres `"Ho"`.
b. El segundo signo de `+` indica concatenación de caracteres y está juntando el `Ho` con el `la` para formar un `Hola`.
c. El segundo signo de `+` está dentro del caracter por lo que estamos escribirndo `Ho+la`.
d. El signo de `+` le indica al usuario que hay un error en la línea previa.
e. El signo de `+` le indica al usuario que la línea previa está incompleta.

6. ¿Qué es el directorio que ponemos con `setwd`?

a. Es la carpeta donde vive `R`.
b. Es la carpeta donde viven los paquetes de `R`.
c. Es la carpeta donde viven los archivos que estoy usando para mi proyecto actual en `R`.

7. Genera un `RScript` de nombre `IMC.R` en el cual las personas almacenen en una variable su `peso` (kilos), en otra su `altura` (metros) y les calcule su índice de masa corporal `IMC` con la siguiente fórmula:

$$\text{IMC} = \frac{\text{Peso}}{\text{Altura}^2}$$