Evaluación "tidy":

Programando con ggplot2 y dplyr

Julio 2019

Traducido por Leonardo Collado-Torres

@fellgernon
lcolladotor@gmail.com
lcolladotor.github.io

Desarrollado por Hadley Wickham para rstudio::conf(2019)
@hadleywickham
Chief Scientist, RStudio

Escribir funciones

Regla de tres: crea una función si haz copiado y pegado tres veces

```
(df$a - min(df$a)) / (max(df$a) - min(df$a))
(df$b - min(df$b)) / (max(df$b) - min(df$b))
(df$c - min(df$c)) / (max(df$c) - min(df$c))
(df$d - min(df$d)) / (max(df$d) - min(df$c))
```

Regla de tres: crea una función si haz copiado y pegado tres veces

```
(df$a - min(df$a)) / (max(df$a) - min(df$a))
(df$b - min(df$b)) / (max(df$b) - min(df$b))
(df$c - min(df$c)) / (max(df$c) - min(df$c))
(df$d - min(df$d)) / (max(df$d) - min(df$c))
```

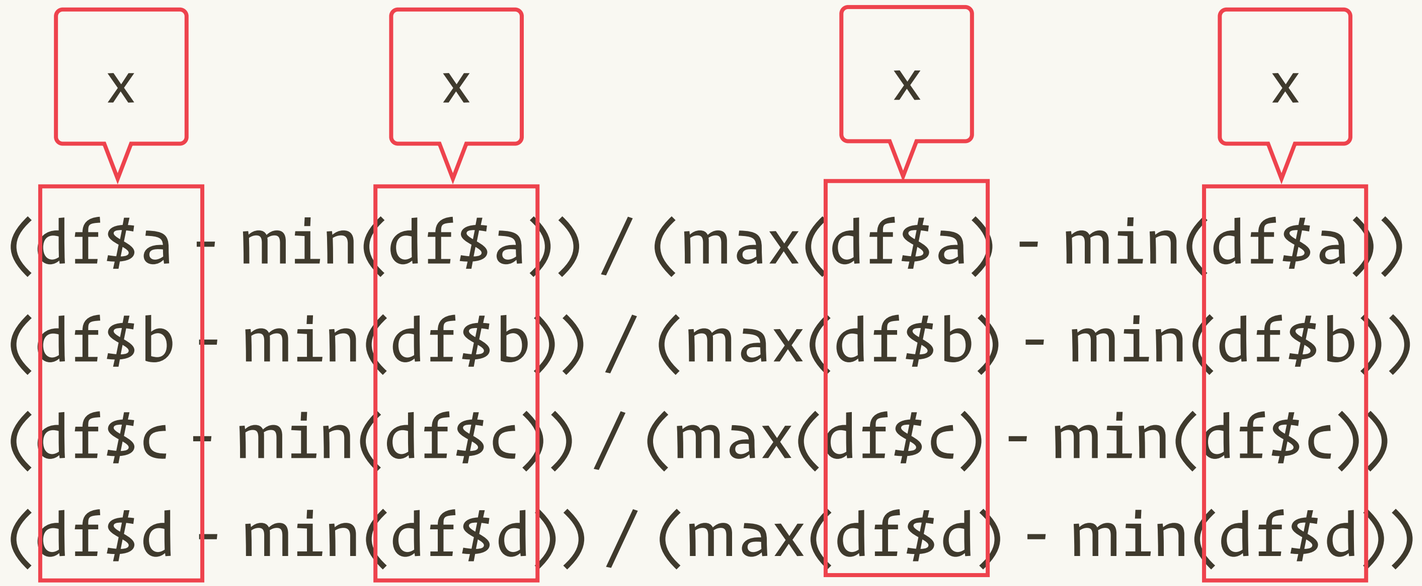
Regla de tres: crea una función si haz copiado y pegado tres veces

```
(df$a - min(df$a)) / (max(df$a) - min(df$a))
(df$b - min(df$b)) / (max(df$b) - min(df$b))
(df$c - min(df$c)) / (max(df$c) - min(df$c))
(df$d - min(df$d)) / (max(df$d) - min(df$d))
```

Primero, identifica las parques que podrían cambiar

```
(df$a - min(df$a)) / (max(df$a) - min(df$a))
(df$b - min(df$b)) / (max(df$b) - min(df$b))
(df$c - min(df$c)) / (max(df$c) - min(df$c))
(df$d - min(df$d)) / (max(df$d) - min(df$d))
```

Luego dales nombres



Crear el templado de la función

```
rescale01 <- function(x) {

}
```

Luego copia un ejemplo

```
rescale01 <- function(x) {
 (df$a - min(df$a)) / (max(df$a) - min(df$a))
}
```

Ahora usa la variable

```
rescale01 <- function(x) {
  (x - min(x)) / (max(x) - min(x))
}
```

Y tal vez cambia el código un poco

```
rescale01 <- function(x) {
  rng <- range(x)
  (x - rng[1]) / (rng[2] - rng[1])
}
```

Ahora agrega más casos

```
rescale01 <- function(x) {
  rng <- range(x, na.rm = TRUE, finite = TRUE)
  (x - rng[1]) / (rng[2] - rng[1])
}
```

Regla de tres: crea una función si haz copiado y pegado tres veces

```
(df$a - min(df$a)) / (max(df$a) - min(df$a))
(df$b - min(df$b)) / (max(df$b) - min(df$b))
(df$c - min(df$c)) / (max(df$c) - min(df$c))
(df$d - min(df$d)) / (max(df$d) - min(df$d))
```

Regla de tres: crea una función si haz copiado y pegado tres veces

```
rescale01(df$a)
rescale01(df$b)
rescale01(df$c)
rescale01(df$d)
```

¿Por qué crear una función? Porque una función:

1. Previene inconsistencias
2. Enfatiza que es lo que cambia
3. Facilita los cambios
4. Puede tener un nombre informativo

Motivación

Tratemos con un poco de código de dplyr

```
df %>% group_by(x1) %>% summarise(mean = mean(y1))
df %>% group_by(x2) %>% summarise(mean = mean(y2))
df %>% group_by(x3) %>% summarise(mean = mean(y3))
df %>% group_by(x4) %>% summarise(mean = mean(y4))
```

Tu turno

Identifica las partes que cambias.

Dales nombres.

Crea una función.

¿Por qué no funciona?

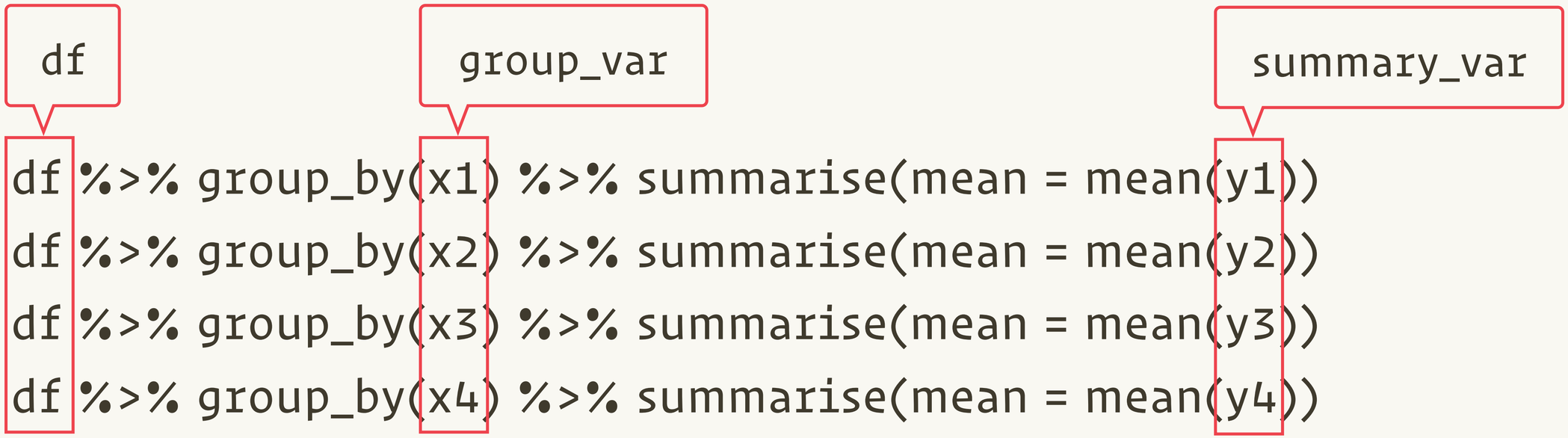
Tratemos con un poco de código de dplyr

```
df %>% group_by(x1) %>% summarise(mean = mean(y1))
df %>% group_by(x2) %>% summarise(mean = mean(y2))
df %>% group_by(x3) %>% summarise(mean = mean(y3))
df %>% group_by(x4) %>% summarise(mean = mean(y4))
```

Primero identifica las partes que cambian

```
df %>% group_by(x1) %>% summarise(mean = mean(y1))
df %>% group_by(x2) %>% summarise(mean = mean(y2))
df %>% group_by(x3) %>% summarise(mean = mean(y3))
df %>% group_by(x4) %>% summarise(mean = mean(y4))
```

Luego dales nombres



Ahora crea una función

```
grouped_mean <- function(df, group_var, summary_var) {
 df %>%
   group_by(group_var) %>%
   summarise(mean = mean(summary_var))
}
```

No funciona 😭

```
grouped_mean <- function(df, group_var, summary_var) {
 df %>%
   group_by(group_var) %>%
   summarise(mean = mean(summary_var))
}

grouped_mean(mtcars, cyl, mpg)
#> Error: Column `group_var` is unknown
```

Vocabulario

Necesitamos un poco de vocabulario nuevo

Evaluado usando las reglas de R usuales

```
(x - min(x)) / (max(x) - min(x))
```

```
mtcars %>%
  group_by(cyl) %>%
  summarise(mean = mean(mpg))
```

Automáticamente **citado (quoted)** y evaluado de una forma "no estándar"

Ya estás familiarizado con la siguiente idea

```
df <- data.frame(
  y = 1,
  var = 2
)

df$y



var <- "y"
df$var
```

¡Predice el resultado!

$ automáticamente agrega citas al nombre de la variable

```
df <- data.frame(
  y = 1,
  var = 2
)

df$y
#> [1] 1

var <- "y"
df$var
#> [1] 2
```

Si quieres referirte a ellas de forma indirecta, tienes que usar [[

```
df <- data.frame(
  y = 1,
  var = 2
)

var <- "y"
df[[var]]
#> [1] 1
```

	Citado (quoted)	Evaluado
Directo	`df$y`	`???`
Indirecto	`???`	`var <- "y"` `df[[var]]`

	Citado	Evaluado
Directo	`df$y`	`df[["y"]]`
Indirecto	`???`	`var <- "y"` `df[[var]]`

	Citado	Evaluado
Directo	`df$y`	`df[["y"]]`
Indirecto		`var <- "y"` `df[[var]]`

Identifica cuales argumentos son automáticamente citados

```
library(MASS)

mtcars2 <- subset(mtcars, cyl == 4)

with(mtcars2, sum(vs))
sum(mtcars2$am)

rm(mtcars2)
```

¿No puedes determinarlo? Intenta correr el código

```
library(MASS)
#> Works

MASS
#> Error: object 'MASS' not found

# -> El primer argumento de library() es citado
```

¿No puedes determinarlo? Intenta correr el código

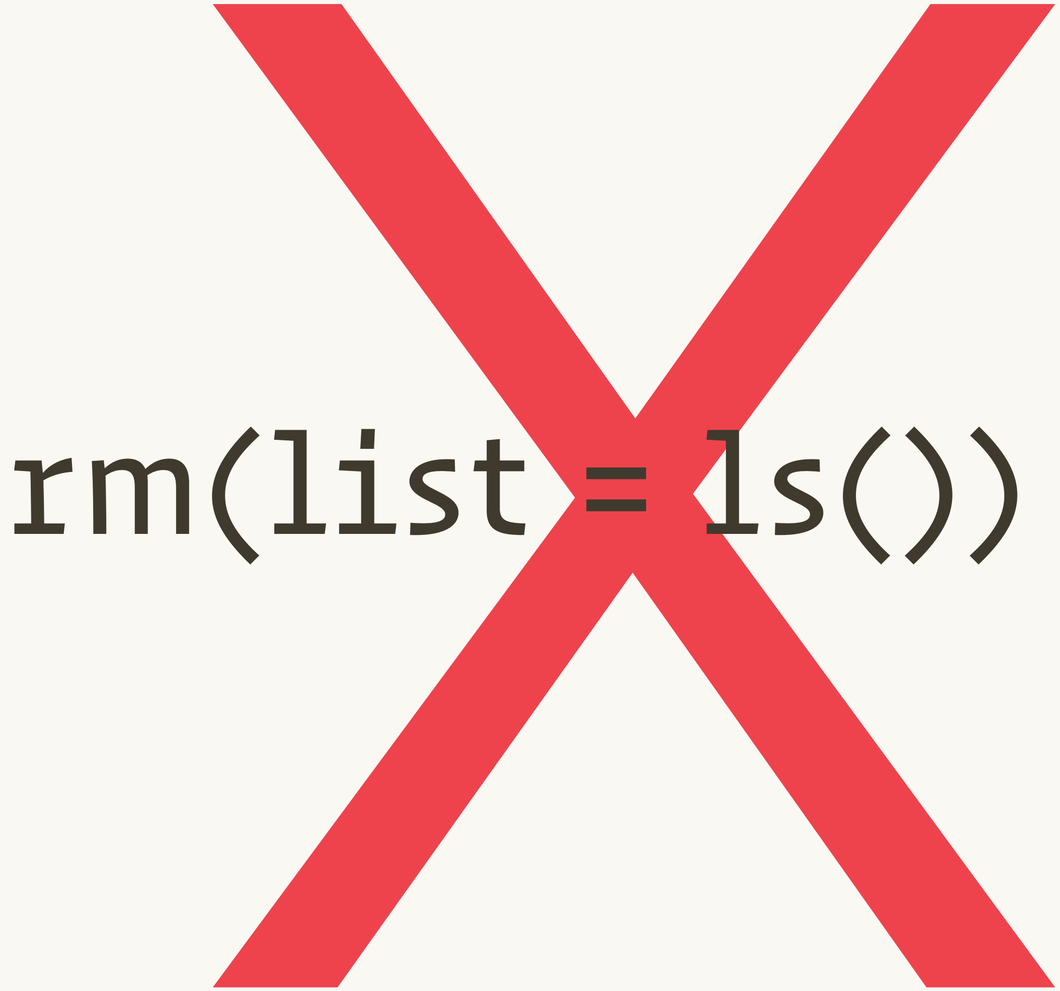
```
subset(mtcars, cyl == 4)
#> Funciona

cyl == 4
#> Error: object 'cyl' not found

# -> El segundo argumento de subset() es citado
```

Ahora podemos identificar a los argumentos citados

```
library(MASS)

mtcars2 <- subset(mtcars, cyl == 4)

with(mtcars2, sum(vs))
sum(mtcars2$am)

rm(mtcars2)
```

R base tiene 3 formas primarias de "**des-citar**" ("unquote")

Citado/Directo	Evaluado/Indirecto
`df$y`	`x <- "y"` `df[[x]]`
`library(MASS)`	`x <- "MASS"` `library(x, character.only = TRUE)`
`rm(mtcars)`	`x <- "mtcars"` `rm(list = x)`



https://www.tidyverse.org/articles/2017/12/workflow-vs-script/

Identifica cuáles argumentos son automáticamente citados

```
library(tidyverse)

mtcars %>% pull(am)

by_cyl <- mtcars %>%
  group_by(cyl) %>%
  summarise(mean = mean(mpg))

ggplot(by_cyl, aes(cyl, mpg)) +
  geom_point()
```

Identifica cuáles argumentos son automáticamente citados

```
library(tidyverse)

mtcars %>% pull(am)

by_cyl <- mtcars %>%
  group_by(cyl) %>%
  summarise(mean = mean(mpg))

ggplot(by_cyl, aes(cyl, mpg)) +
  geom_point()
```

	Citado	Evaluado	Limpio (Tidy)
Directo	`df$y`	`df[["y"]]`	`pull(df, y)`
Indirecto	😢	`var <- "y"` `df[[var]]`	???

	Citado	Evaluado	Limpio (Tidy)
Directo	`df$y`	`df[["y"]]`	`pull(df, y)`
Indirecto	😢	`var <- "y"` `df[[var]]`	`var <- quo(y)` `pull(df, !!var)`

En cualquier lugar del tidyverse usa !! para des-citar (unquote)

Pronunciado bang-bang

```
x_var <- quo(cyl)
y_var <- quo(mpg)

by_cyl <- mtcars %>%
  group_by(!!x_var) %>%
  summarise(mean = mean(!!y_var))

ggplot(by_cyl, aes(!!x_var, !!y_var)) +
  geom_point()
```

```r
x_var <- "cyl"
y_var <- "mpg"

by_cyl <- mtcars %>%
  group_by("cyl") %>%
  summarise(mean = mean("mpg"))
```

Envolviendo funciones que *citan*

Nuevo: Identifica arguments citados vs evaluados

```
df %>% group_by(x1) %>% summarise(mean = mean(y1))
df %>% group_by(x2) %>% summarise(mean = mean(y2))
df %>% group_by(x3) %>% summarise(mean = mean(y3))
df %>% group_by(x4) %>% summarise(mean = mean(y4))
```

Nuevo: Identifica arguments citados vs evaluados

```
df %>% group_by(x1) %>% summarise(mean = mean(y1))
df %>% group_by(x2) %>% summarise(mean = mean(y2))
df %>% group_by(x3) %>% summarise(mean = mean(y3))
df %>% group_by(x4) %>% summarise(mean = mean(y4))
```

Luego identifica las partes que podrían cambiar

```
df %>% group_by(x1) %>% summarise(mean = mean(y1))
df %>% group_by(x2) %>% summarise(mean = mean(y2))
df %>% group_by(x3) %>% summarise(mean = mean(y3))
df %>% group_by(x4) %>% summarise(mean = mean(y4))
```

Estas partes se convierten en argumentos

df

group_var

summary_var

```
df %>% group_by(x1) %>% summarise(mean = mean(y1))
df %>% group_by(x2) %>% summarise(mean = mean(y2))
df %>% group_by(x3) %>% summarise(mean = mean(y3))
df %>% group_by(x4) %>% summarise(mean = mean(y4))
```

Ahora escribe la función templado e identifica los argumentos citados

```
grouped_mean <- function(df, group_var, summary_var) {



  df %>%
   group_by(group_var) %>%
   summarise(mean = mean(summary_var))
}
```

Nuevo: Envuelve cada argumento citado en enquo()

```
grouped_mean <- function(df, group_var, summary_var) {
  group_var <- enquo(group_var)
  summary_var <- enquo(summary_var)

  df %>%
   group_by(group_var) %>%
   summarise(mean = mean(summary_var))
}
```

Nuevo: Y después des-cita (unquote) con !!

```
grouped_mean <- function(df, group_var, summary_var) {
  group_var <- enquo(group_var)
  summary_var <- enquo(summary_var)

  df %>%
   group_by(!!group_var) %>%
   summarise(mean = mean(!!summary_var))
}
```

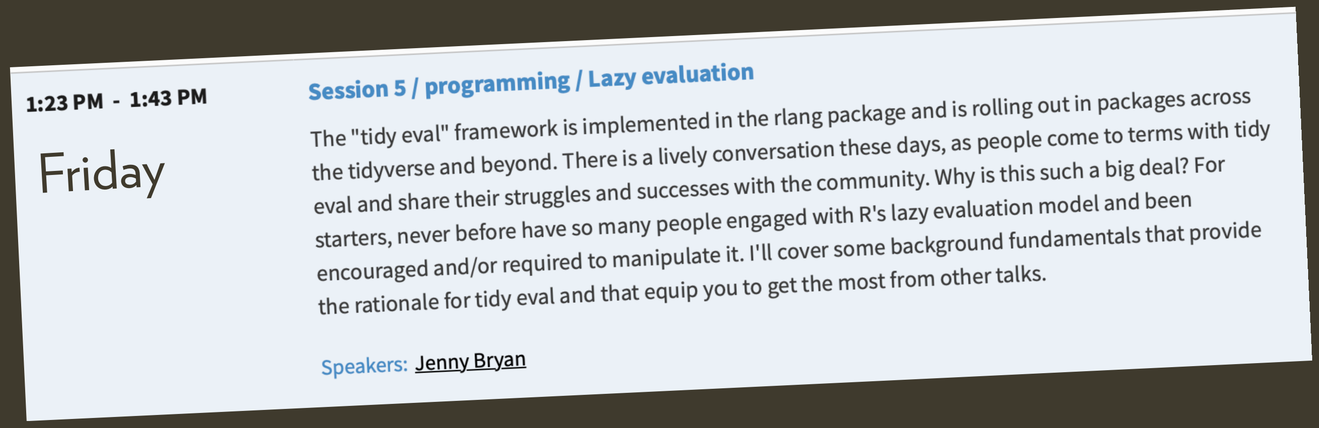
Usa la expresión almacenada en la variable, y no "summary_var" de forma literal

```r
grouped_mean(mtcars, cyl, mpg)

grouped_mean <- function(df, group_var, summary_var) {
  group_var <- enquo(group_var)
  summary_var <- enquo(summary_var)

  df %>%
    group_by(!!group_var) %>%
    summarise(mean = mean(!!summary_var))
}
```

```
grouped_mean(mtcars, cyl, mpg)

grouped_mean <- function(df, group_var, summary_var) {
  group_var <- quo(cyl)
  summary_var <- quo(mpg)

  df %>%
   group_by(!!group_var) %>%
   summarise(mean = mean(!!summary_var))
}
```

```
grouped_mean(mtcars, cyl, mpg)

grouped_mean <- function(df, group_var, summary_var) {



  df %>%
    group_by(cyl) %>%
    summarise(mean = mean(mpg))
}
```

¿Vale la pena?



1:23 PM - 1:43 PM

Friday

Session 5 / programming / Lazy evaluation

The "tidy eval" framework is implemented in the rlang package and is rolling out in packages across the tidyverse and beyond. There is a lively conversation these days, as people come to terms with tidy eval and share their struggles and successes with the community. Why is this such a big deal? For starters, never before have so many people engaged with R's lazy evaluation model and been encouraged and/or required to manipulate it. I'll cover some background fundamentals that provide the rationale for tidy eval and that equip you to get the most from other talks.

Speakers: Jenny Bryan

Nos ahorra mucho código que tendríamos que escribir

```
filter(diamonds, x > 0 & y > 0 & z > 0)


# vs
diamonds[
  diamonds$x > 0 &
  diamonds$y > 0 &
  diamonds$z > 0,
]
```

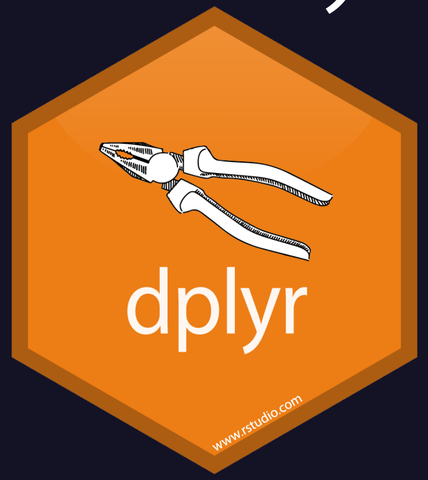
Nos ahorra mucho código que tendríamos que escribir

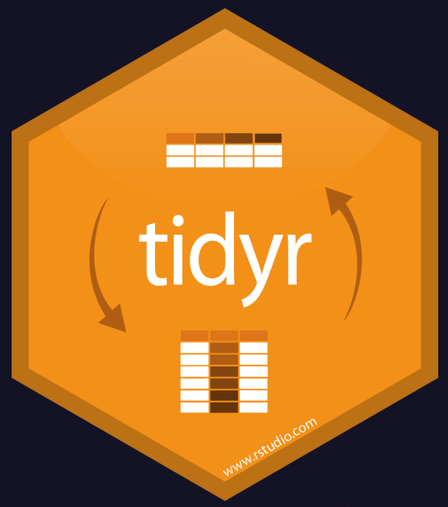
```
filter(diamonds, x > 0 & y > 0 & z > 0)


# vs
diamonds[
  diamonds["x"] > 0 &
  diamonds["y"] > 0 &
  diamonds["z"] > 0,
]
```

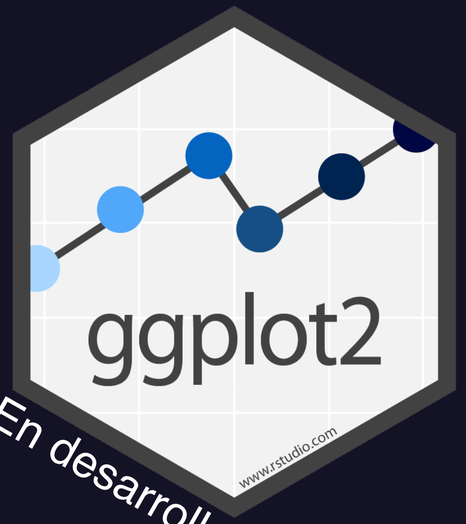
Nos permite traducir el código a otros lenguajes

```
mtcars_db %>%
  filter(cyl > 2) %>%
  select(mpg:hp) %>%
  head(10) %>%
  show_query()

#> SELECT `mpg`, `cyl`, `disp`, `hp`
#> FROM `mtcars`
#> WHERE (`cyl` > 2.0)
#> LIMIT 10
```

Evaluación "Tidy" (limpia) = evaluación no-estandar con principios (principles NSE)







En desarrollo

Ahora un poco de teoría ~~de juegos~~

1. El código de R es un árbol
2. Des-citar construye árboles
3. Ambientes ligan nombres a valores

Práctica

Reduce las copias aquí

```
df <- data.frame(
  g = rep(c("a", "b", "c"), c(3, 2, 2)),
  b = runif(7),
  a = runif(7),
  c = runif(7)
)

summarise(df, mean = mean(a), sd = sd(a), n = n())
summarise(df, mean = mean(b), sd = sd(b), n = n())
summarise(df, mean = mean(c), sd = sd(c), n = n())
```

```r
stat_sum <- function(df, var) {
  var <- enquo(var)

  summarise(df,
    mean = mean(!!var),
    sd = sd(!!var),
    n = n()
  )
}
```

Tu turno

```
# Este código es frecuentemente usado para calcular
# la proporción de una suma agrupada.
# Completa la siguiente función para simplificar
# este patrón útil
mtcars %>% count(cyl) %>% mutate(prop = n / sum(n))


prop <- function(df, x = n) {
  x <- enquo(x)
  ...
}
```

```
prop <- function(df, x = n) {
  x <- enquo(x)
  df %>% mutate(prop = !!x / sum(!!x, na.rm = TRUE))
}
```

```
prop <- function(df, x = n) {
  x <- enquo(x)
  df %>% mutate(prop = prop.table(!!x))
}
```

```r
count_and_prop <- function(df, ..., sort = TRUE) {
  if ("n" %in% names(df)) {
    # ¿¿¿hacer algo???
  }

  df %>%
    count(..., sort = sort) %>%
    mutate(n = n / sum(n))
}
```

Crea una función que se pueda re-usar para este patrón

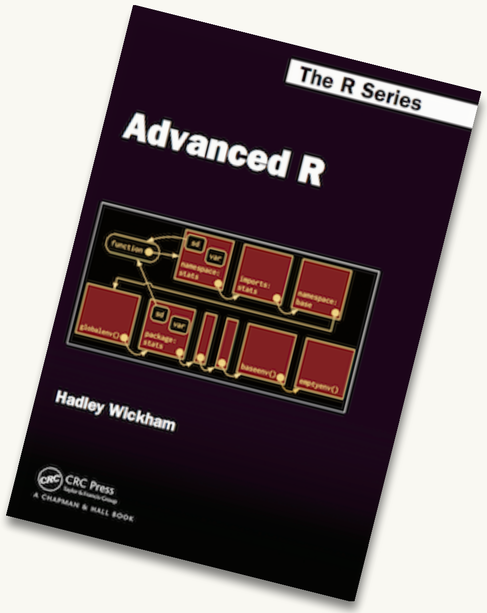
```
counts <- starwars %>%
  group_by(g = homeworld) %>%
  summarise(n = n()) %>%
  head(10) %>%
  mutate(g = reorder(g, n))

counts %>%
  ggplot(aes(g, n)) +
  geom_col() +
  coord_flip() +
  xlab("homeworld")

# Checa el templado en la siguiente diapositiva
```

https://twitter.com/JustTheSpring/status/1082515899821617152

```r
top_n <- function(df, x, n = 10) {

}

# Reto: ¿puedes cambiar el caso base para
# manejar mejor los empates?
```

Aprende más

Teoría



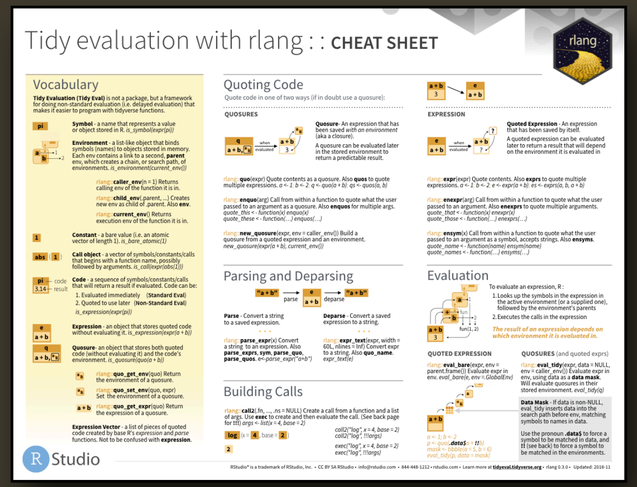
https://adv-r.hadley.nz/expressions.html
https://adv-r.hadley.nz/quasiquotation.html
https://adv-r.hadley.nz/evaluation.html

https://youtu.be/nERXS3ssntw

Práctica

https://tidyeval.tidyverse.org

(siguen desarrollando esta parte)


Tidy evaluation with rlang : : CHEAT SHEET

Vocabulary

Quoting Code

Parsing and Deparsing

Building Calls

Evaluation

Revisemos algunos cambios desde enero del 2019

https://ryo-n7.github.io/2019-07-21-user2019-reflections/