

Clase 3.0

Scripts, funciones y control de flujo

Marcos Rosetti y Luis Pacheco-Cobos

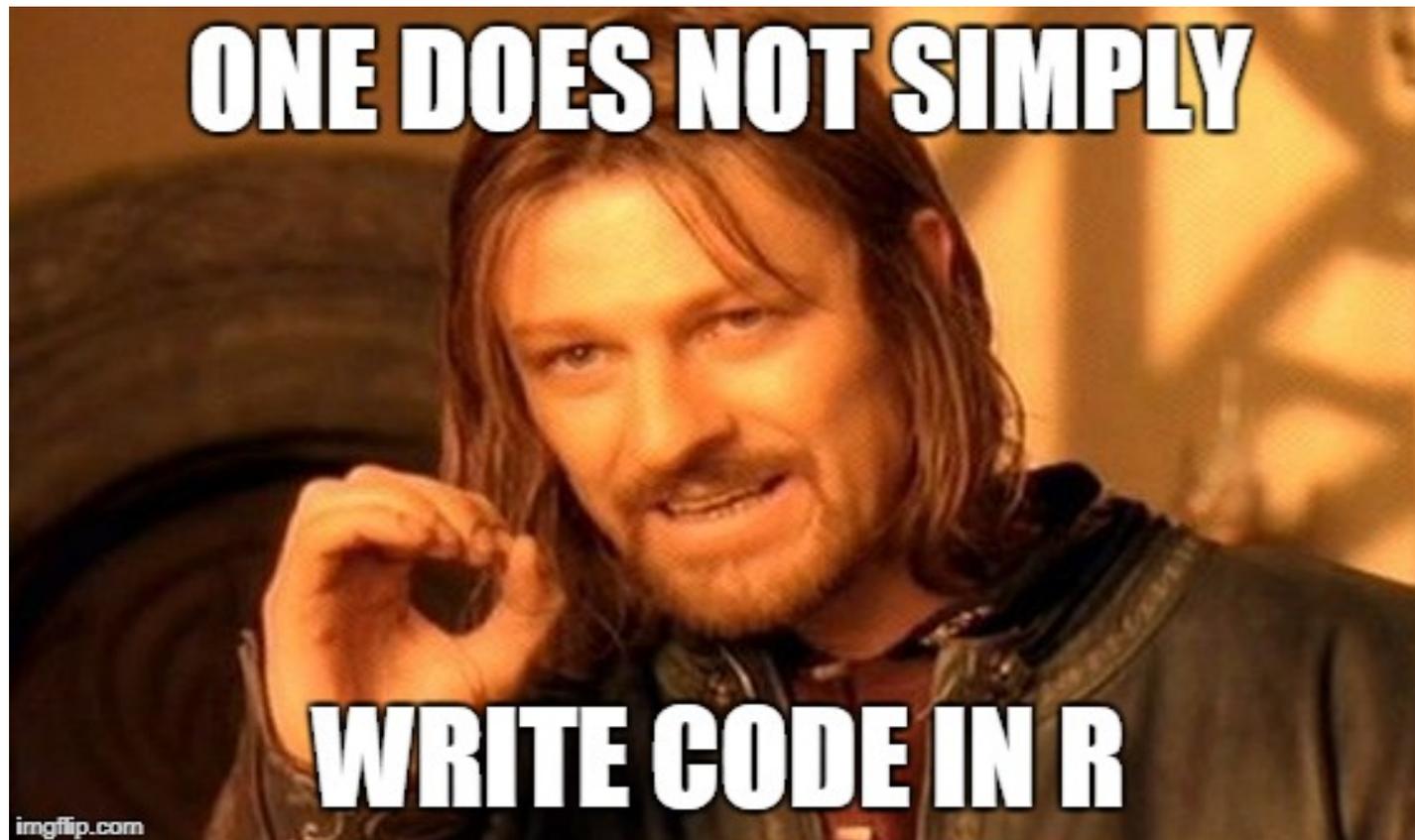
Estadística y Manejo de Datos con R (EMDR) — Virtual

Scripts

Scripts

- ¿Para qué elaborar un *script*?
 - Automatizar un código que queremos correr múltiples veces.
 - Crear y guardar código que sirve para múltiples propósitos.
 - Organizar en módulos editables un proceso largo y complejo que rebasa las capacidades de la línea de comando.

Scripts



Scripts

Code	View	Plots	Session	Build	Debug	Profile
Insert Chunk						Ctrl+Alt+I
Jump To...						Alt+Shift+J
Go To File/Function...						Ctrl+.
Show Document Outline						Ctrl+Shift+O
Show Diagnostics						
Go To Help						
Go To Function Definition						
Extract Function						Ctrl+Alt+X
Extract Variable						Ctrl+Alt+V
Rename in Scope						Ctrl+Alt+Shift+M
Reflow Comment						Ctrl+Shift+/ Ctrl+Shift+C
Comment/Uncomment Lines						Ctrl+Shift+C
Insert Roxygen Skeleton						Ctrl+Alt+Shift+R
Reindent Lines						Ctrl+I
Reformat Code						Ctrl+Shift+A
Run Selected Line(s)						Ctrl+Enter
Re-Run Previous						Ctrl+Shift+P
Run Region						▶
Run Selection as Local Job						
Send to Terminal						Ctrl+Alt+Enter
Source						Ctrl+Shift+S
Source File...						Ctrl+Alt+G

Scripts

```
points(Animals[rownames(Animals) == "African elephant", ],  
       pch = 8, col = "red", cex = 2)
```



```
mPG <- PlantGrowth %>%  
  group_by(group) %>%  
  summarize(mw = mean(weight), sdw = sd(weight))
```



```
ggplot(data = diamonds , aes(x = price , y = carat , color = color )) +  
  geom_point() +  
  facet_grid(~ cut) +  
  xlab("Precio") +  
  ylab("Carats")
```



Scripts

- Emplea nombres descriptivos para tus funciones.

```
# Good  
fit_models.R  
utility_functions.R  
  
# Bad  
foo.r  
stuff.r
```

Scripts

- Emplea nombres descriptivos para tus objetos.

```
# Good
```

```
day_one
```

```
day_1
```

```
# Bad
```

```
first_day_of_the_month
```

```
DayOne
```

```
dayone
```

```
djml
```

Scripts



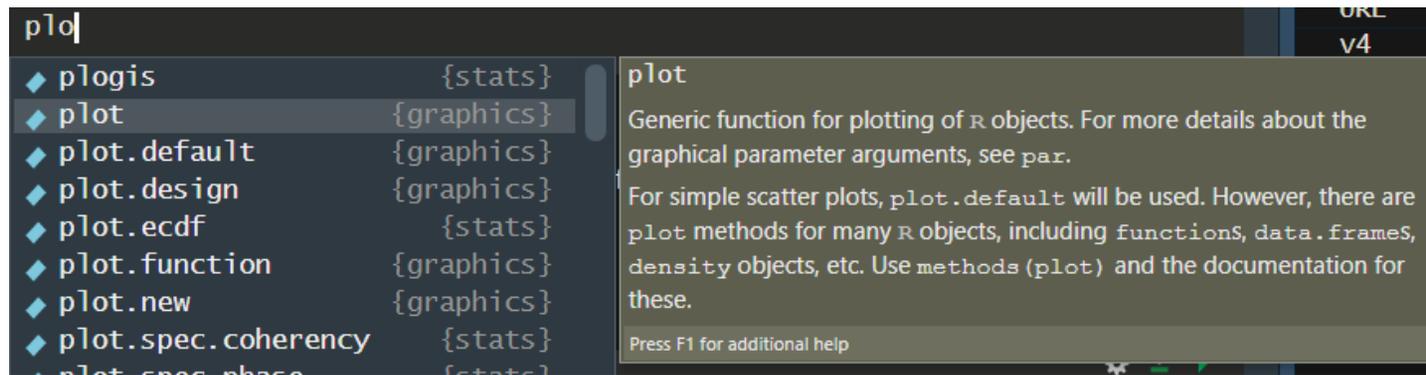
Scripts

- Evita usar o renombrar funciones existentes.

```
# Bad  
T <- FALSE  
c <- 10  
mean <- function(x) { sum(x) }
```

Scripts

- Usa la función de autosugerencia* para completar los nombres de las funciones, variables, etc.
- *Oprime la tecla tabulador después de iniciar la escritura.



The screenshot shows the RStudio interface with the text 'plot' entered in the console. A dropdown menu is visible, listing several functions related to plotting, each with its class in curly braces. The 'plot' function is highlighted. To the right of the dropdown, a tooltip provides detailed documentation for the 'plot' function, including its purpose and usage examples. The tooltip text reads: 'plot: Generic function for plotting of R objects. For more details about the graphical parameter arguments, see par. For simple scatter plots, plot.default will be used. However, there are plot methods for many R objects, including functions, data.frames, density objects, etc. Use methods(plot) and the documentation for these. Press F1 for additional help'.

```
plot|
```

◆ plot	{stats}
◆ plot	{graphics}
◆ plot.default	{graphics}
◆ plot.design	{graphics}
◆ plot.ecdf	{stats}
◆ plot.function	{graphics}
◆ plot.new	{graphics}
◆ plot.spec.coherency	{stats}
◆ plot.spec.phase	{stats}

plot
Generic function for plotting of R objects. For more details about the graphical parameter arguments, see `par`.
For simple scatter plots, `plot.default` will be used. However, there are `plot` methods for many R objects, including `functions`, `data.frames`, `density` objects, etc. Use `methods(plot)` and the documentation for these.
Press F1 for additional help

Scripts

- Espacios entre operadores y después de cada coma hacen el código más legible.

```
# Good  
average <- mean(feet / 12 + inches, na.rm = TRUE)  
  
# Bad  
average<-mean(feet/12+inches,na.rm=TRUE)
```

Scripts

- Excepciones

```
plot(x) # Good  
plot (x) # Bad  
plot( x ) # Bad
```

```
base::get # Good  
base :: get # Bad
```

Scripts

- Conflictos entre paquetes
- Hay paquetes que tienen funciones con el mismo nombre.
 - (p.e. `summarise` de `dplyr` y `summarise` de `MASS`)
- Cuando cargas un paquete, éste anula la función del paquete previo.
- Podemos hacer referencia a una función sin cargar la biblioteca de funciones (*library*) con el operador `::`

```
dplyr::summarise()
```

```
MASS::summarise()
```

Scripts

- Asignación

```
# Good  
x <- 5  
  
# Bad  
x = 5
```

Scripts

Using = instead of <- for assignment



Scripts

- Comentarios

```
# Load data -----
```

```
# Run model -----
```

```
# Plot data -----
```

Scripts

- Ejemplo

```
# Pedro Coyotl, 29/02/2020 ¡Este código es un ejemplo!

# Limpieza del espacio de trabajo
rm(list =ls()) # remueve variables del workspace
graphics.off() # cierra todas las ventanas de graficos

# Carga de bibliotecas o paquetes (ya instalados)
library(dplyr)
library(magrittr)
library(tidyr)

# Carga de datos al espacio de trabajo
datos <- read.table("mis_datos.csv", header =T , sep =",")

# Análisis
mean(datos$pH)
sd(datos$pH)
plot(datos$pH, datos$dia)
datos %>% group_by(mes) %>% summarise(pH_p = mean(pH), pH_de = sd(pH))
```

Licencia CC BY



Estadística y Manejo de Datos con R (EMDR) por Marcos F. Rosetti S. y Luis Pacheco-Cobos se distribuye bajo una [Licencia Creative Commons Atribución 4.0 Internacional](https://creativecommons.org/licenses/by/4.0/).