

Notas de Programación en R

Rubén Fernández Casal (rubenfcasal@gmail.com)

Edición: Marzo de 2023. Impresión: 2023-04-01

Índice general

Prólogo	5
1 Introducción	7
1.1 Organización	8
I El entorno estadístico R	13
2 El lenguaje R	15
2.1 Paquetes	15
2.2 Funciones	17
2.3 Programación orientada a objetos (funciones genéricas)	18
2.4 Desarrollo de funciones y paquetes	20
3 Generación de informes	23
3.1 RMarkdown	23
3.2 Sintaxis de Markdown	24
3.3 Inclusión de código R	24
3.4 Tablas	26
3.5 Cabecera YAML	27
3.6 Extracción del código R	29
3.7 Spin	29
3.8 Extensiones RMarkdown de pandoc	30
II Tidyverse	31
4 El ecosistema tidyverse	33
4.1 Operador <i>pipe</i> (redirección)	34
5 Manipulación de datos con dplyr y tidyr	37
5.1 El paquete dplyr	37
5.2 Operaciones con variables (columnas)	38
5.3 Operaciones con casos (filas)	39
5.4 Herramientas tidyr	41
5.5 Operaciones con tablas de datos	41
5.6 Bases de datos con dplyr	41
Referencias	47
Bibliografía por temas	47
Enlaces	49
A Pandoc	51
A.1 Conversión de documentos con Pandoc	51
A.2 Pandoc y RMarkdown	52

Prólogo

Este es un libro, *en proceso de elaboración*, con notas personales sobre programación en R para el análisis de datos, en el que incluyen referencias a información y recursos adicionales (se asumen unos conocimientos básicos de R). El contenido está sesgado por la experiencia personal (es mi forma de programar en R) pero puede resultar útil para otras personas. Cualquier sugerencia de mejora o comentario será bien recibido.

Este libro ha sido escrito en R-Markdown empleando el paquete `bookdown` y está disponible en el repositorio Github: `rubenfcasal/book_notasr`. Se puede acceder a la versión en línea a través del siguiente enlace:

https://rubenfcasal.github.io/book_notasr.

donde puede descargarse en formato pdf.

Para seguir los ejemplos mostrados en el libro (en la carpeta ejemplos se incluyen algunos ejemplos adicionales) se recomienda tener instalados los siguientes paquetes (realmente no se emplean todos): `Rcmdr`, `caret`, `tidymodels`, `tidyverse`, `openxlsx`, `DT`, `rmarkdown`, `knitr`, `remotes`, `devtools`. Por ejemplo mediante los siguientes comandos:

```
pkgs <- c("Rcmdr", "caret", "tidymodels", "tidyverse", "openxlsx", "DT",
          "rmarkdown", "knitr", "remotes", "devtools")
install.packages(setdiff(pkgs, installed.packages()[,"Package"]), dependencies = TRUE)
```

(puede que haya que seleccionar el repositorio de descarga, e.g. *Oficina de software libre (CIXUG)*).

El código anterior no reinstala los paquetes ya instalados, por lo que podrían aparecer problemas debidos a incompatibilidades entre versiones (aunque no suele ocurrir, salvo que nuestra instalación de R esté muy desactualizada). Si es el caso, en lugar de la última línea se puede ejecutar:

```
install.packages(pkgs, dependencies = TRUE) # Instala todos...
```

Para generar el libro (compilar) serán necesarios paquetes adicionales, para lo que se recomendaría consultar el libro de “Escritura de libros con bookdown” en castellano.

Este obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial-SinObraDerivada 4.0 Internacional (esperamos poder liberarlo bajo una licencia menos restrictiva más adelante...).



Capítulo 1

Introducción

Como aparece en el prólogo, estos apuntes contienen recomendaciones y notas personales sobre programación en R para el análisis de datos, en el que incluyen referencias a información y recursos adicionales que considero de interés. Se tratará de mostrar una forma de llevar a cabo las distintas tareas que pueden surgir en el análisis de datos empleando R, esto no quiere decir que sea la mejor forma de hacerlo o la más cómoda (que dependerá de cada persona).

En estas notas *se asumen unos conocimientos básicos de R*, un lenguaje de programación (interpretado) y un entorno estadístico desarrollado específicamente para el análisis estadístico. Puede ser una herramienta de gran utilidad a lo largo de todo el proceso de obtención de información a partir de datos (ver Figura 1.1).

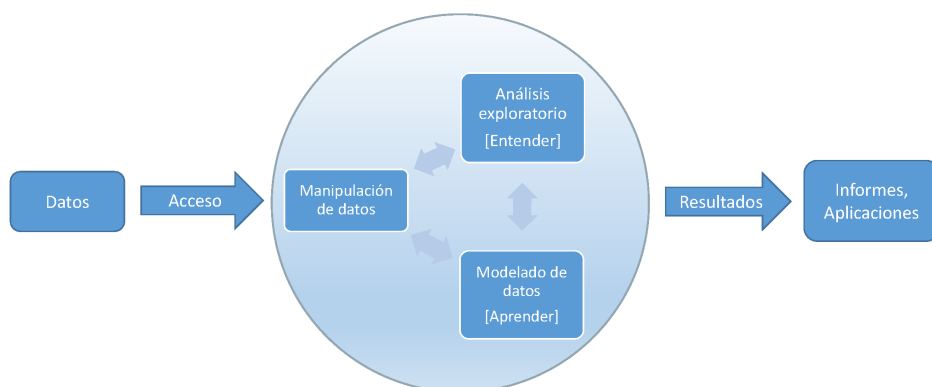


Figura 1.1: Etapas del proceso

Para una introducción a la programación en R se puede consultar el libro:

Fernández-Casal R., Roca-Pardiñas J., Costa J. y Oviedo-de la Fuente M. (2022). *Introducción al Análisis de Datos con R* (github).

Adicionalmente, en este post se incluyen enlaces a recursos adicionales, incluyendo libros y cursos, que pueden ser útiles para el aprendizaje de R.

El primer paso es la instalación de R, para ello se recomienda seguir los pasos en este post.

Para el desarrollo de código e informes la recomendación es emplear *RStudio Desktop*, que se puede instalar y configurar siguiendo las indicaciones en este post. También puede resultar de interés consultar:

- RStudio IDE Cheat Sheet (menú de RStudio *Help* > *Cheat Sheets* > *RStudio IDE Cheat Sheet*).
- Using the RStudio IDE.

Sin embargo, en ciertos casos puede ser recomendable ejecutar el código R directamente desde una ventana de comandos (por ejemplo para ejecutar varios programas de forma simultánea en distintos directorios de trabajo o si los requerimientos computacionales son grandes). En mi caso, cuando trabajo en Windows, acostumbro a emplear el explorador para situarme en el directorio donde quiero ejecutar código y abrir una ventana de comandos, escribiendo `cmd` en el cuadro superior donde se muestra la ruta. Posteriormente, como añadí en directorio de instalación de R al *path* (ver post), ejecuto¹ R y finalmente un comando de la forma:

```
source("mi_script.R", echo = TRUE, encoding = "UTF-8") # UTF-8 importante en R < 4.2
```

1.1 Organización

Para la organización de archivos (datos, código, informes...) lo recomendable es emplear un directorio con la estructura adecuada.

Dependiendo del objetivo puede interesar emplear un proyecto de RStudio (menú *File > New project...*). En mi caso empleo esta opción para paquetes, libros en bookdown, webs con blogdown y aplicaciones shiny. En otros casos empleo una carpeta que puede tener subdirectorios (si el proyecto es más grande) para distintos tipos de archivos o para distintas tareas (con el objetivo de facilitar la búsqueda). Por ejemplo: *datos*, *informes*, *resultados_2023...*

Mi recomendación es emplear nombres de archivos y carpetas en minúscula (o con la primera letra en mayúsculas), sin espacios (por ejemplo empleando `_` para separar palabras o iniciales) y sin caracteres especiales (ASCII, sin acentos...). Los nombres deberían ser lo más descriptivos posibles (en el sentido de evitar confusión). Pueden incluirse descripciones más completas en el código, en ficheros de texto (e.g. *Descripcion_archivos.txt*), o incluso en hojas de cálculo. Yo además acostumbro a incluir archivos del tipo *Notas.txt* (con recordatorios, decisiones...) o *Pendiente.txt* (con próximos pasos, mejoras o verificaciones pendientes...).

Además, nos puede interesar establecer opciones de R específicas para el proyecto (por ejemplo opciones de configuración de memoria, de paquetes o variables de entorno, incluyendo claves privadas), de forma que se establezcan automáticamente al iniciar R o RStudio. Para más detalles ver la ayuda de `?Startup`, el apéndice *Invoking R* o el post de RStudio *Managing R with .Rprofile*, *.Renv*, *Rprofile.site*, *Renv.site*, *rsession.conf*, and *repos.conf*

Para desarrollar código y proyectos de forma colaborativa, la recomendación es emplear un sistema de control de versiones. Se puede configurar RStudio para emplear Git (ver el libro *Happy Git and GitHub for the user* o la sección *Git and GitHub*), sin embargo yo prefiero emplear *GitHub Desktop*.

1.1.1 Código e informes

Mi recomendación a la hora de escribir código es seguir un **proceso iterativo**. Se comienza realizando pruebas y al finalizar cada etapa se trata de reorganizar el código (adaptándolo al estilo de programación elegido, lo que incluiría añadir comentarios y secciones) de forma que sea más cómodo continuar trabajando en siguientes etapas (y si es posible que resulte más fácil de adaptar para otros casos).

En el caso de informes el proceso sería similar, empleando como punto de partida un fichero de código en formato spin (ver Sección 3.7), en el que el texto RMarkdown se incluye como un comentario de código empleando `#'`. Por ejemplo:

```
#' # Sección
#'
#' ## Subsección
#'
#' Texto rmarkdown...
```

¹También se puede ejecutar un script de R de forma no interactiva ejecutando en el intérprete de comandos del sistema operativo: `R CMD BATCH [opciones] mi_script.R [fichero_salida]` (cambiando R por la ruta completa, e.g. `"C:\Program Files\R\R-4.2.1\bin\R.exe"`, si no se añadió al path. También se puede incluir en un fichero *.bat*, para poder ejecutarlo repetidas veces con mayor facilidad). Ver Appendix B *Invoking R* de *Introduction to R* para información sobre las distintas opciones.

En primer lugar me preocupo de escribir un código funcional y, además de ir añadiendo comentarios de la forma habitual, voy añadiendo secciones y texto rmarkdown en formato spin. Finalmente, cuando tengo una primera versión del código (que puedo ir previsualizando; en RStudio basta con pulsar² *Ctrl + Shift + K*, el icono correspondiente en la barra superior, o seleccionar *File > Compile Report...*), lo transformo a formato *.Rmd* con un comando de la forma:

```
knitr::spin("Informe.R", knit = FALSE)
```

donde termino de redactar (`knitr::purl("Informe.Rmd", documentation = 2)` genera un nuevo fichero *Informe.R* donde resulta más cómodo modificar o desarrollar código).

Se recomienda **elegir un estilo que sea consistente y seguirlo por completo** en todo el proyecto. Lo principal sería el operador de asignación, el espaciado y el estilo de nombres (de objetos, variables o ficheros):

- `estilo.clasico`: es el estilo del paquete base de R. Muchos programadores no lo recomiendan (principalmente porque este separador no se admite en otros lenguajes y porque puede dar lugar a confusión con métodos S3, ver Sección 2.3).
- `estilo_serpiente` (o `Estilo_serpiente`): es el estilo de la colección de paquetes `tidyverse`.
- `EstiloCamello` (o `estiloCamello`): es el estilo (casi obligatorio) para las clases R6 (ver Sección 2.3). El paquete `shiny` emplea la variante que comienza por minúsculas.

Recomiendo emplear `<-` como operador de asignación y escribir todos los **nombres en minúsculas**. Yo tengo tendencia a emplear el `estilo.clasico`, sobre todo si el código no depende de paquetes `tidyverse` (en ese caso suelo emplear `estilo_serpiente`). También influye el estilo de nombres empleado por la fuente de datos o el requerido en los resultados.

El estilo también debe especificar el sangrado, el espaciado, etc. Por ejemplo:

- Tidyverse style guide
- Google's R Style Guide

Para facilitar la legibilidad **es muy recomendable incluir un espacio** entre los elementos del comando. En RStudio se puede seleccionar un trozo de (una línea de) código y pulsar *Ctrl + Shift + A* para formatearlo. También podemos emplear el paquete `styler` para formatear el código. Por ejemplo, en RStudio podemos emplear *Addins > Styler > Style active file*.

Además se recomienda **crear secciones y documentar el código adecuadamente**. En RStudio se puede crear una sección pulsando *Ctrl + Shift + R* o añadiendo al menos 4 guiones (`-`, también `=` o `#`) después de un comentario. Por ejemplo:

```
# Sección ----
## Subsección ----
```

El orden de las secciones y subsecciones es importante. Al principio del código debería ir:

1. Los parámetros o variables globales.
2. La carga de paquetes (únicamente los mínimos requeridos).
3. La carga de código externo.
4. La carga de archivos de datos (o al principio de la sección donde se emplean, si son datos auxiliares).

No se recomienda emplear rutas absolutas en el código, del tipo:

```
setwd("C:/Documentos/Proyectos/Proyecto_X")
load("C:/Documentos/Proyectos/Proyecto_X/datos_x.RData")
source("C:/Documentos/Proyectos/R/Herramientas.R")
```

Como punto de partida el directorio de trabajo debería ser la carpeta del proyecto. Esto ya ocurre por defecto si empleamos proyectos de RStudio o si iniciamos RStudio abriendo un archivo de código

²Para mostrar las combinaciones de teclas en RStudio podemos emplear el menú *Tools > Keyboard Shortcuts Help*.

en esta carpeta. En general, la recomendación es asumir que el directorio de trabajo es aquel en el que se encuentra el archivo de código (lo que también ocurre por defecto al compilar un documento RMarkdown). Si no es el caso se puede emplear el menú *Sesion > Set Working Directory > To Source File Location*.

Para establecer la ruta a archivos o directorios **se recomienda emplear rutas relativas** (usando `../` para acceder a la carpeta anterior; `./` sería el actual directorio de trabajo). Por ejemplo:

```
load("datos/datos_x.RData")
source("../R/Herramientas.R")
fecha_txt <- as.character(Sys.Date() - 1, format = "%m_%d") # Por ejemplo...
rmarkdown::render("informe.Rmd", params = list(fecha_txt = fecha_txt),
                 output_file = paste0('informes/informe_', fecha_txt, '.html'),
                 envir = new.env(), encoding = "UTF-8")
```

La mejor forma de organizar funciones es desarrollar un paquete, como se comenta más adelante en la Sección 2.4.

1.1.2 Datos

La recomendación es emplear ficheros de datos con el formato por defecto de R (datos binarios comprimidos), con extensión *.RData*. Hay que tener en cuenta que lo esperable es que el archivo contenga un conjunto de datos con el mismo nombre, aunque podría no ser el caso e incluso contener varios objetos.

Uno de los problemas con los ficheros *.RData* es que, al cargarlos con `load()` de la forma habitual, se añaden al entorno de trabajo los objetos que contienen con los nombres con que se almacenaron (y si ya existe alguno con ese nombre lo sobrescribe) Para almacenar un único objeto de forma que se pueda cargar posteriormente especificando el nombre, se pueden emplear las funciones `saveRDS()` y `readRDS()`.

Sin embargo, lo habitual es que inicialmente los datos procedan de una fuente externa. Se pueden importar datos externos en casi cualquier formato a R (aunque puede requerir instalar paquetes adicionales). Mi recomendación es separar los análisis de la importación de los datos. Crear un fichero de código específicamente para importar los datos³, hacer el (pre)procesado y guardarlos en formato *.RData*. Yo habitualmente empleo el mismo nombre para el archivo de código y el archivo de datos que se genera (e.g. *datos.R* contiene el código necesario para generar *datos.RData*; no suelo renombrar el fichero fuente de datos externo, aunque se aleje mucho del estilo elegido). Asociado a un mismo conjunto de datos puede haber distintos archivos de código para realizar distintos análisis (el nombre de esos archivos debería dar una pista del análisis que realizan).

En muchas ocasiones, para modificar los nombres de las variables o los niveles de un factor, suelo recurrir a la función `dput()` para escribirlos en modo texto (e.g. `dput(tolower(names(datos)))` o `dput(levels(datos$factor))`) y posteriormente modificarlos a mano.

Yo recomiendo añadir un atributo `variable.labels` que contenga un vector de etiquetas de las variables y empleando como nombres de las componentes las propias variables:

```
data(cars)
# dput(names(cars))
var.lab <- c(speed = "Speed (mph)", dist = "Stopping distance (ft)")
attr(cars, "variable.labels") <- var.lab
str(cars)

## 'data.frame':   50 obs. of  2 variables:
## $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
## $ dist : num  2 10 4 22 16 10 18 26 34 17 ...
## - attr(*, "variable.labels")= Named chr [1:2] "Speed (mph)" "Stopping distance (ft)"
## ..- attr(*, "names")= chr [1:2] "speed" "dist"
```

³Con algunos tipos de datos, se puede emplear los submenús de RStudio *File > Import Dataset* para seleccionar los ajustes, previsualizando el resultado, y generar el código para importarlos.

```
# View(cars)
# with(cars, plot(speed, dist, xlab = var.lab["speed"],
#              ylab = var.lab["dist"]))
```

Para leer ficheros de Excel acostumbro a utilizar los paquetes `openxlsx` (solo para archivos con extensión `.xlsx`) o `readxl` (colección `tidyverse`; Sección 4). En estos casos además se puede añadir una nueva hoja de cálculo con los nombres de las variables junto con su etiqueta, que se puede cargar y emplear durante el preprocesado. Adicionalmente esta tabla puede incluir una columna con los nuevos nombres (yo recomiendo no modificar los antiguos en este fichero), otra con un filtro para seleccionar variables (o el orden después del procesado) e incluso una columna con anotaciones o observaciones. Ver `top500.R` en ejemplos.

Parte I

El entorno estadístico R

Capítulo 2

El lenguaje R

Cualquier análisis de R requiere programación, aunque normalmente se puede llevar a cabo sin conocimientos profundos del lenguaje (*useR*). Sin embargo, para desarrollar nuevas herramientas de forma efectiva (*programeR*) es necesario tener una idea del funcionamiento interno de R. La referencia recomendada para usuarios de R que deseen mejorar sus conocimientos de programación y comprensión del lenguaje es:

Wickham, Hadley (2019). *Advanced R*, 2ª edición, Chapman & Hall, 1ª edición.

También puede ser de utilidad el manual R Language Definition para consultas adicionales¹.

2.1 Paquetes

Al instalar R se instalan los denominados **paquetes base** y (por defecto) los **paquetes recomendados** por los desarrolladores de R (el *R Core Team*). Podemos acceder a la lista de paquetes instalados:

```
pkgs <- installed.packages()
names(which(pkgs[, "Priority"] == "base"))

## [1] "base"      "compiler" "datasets"  "graphics"  "grDevices" "grid"
## [7] "methods"  "parallel"  "splines"   "stats"     "stats4"    "tcltk"
## [13] "tools"    "utils"
```

```
names(which(pkgs[, "Priority"] == "recommended"))
```

```
## [1] "boot"      "class"     "cluster"   "codetools" "foreign"
## [6] "KernSmooth" "lattice"   "MASS"      "Matrix"    "mgcv"
## [11] "nlme"     "nnet"      "rpart"     "spatial"   "survival"
```

Para instalar paquetes adicionales se puede emplear `install.packages()` (actualmente, 2023-04-01, están disponibles 19217 en CRAN, incluso para interactuar con ChatGPT como `gptstudio`). Por ejemplo:

```
pkgs <- c("Rcmdr", "caret", "tidymodels", "tidyverse", "remotes", "devtools",
         "sf", "gstat", "geoR", "quadprog", "DEoptim", "spam", "openxlsx",
         "bookdown", "blogdown", "pkgdown")
install.packages(setdiff(pkgs, installed.packages()[, "Package"]), dependencies = TRUE)
```

En Windows (y en MacOS) esta función instala por defecto paquetes compilados (`type = "binary"`, que dependen del sistema operativo y de la versión R) disponibles en CRAN. Aunque podría instalar paquetes disponibles en otros repositorios. Por ejemplo:

¹Los manuales oficiales también están disponibles en formato bookdown en este post.

```
url <- "https://github.com/rubenfcasal/simres/releases/download/v0.1/simres_0.1.3.zip"
install.packages(url, repos = NULL)
```

También se pueden instalar paquetes directamente a partir del código fuente con `type = "source"` (por defecto en Linux), pero en ciertos casos es necesario tener instaladas herramientas adicionales (por ejemplo Rtools en Windows si el paquete contiene código en C, C++ o Fortran). Esto permitiría incluso instalar paquetes retirados de CRAN (e.g. actualmente `kedd`), ya que siempre se mantiene el código (en un archivo comprimido de la forma `paquete_x.y.z.tar.gz`).

Si se quieren instalar paquetes de repositorios distintos de CRAN (GitHub, GitLab, Bitbucket...), puede ser recomendable instalar `remotes`. Por ejemplo:

```
remotes::install_github("rubenfcasal/simres", INSTALL_opts = "--with-keep.source")
```

Además puede ser de utilidad mantener los comentarios originales del paquete para entender mejor el código (por ejemplo si se quiere modificar).

Otras funciones que pueden ser de interés son: `remove.packages()`, `update.packages()` y `available.packages()`.

Al iniciar el programa R se cargan por defecto en memoria los principales paquetes base, añadiéndolos a la ruta de búsqueda (a continuación del entorno de trabajo `.GlobalEnv` y siempre terminando con en el paquete `base`, el primero que se carga):

```
search()

## [1] ".GlobalEnv"      "package:dbplyr"    "package:forcats"
## [4] "package:stringr" "package:dplyr"     "package:purrr"
## [7] "package:readr"   "package:tidyr"     "package:tibble"
## [10] "package:ggplot2" "package:tidyverse" "package:magrittr"
## [13] "package:stats"   "package:graphics" "package:grDevices"
## [16] "package:utils"   "package:datasets" "package:methods"
## [19] "Autoloads"       "package:base"
```

Concretamente se añade a la ruta de búsqueda un entorno que contiene el conjunto de objetos exportables del paquete, definido en el denominado *namespace* del paquete. Esta ruta determina los objetos visibles en el entorno global y el orden en se buscan (para más detalles ver 7.2 Environment basics y 7.4 Special environments de Advanced R).

Podemos cargar paquetes adicionales (previamente instalados) con `library()` o `require()`, por ejemplo:

```
if (!require(knitr)) {
  install.packages("knitr")
  library(knitr)
}
spin("01-Introduccion.R", knit = FALSE)
```

Aunque **no se recomienda que el código instale automáticamente paquetes** (en general que haga cambios en la configuración del equipo en el que se ejecuta).

Al cargar un paquete se añade por defecto en la segunda posición de la ruta de búsqueda (justo después del entorno global, desplazando al resto). También se podrían añadir otros objetos, por ejemplo `data.frames`, con la función `attach()` pero **no se recomienda** (se puede utilizar `with()` como alternativa).

Hay que tener cuidado con las versiones instaladas de los paquetes:

```
packageVersion("dplyr")
```

```
## [1] '1.0.10'
```


y con sus dependencias (los paquetes tienen su propia ruta de búsqueda, determinada por el *namespace* del paquete). Al actualizar o instalar nuevos paquetes pueden aparecer problemas al ejecutar código antiguo (a veces al trabajar en nuevos proyectos acabamos haciendo que los antiguos dejen de funcionar).

Se puede instalar versiones específicas de un paquete con `remotes::install_version()`:

```
remotes::install_version("dplyr", version = "1.11") # repos = "https://ftp.ciug.es/CRAN"
```

Para asegurarse que el código de un proyecto se pueda ejecutar a lo largo del tiempo se puede emplear el paquete `renv` (se puede configurar automáticamente al crear un proyecto de RStudio). Este paquete permite registrar las versiones exactas de los paquetes de los que depende un proyecto y volver a instalarlas (incluso en otro equipo) si es necesario. Para más detalles ver la viñeta *Introduction to renv*.

Sin embargo de esta forma aún dependemos del sistema operativo que deberíamos configurar adecuadamente. La recomendación para que un proyecto en R (por ejemplo una aplicación *shiny*) se pueda ejecutar en cualquier equipo, es emplear un contenedor *docker*. Para más detalles ver *Docker overview* y *The Rocker Project*. Ver ejemplos/*covid19/prediccion_cooperativa*.

2.2 Funciones

“Everything that happens in R is the result of a function call”.

— John M. Chambers

Como es bien conocido, en R se pueden asignar los argumentos de una función por posición o por nombre (del correspondiente parámetro en la definición de la función, denominado *argumento formal en R*). En general, la recomendación es asignar los argumentos por nombre:

```
funcion(parametro1 = argumento1, parametro2 = argumento2, ...)
```

De esta forma no importa el orden de los parámetros y, por ejemplo, evitaremos problemas si en el futuro hay cambios en la definición de la función. Los parámetros pueden tener valores por defecto y solo sería necesario especificarlos para asignarles un valor distinto.

Podemos llamar a una función de un paquete sin necesidad de cargarlo (añadirlo a la ruta de búsqueda) empleando `paquete::funcion`. Esto es especialmente recomendable al desarrollar nuevas funciones (es un requisito para subir paquetes a CRAN), ya que de esta forma se evitan conflictos entre funciones con el mismo nombre en paquetes distintos. Por ejemplo:

```
if (!requireNamespace("knitr")) stop("'knitr' package required")
knitr::spin("01-Introduccion.R", knit = FALSE)
```

Hay que tener en cuenta que R emplea *Lazy evaluation*, los argumentos no se evalúan hasta que se necesitan (lo cual puede producir mensajes de error inesperados, pero también permite añadir funcionalidades adicionales empleando la denominada *evaluación no estándar* o *metaprogramación*).

R es un lenguaje interpretado y podemos evaluar expresiones empleando código. Por ejemplo, podemos reproducir el proceso de introducir un comando en la consola con las funciones `eval()` y `parse()` (aunque esta forma de proceder no es la más eficiente):

```
eval(parse(text = "1:10"))
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

```
distr <- "norm" # "unif", "exp", "t"
ddistr <- eval(parse(text = paste0("d", distr)))
# str(ddistr)
# curve(ddistr(x, 0, 0.5), -3, 3)
```

Para llamar a una función especificando los parámetros de forma dinámica (empleando una lista) podemos emplear `do.call()`. Por ejemplo:

```
# Listar ficheros csv
files.csv <- dir(path = "datos", pattern = "*.csv", full.names = TRUE)
# Leer datos a una lista
# (suponemos variante local con ; para separar valores)
data.list <- lapply(files.csv, read.csv2)
# Combinar
datos <- do.call('rbind', data.list)
```

R dispone además de otras herramientas que permiten la programación dinámica. Por ejemplo `reformulate()` permite construir formulas para ajuste de modelos o análisis descriptivos.

Hay que tener en cuenta que las funciones tienen su propio entorno y su propia ruta de búsqueda, determinada por el entorno donde se crearon (el *namespace* en el caso de las funciones de un paquete). Esto es lo que se conoce como Lexical scoping.

```
x <- 1
addx <- function(y) {
  x + y
}
addx(10)

## [1] 11

addx10 <- function() {
  x <- 10 # x <- 10 # assign("x", 10, envir = .GlobalEnv)
  addx(x)
}
addx10()

## [1] 11
x

## [1] 1
```

2.3 Programación orientada a objetos (funciones genéricas)

“Everything that exists in R is an object”.

— John M. Chambers

R implementa programación orientada a objetos (OOP). Por ejemplo, es bien conocido que algunas funciones (entre ellas `print()`, `plot()` o `summary()`) se comportan de manera diferente dependiendo de la clase (el tipo de objeto) de sus argumentos, son las denominadas *funciones genéricas*.

Realmente R dispone de varios sistemas de OOP, entre ellos podríamos destacar (ver capítulos en Object-oriented programming de Advanced R):

- S3: Es un sistema muy simple, las clases no tienen una definición formal (no se verifica su consistencia). Es el empleado en el paquete `base` de R y en la mayoría de paquetes que usan OOP. Descrito inicialmente en:

Becker R.A., Chambers J.M. y Wilks A.R. (1988), *The New S Language: A Programming Environment for Data Analysis and Graphics* (A.K.A. the *Blue Book*). Chapman & Hall.

Chambers J.M. y Hastie T.J. eds. (1992), *Statistical Models in S* (A.K.A. the *White Book*). Chapman & Hall.

- S4 (**no lo recomiendo**): Es similar a S3 pero mucho más formal. Está implementado en el paquete `methods` (uno de los *paquetes base*) de R. Se emplea por ejemplo en los paquetes `sp` y `distr`. Descrito inicialmente en:

Chambers J.M. (1998), *Programming with Data* (A.K.A. the *Green Book*). Springer.

- R6: Es un sistema OOP encapsulado similar al de otros lenguajes de programación. Está implementado en el paquete R6 (no se instala por defecto).

Yo en principio **recomendaría usar el sistema S3**, aunque es bastante rudimentario y puede resultar inicialmente confuso a programadores con experiencia en otros lenguajes. En cualquier caso es muy recomendable conocer su funcionamiento. Este sistema esta basado en funciones genéricas. La clase es un atributo de los objetos (*encapsulación*), una cadena de texto o un vector de cadenas (*herencia*), al que se puede acceder con la función `class()`. A partir de la clase del argumento, la función genérica determina el método (función especializada) al que debe llamar (*polimorfismo*). En S3 el despacho de métodos (*method dispatch*) es muy simple, si la función genérica es `generica()` y la clase del primer argumento es `"class"`, se llama a la función (método) `generica.class()` si existe. Si la clase del objeto es heredada (un vector de cadenas), se van buscando los métodos por orden de parentesco y si no se encuentra ninguno, se llama al método por defecto `generica.default()` (se llama a la primera función de `paste0("generica.", c(class(x), "default"))` que se encuentre en la ruta de búsqueda; podríamos reemplazarla...).

La función genérica suele ser muy sencilla, básicamente incluye una llamada a `UseMethod("generica")`. Por ejemplo:

```
plot

## function (x, y, ...)
## UseMethod("plot")
## <bytecode: 0x000000001d8a0168>
## <environment: namespace:base>
```

Podemos obtener los métodos asociados a una función genérica con `methods(generica)`. Por ejemplo:

```
methods(plot)

## [1] plot,ANY-method      plot,color-method    plot.acf*
## [4] plot.data.frame*     plot.decomposed.ts*  plot.default
## [7] plot.dendrogram*     plot.density*        plot.ecdf
## [10] plot.factor*         plot.formula*        plot.function
## [13] plot.ggplot*         plot.gtable*         plot.hcl_palettes*
## [16] plot.hclust*         plot.histogram*      plot.HoltWinters*
## [19] plot.isoreg*         plot.lm*              plot.medpolish*
## [22] plot.mlm*            plot.ppr*             plot.prcomp*
## [25] plot.princomp*       plot.profile.nls*    plot.R6*
## [28] plot.raster*         plot.spec*           plot.stepfun
## [31] plot.stl*            plot.table*          plot.trans*
## [34] plot.ts              plot.tskernel*       plot.TukeyHSD*
## see '?methods' for accessing help and source code
```

Podemos acceder a la ayuda del correspondiente método de la forma habitual (e.g. `?plot.lm`), pero puede que algunos métodos no sean objetos definidos como exportables en el *namespace* del paquete que los implementa (los marcados con un `*`) y por tanto no son en principio accesibles para el usuario. Siempre podemos acceder a ellos empleando `paquete:::metodo` o `getAnywhere(metodo)` (e.g. `stats:::plot.lm` o `getAnywhere(plot.lm)`).

Para listar los métodos disponibles para una clase, podemos emplear el parámetro `class`. Por ejemplo:

```
methods(class = "lm")

## [1] add1          alias          anova          case.names     coerce
## [6] confint       cooks.distance deviance       dfbeta         dfbetas
## [11] drop1         dummy.coef     effects        extractAIC     family
## [16] formula      fortify        hatvalues      influence      initialize
## [21] kappa        labels         logLik         model.frame    model.matrix
## [26] nobs         plot           predict        print          proj
## [31] qr           residuals      rstandard     rstudent       show
```

```
## [36] simulate      slotsFromS3    summary      variable.names vcov
## see '?methods' for accessing help and source code
```

Para una programación orientada a objetos más formal la recomendación es emplear el sistema R6.

2.4 Desarrollo de funciones y paquetes

Antes de ponerse a programar, sobre todo si puede terminar siendo un código complejo, la recomendación es hacer una búsqueda por si resulta que ya está implementado (o hay algo que podemos tomar como base; es lo bueno de GNU!): en la descripción de los paquetes en CRAN, en los buscadores especializados (rdr.io, RDocumentation o RSeek), en foros de programación (StackOverflow, StackOverflow.es, Cross Validated), en listas de correo (r-project.org, r-help-es) o directamente en Google (añadiendo “r-project” o similar en la búsqueda).

El primer paso es escribir el código como si fuese un programa, asignando valores de prueba a los parámetros, y cuando nos aseguramos de que funciona, reescribirlo como función (yo suelo mantener unos valores de prueba como comentarios por si quiero ejecutar paso a paso el cuerpo de la función).

Al finalizar, la recomendación es **documentar la función**, preferiblemente empleando el formato roxygen2 (ver el menú de RStudio *Help > Roxygen Quick Reference*). Por ejemplo:

```
# read_excel_list(path, pattern, ...)
# .....
#' Lee los ficheros xls yxlsx de un directorio
#'
#' @param path Ruta al directorio con los ficheros excel
#' (por defecto el directorio de trabajo).
#' @param pattern Expresión regular empleada en la selección de ficheros
#' (ver `list.files()`).
#' @param ... Parámetros adicionales de `readxl::read_excel()`.
#' @return Una lista cuyas componentes son las correspondientes tablas de datos
#' (`tibble`) y con nombres los nombres de los archivos sin extensión.
#' @examples \dontrun{
#' data_list <- read_excel_list("datos") # "./datos"
#' data_all <- dplyr::bind_rows(data_list)
#' }
# .....
# Pruebas:
# readxl::readxl_example("geometry.xls")
# path = "C:/Program Files/R/R-4.2.2/library/readxl/extdata"
# pattern = "\\.(xls|xlsx)$"
# Pendiente:
# - Controlar posible error al leer
# .....
read_excel_list <- function(path = ".", pattern = "\\.(xls|xlsx)$", ...) {
  if (!requireNamespace(readxl)) stop("'readxl' package required")
  files <- dir(path, pattern = pattern, full.names = TRUE) # ?list.files
  data_list <- vector(length(files), mode = 'list')
  for (i in seq_along(files))
    data_list[[i]] <- readxl::read_excel(files[i], ...)
  data_names <- sub('\\.xlsx$', '', basename(files))
  names(data_list) <- data_names
  data_list
}
```

Como ya se comentó, en ocasiones se emplea como punto de partida una función ya implementada en algún paquete de R. En RStudio la forma más sencilla de obtener el código de la función es emplear `View(funcion)` (si la función es visible, en caso contrario `View(paquete:::funcion)`).

Si la función llama a funciones internas (que no se exportan en el namespace) del paquete que la implementa, podríamos emplear también los tres dobles puntos para llamarlas, pero la recomendación sería descargar el código del paquete (si está en CRAN, un fichero comprimido de la forma `paquete_x.y.z.tar.gz` que se puede descargar en la sección *Downloads* de la web del paquete <https://CRAN.R-project.org/package=paquete>).

La mejor forma de organizar funciones es crear un paquete. Para ello se recomienda seguir:

Wickham, Hadley (2015). *R packages: organize, test, document, and share your code* (actualmente 2ª edición en desarrollo con H. Bryan), O'Reilly, 1ª edición.

También puede ser de utilidad el manual *Writing R Extensions* para información adicional.

Capítulo 3

Generación de informes

Este documento se ha generado empleando RMarkdown, una extensión de Markdown que permite incorporar código y resultados de R. RMarkdown es recomendable para difundir análisis realizados con R en formato HTML, PDF y DOCX (Word), entre otros.

3.1 RMarkdown

El paquete `rmarkdown` permite combinar Markdown con R para la generación de documentos. Markdown se diseñó inicialmente para la creación de páginas web a partir de documentos de texto de forma muy sencilla y rápida (tiene unas reglas sintácticas muy simples). Es lo que se conoce como un lenguaje de marcado ligero, tiene unas reglas sintácticas muy simples y se busca principalmente la facilidad de lectura. Posteriormente se fueron añadiendo funcionalidades, por ejemplo para incluir opciones de publicación en muchos otros formatos. La implementación original de Markdown es de John Gruber, pero actualmente están disponibles múltiples dialectos (sobre todo para publicar en gestores de contenido). RMarkdown utiliza las extensiones de la sintaxis proporcionada por *Pandoc* (ver Apéndice A), y adicionalmente permite la inclusión de código R.

Al renderizar un fichero RMarkdown se generará un documento que incluye el código R y los resultados incrustados en el documento¹. En RStudio basta con hacer clic en el botón *Knit*. En R se puede emplear la función `render` del paquete `rmarkdown` (por ejemplo `render("Informe.Rmd")`). También se puede abrir directamente el informe generado:

```
library(rmarkdown)
browseURL(url = render("Informe.Rmd"))
```

A continuación se darán algunos detalles sobre RMarkdown (y las extensiones Markdown de Pandoc que admite: notas al pie de página, tablas, citas, ecuaciones LaTeX...). Para más información (incluyendo introducciones a Markdown y RMarkdown), se recomienda consultar alguna de las numerosas fuentes disponibles, comenzando por la web oficial <http://rmarkdown.rstudio.com/>.

También se dispone de información en la ayuda de RStudio:

- *Help > Markdown Quick Reference*
- *Help > Cheatsheets > R Markdown Cheat Sheet*
- *Help > Cheatsheets > R Markdown Reference Guide*

Otras fuentes permiten obtener documentación más detallada, como por ejemplo:


- Web original del paquete `knitr` (Xie, 2021): `knitr`; Elegant, flexible, and fast dynamic report generation with R.

¹Se llama al paquete `knitr` para “tejer” el código de R y los resultados en un fichero Markdown, que posteriormente es procesado con *pandoc*

- Xie et al. (2018) : R Markdown: The Definitive Guide.
- Extensiones RMarkdown de Pandoc: Pandoc's Markdown.

3.2 Sintaxis de Markdown

Como ya se comentó la sintaxis de Markdown es muy sencilla. El texto no marcado se renderiza como texto normal aunque es necesario dejar una línea en blanco para insertar un salto de línea. Las principales reglas de Markdown se resumen en la siguiente tabla:

Escribir	o alternativamente	para obtener ...
# Título 1	Título 1 =====	Los distintos niveles de encabezados
## Título 2	Título 2 -----	
### Título 3 <i>*Cursiva*</i> **Negrita**	### Título 3 _Cursiva_ __Negrita__	<i>Cursiva</i> Negrita
[Enlace] (http://url.gz)	[Enlace] [1] Más adelante... [1]: http://url.gz	Enlace
![Imagen] (rmarkdown.png)	![Imagen] [1] Más adelante... [1]:http://url/b.jpg	
> Blockquote > > --- El Autor		Blockquote — El Autor
* Lista * Lista + Sub-lista	- Lista - Lista - Sub-lista	• Lista • Lista - Sub-lista
1. Uno 2. Dos a. A	1) Uno 2) Dos a) A	1. Uno 2. Dos a. A
Regla horizontal	Regla horizontal	Regla horizontal _____
--- `código en línea` entre comillas invertidas ```` # bloque de código 3 comillas invertidas o sangría de 4 espacios ``````	*** [····]# bloque de código [····]3 comillas invertidas [····]o sangría de 4 espacios os	código en línea entre comillas invertidas # bloque de código 3 comillas invertidas o sangría de 4 espacios

Es muy recomendable dejar siempre una línea de separación entre elementos distintos consecutivos.

3.3 Inclusión de código R

Se puede incluir código R entre los delimitadores ````${r}```` y `````. Por defecto, se mostrará el código, se evaluará y se mostrarán los resultados justo a continuación. Por ejemplo el siguiente código:

```
```${r}```  
head(mtcars[1:3])
```



```
summary(mtcars[1:3])
```

```
...
```

produce:

```
head(mtcars[1:3])
```

```
mpg cyl disp
Mazda RX4 21.0 6 160
Mazda RX4 Wag 21.0 6 160
Datsun 710 22.8 4 108
Hornet 4 Drive 21.4 6 258
Hornet Sportabout 18.7 8 360
Valiant 18.1 6 225
```

```
summary(mtcars[1:3])
```

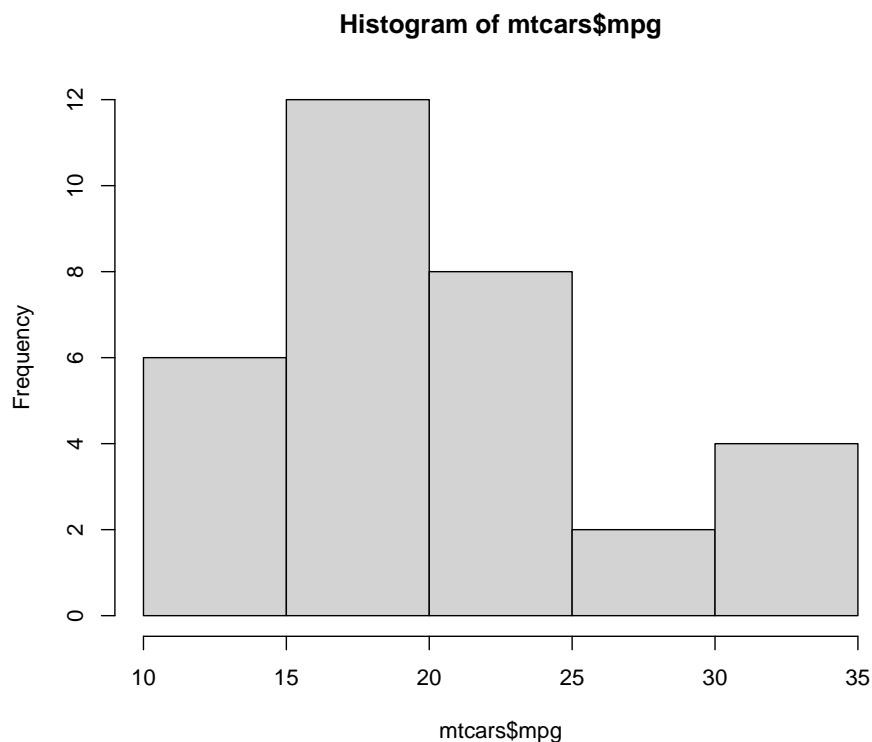
```
mpg cyl disp
Min. :10.40 Min. :4.000 Min. : 71.1
1st Qu.:15.43 1st Qu.:4.000 1st Qu.:120.8
Median :19.20 Median :6.000 Median :196.3
Mean :20.09 Mean :6.188 Mean :230.7
3rd Qu.:22.80 3rd Qu.:8.000 3rd Qu.:326.0
Max. :33.90 Max. :8.000 Max. :472.0
```

En RStudio pulsando *Ctrl + Alt + I* o en el icono correspondiente se incluye un trozo de código.

También se puede incluir código en línea empleando ``r código``, por ejemplo ``r 2 + 2`` produce 4.

### 3.3.1 Gráficos

Si el código genera un gráfico, este se incluirá en el documento justo después de donde fué generado (por defecto). Por ejemplo el siguiente gráfico:



se generó empleando:

```
```${r} figural, echo=FALSE}
hist(mtcars$mpg)
```
```

aunque no se mostró previamente el código al haber establecido la opción ````${r}, echo=FALSE`.

### 3.3.2 Opciones de bloques de código

Los trozos de código pueden tener nombre y opciones, se establecen en la cabecera de la forma ````${r} nombre, op1, op2`. Para un listado de las opciones disponibles ver <http://yihui.name/knitr/options> (en la Sección 2.6 del libro de RMarkdown se incluye un resumen). En RStudio se puede pulsar en los iconos en la parte superior derecha del bloque de código para establecer opciones, ejecutar todo el código anterior o sólo el correspondiente trozo.

Algunas opciones sobre evaluación y resultados:

- `eval`: si `=FALSE` no se evalúa el código.
- `echo`: si `=FALSE` no se muestra el código.
- `include`: si `=FALSE` no se muestra el código ni ningún resultado.
- `message`, `warning`, `error`: oculta el correspondiente tipo de mensaje de R (los errores o warnings se mostrarán en la consola).
- `cache`: si se activa, guarda los resultados de la última evaluación y se reutilizan si no cambió el bloque de código (más detalles aquí). Puede ser de utilidad durante la redacción del documento para reducir el tiempo de renderizado (usándolo con cuidado y desactivándolo al terminar).

Algunas opciones sobre resultados gráficos:

- `fig.width`, `fig.height`, `fig.dim`: dimensiones del dispositivo gráfico de R (no confundir con el tamaño del resultado), e.g. `fig.width = 5`.
- `out.width`, `out.heigh`: tamaño del gráfico, e.g. `'80%'`.
- `fig.align`: `'left'`, `'center'`, `'right'`, establece la alineación.
- `fig.cap`: leyenda de la figura<sup>2</sup>.
- `dev`: dispositivo gráfico de R, por defecto `'pdf'` para LaTeX y `'png'` para HTML. Otras opciones son `'svg'` o `'jpeg'`.

Para establecer valores por defecto para todos los bloques de código se suele incluir uno de configuración al principio del documento, por ejemplo:

```
```${r}, setup, include=FALSE}
knitr::opts_chunk$set(comment=NA, prompt=TRUE, dev='svg', fig.dim=c(5, 7), collapse=TRUE)
```
```

## 3.4 Tablas

En Markdown se pueden escribir tablas en varios formatos (ver Pandoc Tables). Por ejemplo:

| Right | Left | Center | Default |
|-------|------|--------|---------|
| 12    | 12   | 12     | 12      |
| 123   | 123  | 123    | 123     |
| 1     | 1    | 1      | 1       |

También:

| Variable         | Descripción             |
|------------------|-------------------------|
| <code>mpg</code> | Millas / galón (EE.UU.) |

<sup>2</sup>Si se genera un documento en PDF/LaTeX el gráfico se mostrará en un entorno flotante y se puede ajustar la posición empleando la opción `fig.pos` (por ejemplo, `fig.pos = '!htb'`).

Tabla 3.3: Una kable knitr

|                   | mpg  | cyl | disp | hp  | drat | wt    | qsec  | vs | am | gear | carb |
|-------------------|------|-----|------|-----|------|-------|-------|----|----|------|------|
| Mazda RX4         | 21.0 | 6   | 160  | 110 | 3.90 | 2.620 | 16.46 | 0  | 1  | 4    | 4    |
| Mazda RX4 Wag     | 21.0 | 6   | 160  | 110 | 3.90 | 2.875 | 17.02 | 0  | 1  | 4    | 4    |
| Datsun 710        | 22.8 | 4   | 108  | 93  | 3.85 | 2.320 | 18.61 | 1  | 1  | 4    | 1    |
| Hornet 4 Drive    | 21.4 | 6   | 258  | 110 | 3.08 | 3.215 | 19.44 | 1  | 0  | 3    | 1    |
| Hornet Sportabout | 18.7 | 8   | 360  | 175 | 3.15 | 3.440 | 17.02 | 0  | 0  | 3    | 2    |
| Valiant           | 18.1 | 6   | 225  | 105 | 2.76 | 3.460 | 20.22 | 1  | 0  | 3    | 1    |

```

cyl | Número de cilindros
disp | Desplazamiento (pulgadas cúbicas)
hp | Caballos de fuerza bruta
drat | Relación del eje trasero
wt | Peso (miles de libras)
qsec | Tiempo de 1/4 de milla
vs | Cilindros en V/Straight (0 = cilindros en V, 1 = cilindros en línea)
am | Tipo de transmisión (0 = automático, 1 = manual)
gear | Número de marchas (hacia adelante)
carb | Número de carburadores

```

que resulta en:

| Variable | Descripción                                                          |
|----------|----------------------------------------------------------------------|
| mpg      | Millas / galón (EE.UU.)                                              |
| cyl      | Número de cilindros                                                  |
| disp     | Desplazamiento (pulgadas cúbicas)                                    |
| hp       | Caballos de fuerza bruta                                             |
| drat     | Relación del eje trasero                                             |
| wt       | Peso (miles de libras)                                               |
| qsec     | Tiempo de 1/4 de milla                                               |
| vs       | Cilindros en V/Straight (0 = cilindros en V, 1 = cilindros en línea) |
| am       | Tipo de transmisión (0 = automático, 1 = manual)                     |
| gear     | Número de marchas (hacia adelante)                                   |
| carb     | Número de carburadores                                               |

Para convertir resultados de R en tablas de una forma simple se puede emplear la función `ktable()` del paquete `knitr`. Por ejemplo la Tabla 3.3 se obtuvo mediante el siguiente código:

```

knitr::kable(
 head(mtcars),
 caption = "Una kable knitr"
)

```

Otros paquetes proporcionan opciones adicionales: `xtable`, `stargazer`, `pander`, `tables` y `ascii`.

## 3.5 Cabecera YAML

En un fichero RMarkdown se puede incluir metadatos en una cabecera en formato YAML (YAML Ain't Markup Language, <https://en.wikipedia.org/wiki/YAML>), comenzando y terminando con tres guiones `---`. Los metadatos de YAML son típicamente opciones de renderizado consistentes en pares de etiquetas y valores separados por dos puntos. Por ejemplo:

```

title: "Creación de contenidos con RMarkdown"

```

```
author: "Fernández-Casal, R. y Cotos-Yáñez, T.R."
date: "`r Sys.Date()`"
output: html_document

```

Aunque no siempre es necesario, se recomienda que los valores de texto se introduzcan entre comillas (se puede incluir código R en línea, como por ejemplo ``r Sys.Date()`` para obtener la fecha actual). Para valores lógicos se puede emplear `yes/true` y `no/false` para verdadero y falso, respectivamente.

Los valores pueden ser vectores, por ejemplo las siguientes opciones son equivalentes:

```
bibliography: [book.bib, packages.bib]
```

```
bibliography:
- book.bib
- packages.bib
```

También pueden ser listas, añadiendo una sangría de dos espacios (importante):

```
output:
 html_document:
 toc: yes
 toc_float: yes
 pdf_document:
 toc: yes
```

El campo `output` permite especificar el formato y las opciones de salida (por defecto se empleará la primera). Empleando este campo también se pueden especificar opciones gráficas para los bloques de código, por ejemplo:

```
output:
 html_document:
 fig_width: 7
 fig_height: 6
 fig_caption: true
```

La mayoría de los campos YAML son opciones que el paquete `rmarkdown` le pasa a Pandoc (ver documentación en el Apéndice A).

Un ejemplo adicional<sup>3</sup>:

```

title: "Creación de contenidos con RMarkdown"
subtitle: "Curso de introducción a R"
author:
- name: "Rubén Fernández Casal (ruben.fcasal@udc.es)"
 affiliation: "Universidade da Coruña"
- name: "Tomás R. Cotos Yáñez (tcotos@uvigo.es)"
 affiliation: "Universidade de Vigo"
date: "2023-04-01"
logo: rmarkdown.png
output:
 html_document:
 toc: yes # incluir tabla de contenido
 toc_float: yes # toc flotante a la izquierda
 number_sections: yes # numerar secciones y subsecciones
 code_folding: hide # por defecto el código aparecerá oculto
 mathjax: local # emplea una copia local de MathJax, hay que establecer:
 self_contained: false # las dependencias se guardan en ficheros externos
```

<sup>3</sup>Puede ser interesante ejecutar `str(rmarkdown::html_document())` para ver un listado de todas las opciones disponibles de `html_document`

```

lib_dir: libs # directorio para librerías (Bootstrap, MathJax, ...)
pdf_document:
 toc: yes
 toc_depth: 2
 keep_tex: yes # conservar fichero latex

```

Como se puede deducir del ejemplo anterior, en el formato YAML podemos incluir comentarios con el carácter # (por ejemplo para no emplear alguna de las opciones sin borrarla del encabezado).

En el Capítulo 3 del libro de RMarkdown se tiene información detallada sobre las opciones de los distintos formatos de salida (sobre ficheros HTML en la sección HTML document y sobre PDF/LaTeX en PDF document).

## 3.6 Extracción del código R

Para generar un fichero con el código R se puede emplear la función `knitr::ktable()`. Por ejemplo:

```
purl("Informe.Rmd")
```

Si se quiere además el texto RMarkdown como comentarios tipo `spin()`, se puede emplear:

```
purl("Informe.Rmd", documentation = 2)
```

## 3.7 Spin

Una forma rápida de crear este tipo de informes a partir de un fichero de código R es emplear la función `spin()` del paquete `knitr` (ver p.e. <https://yihui.org/knitr/demo/stitch/#spin-comment-out-texts>).

Para ello se debe comentar todo lo que no sea código R de una forma especial:

- El texto RMarkdown se comenta con `#'`.

Por ejemplo:

```
#' # Este es un título de primer nivel
#' ## Este es un título de segundo nivel
```

- Las opciones de un trozo de código se comentan con `#+`.

Por ejemplo:

```
#+ setup, include=FALSE
opts_chunk$set(comment=NA, prompt=TRUE, dev='svg', fig.height=6, fig.width=6)
```

Para generar el informe se puede emplear la función `knitr::purl()`. Por ejemplo: `spin("Ridge_Lasso.R")`. También se podría abrir directamente el informe generado:

```
browseURL(url = knitr::spin("Ridge_Lasso.R"))
```

Pero puede ser recomendable renderizarlo con `rmarkdown`:

```
library(rmarkdown)
browseURL(url = render(knitr::spin("Ridge_Lasso.R", knit = FALSE)))
```

En RStudio basta con pulsar `Ctrl + Shift + K`, el icono correspondiente en la barra superior, o seleccionar `File > Compile Report...`

Por ejemplo, si se quiere convertir la salidas de un fichero de código de R a formato LaTeX (para añadirlas fácilmente a un documento en este formato), bastaría con incluir una cabecera de la forma:

```
#' ---
#' title: "Título"
#' author: "Autor"
```

```
#' date: "Fecha"
#' output:
#' pdf_document:
#' keep_tex: true
#' ---
```

### 3.8 Extensiones RMarkdown de pandoc

Como ya se comentó, RMarkdown utiliza la sintaxis extendida proporcionada por Pandoc. Por ejemplo, se pueden añadir sub<sup>índices</sup> y super<sup>índices</sup> con `sub~índices~` y `super~índices~`, y notas al pie con `^[texto]`.

Podemos incluir expresiones matemáticas en formato LaTeX (ver e.g. Free online introduction to LaTeX):

- En línea escribiendo la expresión latex entre dos símbolos de dolar, por ejemplo `$$\alpha, \beta, \gamma, \delta$` resultaría en  $\alpha, \beta, \gamma, \delta$ .
- En formato ecuación empleando dos pares de símbolos de dolar. Por ejemplo: `$$\Theta = \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}$$` resultaría en:

$$\Theta = \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}$$

También admite bibliografía, ver p.e. Pandoc Citations. Lo más cómodo puede ser emplear un archivo de bibliografía en formato BibTeX, lo que se describe con detalle en Citations. Será necesario añadir un campo `bibliography` en la cabecera YAML, por ejemplo:

```
bibliography: bibliografia.bib
csl: apa.csl # opcional
```

Suponiendo que en el directorio de trabajo están los ficheros de bibliografía `bibliografia.bib` y de estilo `apa.csl` (ver <http://citationstyles.org/>, desde donde se pueden descargar distintos archivos de estilo).

Las referencias en el texto RMarkdown se incluyen con `@referencia` o `[@referencia]`. Pandoc generará el listado de referencias al final del documento, por lo que nos puede interesar insertar una última sección `# Bibliografía {-}` al generar documentos HTML (en PDF se hará automáticamente al emplear LaTeX). En RStudio se puede instalar el “Addin” `cittr` para insertar citas a referencias bibliográficas en formato BibTeX.

Para más detalles de las extensiones de Pandoc ver por ejemplo Pandoc’s Markdown.

**Parte II**

**Tidyverse**





## Capítulo 4

# El ecosistema tidyverse

En los capítulos de esta parte se pretende realizar una breve introducción al *ecosistema Tidyverse*, una colección de paquetes diseñados de forma uniforme (con la misma filosofía y estilo) para trabajar conjuntamente.

La referencia recomendada para usuarios de R que deseen iniciarse en el uso de estos paquetes es:

Wickham, H., y Grolemund, G. (2016). *R for data science: import, tidy, transform, visualize, and model data*, online-castellano, O'Reilly.

El paquete `tidyverse` está diseñado para facilitar la instalación y carga de los paquetes principales de la colección tidyverse con un solo comando. Al instalar este paquete se instalan paquetes que forman el denominado núcleo de tidyverse (se cargan con `library(tidyverse)`):

- `ggplot2`: visualización de datos.
- `dplyr`: manipulación de datos.
- `tidyr`: reorganización (limpieza) de datos.
- `readr`: importación de datos.
- `tibble`: tablas de datos (extensión de `data.frame`).
- `purrr`: programación funcional.
- `stringr`: manipulación de cadenas de texto.
- `forcats`: manipulación de factores.
- `lubridate`: manipulación de fechas y horas.

y un conjunto de paquetes recomendados (`feather`, `haven`, `modelr`, `broom`...), entre los que destacaría:

- `readxl`: archivos excel.
- `hms`: manipulación de medidas de tiempo.
- `httr`: web APIs.
- `jsonlite`: archivos JSON.
- `rvest`: web scraping.
- `xml2`: archivos XML.

```
library(tidyverse)
```

También hay paquetes “asociados”:

- `rlang`
- `tidyselect`
- `tidymodels`

Muchos otros paquetes están adaptando este estilo (ver e.g. `tidyverts`): `fable`, `sf`...

Resumiendo, está muy de moda y puede terminar convirtiéndose en un dialecto del lenguaje R... para mí ya lo es... todo lo que resulte de utilidad es bien venido... Recomiendo evitar estos paquetes en las primeras etapas de formación en R...

El estilo de programación tiene como origen la gramática de `ggplot2` para crear gráficos de forma declarativa, basado a su vez en:

Wilkinson, L. (2005). *The Grammar of Graphics*. Springer.

Yo empleo este paquete como sustituto de los gráficos `lattice`, en algunos informes finales o aplicaciones para empresas, o para gráficos muy especializados. En condiciones normales **prefiero emplear los gráficos estándar** de R (mucho más rápidos de generar y programar).

Para iniciarse en este paquete lo recomendado es consultar los capítulos Data Visualización y Graphics for communication de R for Data Science. También puede resultar de interés la chuleta). La referencia que cubre con mayor profundidad este paquete es:

Wickham, H. (2016). *ggplot2: Elegant graphics for Data Analysis* (3ª edición, en desarrollo junto a Navarro, D. y Pedersen, T.L.). Springer.

Aunque yo recomendaría:

Chang, W. (2023). *The R Graphics Cookbook*. O'Reilly.

En `ggplot2` se emplea el operador `+` para añadir componentes de los gráficos (ver , en *Tidyverse* se emplea un operador de redirección para añadir operaciones.

## 4.1 Operador *pipe* (redirección)

El operador `%>%` (paquete `magrittr`) permite canalizar la salida de una función a la entrada de otra. Por ejemplo, `segundo(primer(datos))` se traduce en `datos %>% primero %>% segundo`, lo que facilita la lectura de operaciones al escribir las funciones de izquierda a derecha.

Desde la versión 4.1 de R está disponible un operador interno `|>` (aunque yo sigo prefiriendo `%>%`). Por ejemplo:

```
El fichero 'empleados.RData' contiene datos de empleados de un banco.
Supongamos por ejemplo que estamos interesados en estudiar si hay
discriminación por cuestión de sexo o raza.

load("datos/empleados.RData")
NOTA: Cuidado con la codificación latin1 (no declarada)
al abrir archivos creados en versiones anteriores de R < 4.2:
load("datos/empleados.latin1.RData")

Listamos las etiquetas
knitr::kable(attr(empleados, "variable.labels"), col.names = "Etiqueta")
```

|          | Etiqueta                      |
|----------|-------------------------------|
| id       | Código de empleado            |
| sexo     | Sexo                          |
| fechnac  | Fecha de nacimiento           |
| educ     | Nivel educativo (años)        |
| catlab   | Categoría laboral             |
| salario  | Salario actual                |
| salini   | Salario inicial               |
| tiempemp | Meses desde el contrato       |
| expprev  | Experiencia previa (meses)    |
| minoría  | Clasificación étnica          |
| sexoraza | Clasificación por sexo y raza |

```
Eliminamos las etiquetas para que no molesten...
attr(empleados, "variable.labels") <- NULL
```

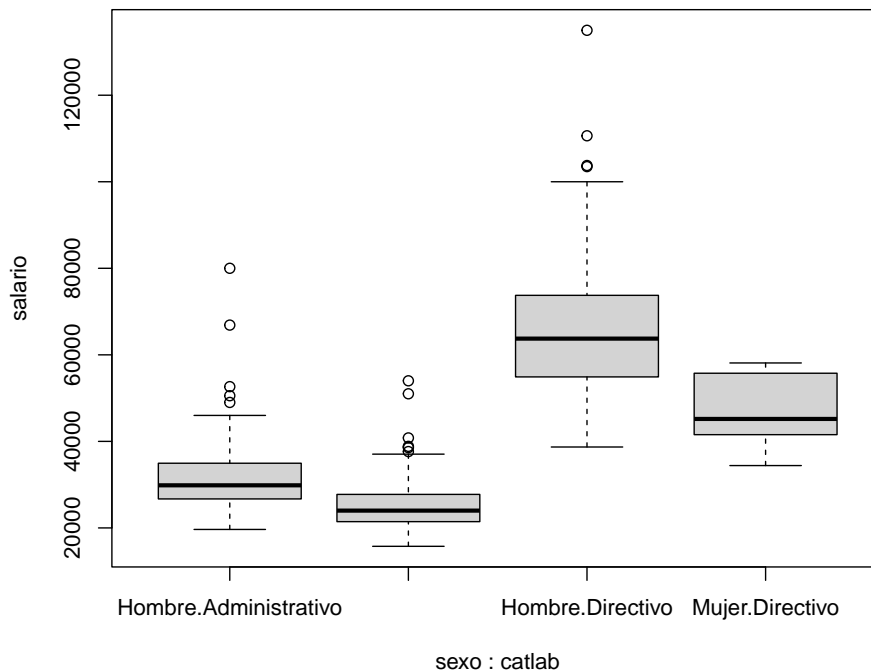
```
empleados |> subset(catlab == "Directivo", catlab:sexoraza) |> summary()
```

```
catlab salario salini tiempemp
Administrativo: 0 Min. : 34410 Min. :15750 Min. :64.00
Seguridad : 0 1st Qu.: 51956 1st Qu.:23063 1st Qu.:73.00
Directivo :84 Median : 60500 Median :28740 Median :81.00
Mean : 63978 Mean :30258 Mean :81.15
3rd Qu.: 71281 3rd Qu.:34058 3rd Qu.:91.00
Max. :135000 Max. :79980 Max. :98.00
expprev minoria sexoraza
Min. : 3.00 No:80 Blanca varón :70
1st Qu.: 19.75 Sí: 4 Blanca mujer : 4
Median : 52.00 Minoría varón:10
Mean : 77.62 Minoría mujer: 0
3rd Qu.:125.25
Max. :285.00
```

Para que una función sea compatible con este tipo de operadores el primer parámetro debería ser siempre los datos. Sin embargo, el operador `%>%` permite redirigir el resultado de la operación anterior a un parámetro distinto mediante un `..` Por ejemplo:

```
"?"|>"
empleados |> subset(catlab != "Seguridad") |> droplevels |>
boxplot(salario ~ sexo*catlab, data = .) # ERROR

library(magrittr)
empleados %>% subset(catlab != "Seguridad") %>% droplevels() %>%
 boxplot(salario ~ sexo*catlab, data = .)
```





## Capítulo 5

# Manipulación de datos con dplyr y tidyr

En este capítulo se realiza una breve introducción al paquete `dplyr` y se comentan algunas de las utilidades del paquete `tidyr` que pueden resultar de interés<sup>1</sup>.

La referencia recomendada para iniciarse en esta herramienta es el Capítulo 5 Data transformation de R for Data Science. También puede resultar de utilidad la viñeta del paquete Introduction to dplyr o la chuleta (menú de RStudio *Help > Cheat Sheets > Data Transformation with dplyr*).

### 5.1 El paquete dplyr

```
library(dplyr)
```

La principal ventaja de `dplyr` es que permite trabajar (de la misma forma) con datos en distintos formatos:

- `data.frame`, `tibble`.
- `data.table`: extensión (paquete *backend*) `dtplyr`.
- conjuntos de datos más grandes que la memoria disponible: extensiones `duckdb` y `arrow` (incluyendo almacenamiento en la nube, e.g. AWS).
- bases de datos relacionales (lenguaje SQL, locales o remotas); extensión `dbplyr`.
- grandes volúmenes de datos (incluso almacenados en múltiples servidores; ecosistema Hadoop/Spark): extensión `sparklyr` (ver menú de RStudio *Help > Cheat Sheets > Interfacing Spark with sparklyr*).

El paquete `dplyr` permite sustituir operaciones con funciones base de R (como `subset`, `split`, `apply`, `sapply`, `lapply`, `tapply`, `aggregate`...) por una “gramática” más sencilla para la manipulación de datos. En lugar de operar sobre vectores como la mayoría de las funciones base, opera sobre conjuntos de datos (de forma que es compatible con el operador `%>%`). Los principales “verbos” (funciones) son:

- `select()`: seleccionar variables (ver también `rename`, `relocate`, `pull`).
- `mutate()`: crear variables (ver también `transmute()`).
- `filter()`: seleccionar casos/filas (ver también `slice()`).
- `arrange()`: ordenar casos/filas.
- `summarise()`: resumir valores.

---

<sup>1</sup>Otra alternativa (más rápida) es `data.table` pero en versiones recientes ya se puede emplear desde `dplyr`, como se comenta más adelante.

- `group_by()`: permite operaciones por grupo empleando el concepto “dividir-aplicar-combinar” (`ungroup()` elimina el agrupamiento).

NOTA: Para entender el funcionamiento de ciertas funciones (como `rowwise()`) y las posibilidades en el manejo de datos, hay que tener en cuenta que un `data.frame` no es más que una lista cuyas componentes (variables) tienen la misma longitud. Realmente las componentes también pueden ser listas de la misma longitud y, por tanto, podemos almacenar casi cualquier estructura de datos en un `data.frame`.

En la primera parte de este capítulo consideraremos solo `data.frame` por comodidad. Emplearemos como ejemplo los datos de empleados de banca almacenados en el fichero `empleados.RData` (y supondremos que estamos interesados en estudiar si hay discriminación por cuestión de sexo o raza).

```
load("datos/empleados.RData")
attr(empleados, "variable.labels") <- NULL
```

En la Sección 5.6 final emplearemos una base de datos relacional como ejemplo.

## 5.2 Operaciones con variables (columnas)

Podemos seleccionar variables con `select()`:

```
emplea2 <- empleados %>% select(id, sexo, minoria, tiempemp, salini, salario)
head(emplea2)
```

```
id sexo minoria tiempemp salini salario
1 1 Hombre No 98 27000 57000
2 2 Hombre No 98 18750 40200
3 3 Mujer No 98 12000 21450
4 4 Mujer No 98 13200 21900
5 5 Hombre No 98 21000 45000
6 6 Hombre No 98 13500 32100
```

Se puede cambiar el nombre (ver también `rename()`):

```
empleados %>% select(sexo, noblanca = minoria, salario) %>% head()
```

```
sexo noblanca salario
1 Hombre No 57000
2 Hombre No 40200
3 Mujer No 21450
4 Mujer No 21900
5 Hombre No 45000
6 Hombre No 32100
```

Se pueden emplear los nombres de variables como índices:

```
empleados %>% select(sexo:salario) %>% head()
```

```
sexo fechnac educ catlab salario
1 Hombre 1952-02-03 15 Directivo 57000
2 Hombre 1958-05-23 16 Administrativo 40200
3 Mujer 1929-07-26 12 Administrativo 21450
4 Mujer 1947-04-15 8 Administrativo 21900
5 Hombre 1955-02-09 15 Administrativo 45000
6 Hombre 1958-08-22 15 Administrativo 32100
```

```
empleados %>% select(-(sexo:salario)) %>% head()
empleados %>% select(!(sexo:salario)) %>% head()
```

```
id salini tiempemp expprev minoria sexoraza
1 1 27000 98 144 No Blanca varón
```

```
2 2 18750 98 36 No Blanca varón
3 3 12000 98 381 No Minoría varón
4 4 13200 98 190 No Minoría varón
5 5 21000 98 138 No Blanca varón
6 6 13500 98 67 No Blanca varón
```

Se pueden emplear distintas herramientas (*selection helpers*) para seleccionar variables (ver paquete `tidyselect`):

- `starts_with`, `ends_with`, `contains`, `matches`, `num_range`: variables que coincidan con un patrón.
- `all_of`, `any_of`: variables de un vector de caracteres.
- `everything`, `last_col`: todas las variables o la última variable.
- `where()`: a partir de una función (e.g. `where(is.numeric)`)

Por ejemplo:

```
empleados %>% select(starts_with("s")) %>% head()
```

```
sexo salario salini sexoraza
1 Hombre 57000 27000 Blanca varón
2 Hombre 40200 18750 Blanca varón
3 Mujer 21450 12000 Minoría varón
4 Mujer 21900 13200 Minoría varón
5 Hombre 45000 21000 Blanca varón
6 Hombre 32100 13500 Blanca varón
```

Podemos crear variables con `mutate()`:

```
emplea2 %>% mutate(incsal = salario - salini, tsal = incsal/tiempemp) %>% head()
```

```
id sexo minoria tiempemp salini salario incsal tsal
1 1 Hombre No 98 27000 57000 30000 306.12245
2 2 Hombre No 98 18750 40200 21450 218.87755
3 3 Mujer No 98 12000 21450 9450 96.42857
4 4 Mujer No 98 13200 21900 8700 88.77551
5 5 Hombre No 98 21000 45000 24000 244.89796
6 6 Hombre No 98 13500 32100 18600 189.79592
```

## 5.3 Operaciones con casos (filas)

Podemos seleccionar casos con `filter()`:

```
emplea2 %>% filter(sexo == "Mujer", minoria == "Sí") %>% head()
```

```
id sexo minoria tiempemp salini salario
1 14 Mujer Sí 98 16800 35100
2 23 Mujer Sí 97 11100 24000
3 24 Mujer Sí 97 9000 16950
4 25 Mujer Sí 97 9000 21150
5 40 Mujer Sí 96 9000 19200
6 41 Mujer Sí 96 11550 23550
```

Podemos reordenar casos con `arrange()`:

```
emplea2 %>% arrange(salario) %>% head()
```

```
id sexo minoria tiempemp salini salario
1 378 Mujer No 70 10200 15750
2 338 Mujer No 74 10200 15900
```

```
3 90 Mujer No 92 9750 16200
4 224 Mujer No 82 10200 16200
5 411 Mujer No 68 10200 16200
6 448 Mujer Si 66 10200 16350
```

```
emplea2 %>% arrange(desc(salini), salario) %>% head()
```

```
id sexo minoria tiempemp salini salario
1 29 Hombre No 96 79980 135000
2 343 Hombre No 73 60000 103500
3 205 Hombre No 83 52500 66750
4 160 Hombre No 86 47490 66000
5 431 Hombre No 66 45000 86250
6 32 Hombre No 96 45000 110625
```

Podemos resumir valores con `summarise()`:

```
empleados %>% summarise(sal.med = mean(salario), n = n())
```

```
sal.med n
1 34419.57 474
```

Para realizar **operaciones con múltiples variables podemos emplear `across()`** (admite selección de variables `tidyselect`):

```
empleados %>% summarise(across(where(is.numeric), mean), n = n())
```

```
id educ salario salini tiempemp expprev n
1 237.5 13.49156 34419.57 17016.09 81.1097 95.86076 474
```

```
empleados %>% summarise(across(where(is.numeric) & !id, mean), n = n())
```

NOTA: Esta función sustituye a las “variantes de ámbito” `_at()`, `_if()` y `_all()` de versiones anteriores de `dplyr` (como `summarise_at()`, `summarise_if()`, `summarise_all()`, `mutate_at()`, `mutate_if()`...) y también el uso de `vars()`. En el caso de `filter()` se puede emplear `if_any()` e `if_all()`.

Podemos **agrupar casos con `group_by()`**:

```
empleados %>% group_by(sexo, minoria) %>%
 summarise(sal.med = mean(salario), n = n()) %>%
 ungroup()
```

```
A tibble: 4 x 4
sexo minoria sal.med n
<fct> <fct> <dbl> <int>
1 Hombre No 44475. 194
2 Hombre Si 32246. 64
3 Mujer No 26707. 176
4 Mujer Si 23062. 40
```

```
empleados %>% group_by(sexo, minoria) %>%
 summarise(sal.med = mean(salario), n = n(), .groups = "drop")
```

```
A tibble: 4 x 4
sexo minoria sal.med n
<fct> <fct> <dbl> <int>
1 Hombre No 44475. 194
2 Hombre Si 32246. 64
3 Mujer No 26707. 176
4 Mujer Si 23062. 40
```



```
dplyr >= 1.1.0 # packageVersion("dplyr")
empleados %>% summarise(sal.med = mean(salario), n = n(),
.by = c(sexo, minoria))
```

Por defecto la agrupación se mantiene para el resto de operaciones, habría que emplear `ungroup()` (o el argumento `.groups = "drop"`) para eliminarla (se puede emplear `group_vars()` o `str()` para ver la agrupación). Desde `dplyr` 1.1.0 (2023-01-29) está disponible un parámetro `.by/by` en `mutate()`, `summarise()`, `filter()` y `slice()` como alternativa a agrupar y desagrupar posteriormente. Para más detalles ver [Per-operation grouping with .by/by](#).

## 5.4 Herramientas tidyr

Algunas funciones del paquete `tidyr` que pueden resultar de especial interés son:

- `pivot_wider()`: permite transformar valores de grupos de casos a nuevas variables.
- `pivot_longer()`: realiza la transformación inversa, colapsar varias columnas en una.

Ver la viñeta [Pivoting](#) para más detalles.

- `separate()`: permite separar una columna de texto en varias (ver también `extract()`).

Ver [mortalidad.R](#) en ejemplos.

## 5.5 Operaciones con tablas de datos

Se emplean funciones `xxx_join()` (ver la documentación del paquete [Join two tbls together](#), o la vignette [Two-table verbs](#)):

- `inner_join()`: devuelve las filas de `x` que tienen valores coincidentes en `y`, y todas las columnas de `x` e `y`. Si hay varias coincidencias entre `x` e `y`, se devuelven todas las combinaciones.
- `left_join()`: devuelve todas las filas de `x` y todas las columnas de `x` e `y`. Las filas de `x` sin correspondencia en `y` contendrán `NA` en las nuevas columnas. Si hay varias coincidencias entre `x` e `y`, se devuelven todas las combinaciones (duplicando las filas).

`right_join()` hace lo contrario, devuelve todas las filas de `y`.

`full_join()` devuelve todas las filas de `x` e `y` (duplicando o asignando `NA` si es necesario).

- `semi_join()`: devuelve las filas de `x` que tienen valores coincidentes en `y`, manteniendo sólo las columnas de `x` (al contrario que `inner_join()` no duplica filas).

`anti_join()` hace lo contrario, devuelve las filas sin correspondencia.

El parámetro `by` determina las variables clave para las correspondencias. Si no se establece se considerarán todas las que tengan el mismo nombre en ambas tablas. Se puede establecer a un vector de nombres coincidentes y en caso de que los nombres sean distintos a un vector con nombres de la forma `c("clave_x" = "clave_y")`.

Adicionalmente, si las tablas `x` e `y` tienen las mismas variables, se pueden combinar las observaciones con operaciones de conjuntos:

- `intersect(x, y)`: observaciones en `x` y en `y`.
- `union(x, y)`: observaciones en `x` o `y` no duplicadas.
- `setdiff(x, y)`: observaciones en `x` pero no en `y`.

## 5.6 Bases de datos con dplyr

Para poder usar tablas en bases de datos relacionales con `dplyr` hay que emplear el paquete `dbplyr` (convierte automáticamente el código de `dplyr` en consultas SQL).

Algunos enlaces:

- Best Practices in Working with Databases
- Introduction to dbplyr
- Data Carpentry: SQL databases and R,
- R and Data – When Should we Use Relational Databases?

### 5.6.1 Ejemplos

Como ejemplo emplearemos la base de datos de SQLite Sample Database Tutorial, almacenada en el archivo *chinook.db*.

```
install.packages('dbplyr')
library(dplyr)
library(dbplyr)
```

En primer lugar hay que conectar la base de datos:

```
chinook <- DBI::dbConnect(RSQLite::SQLite(), "datos/chinook.db")
```

Podemos listar las tablas:

```
src_dbi(chinook)
```

```
src: sqlite 3.36.0 [D:\OneDrive - Universidade da Coruña_Actual_IGE_book_notasr\datos\chinook.db]
tbls: albums, artists, customers, employees, genres, invoice_items, invoices,
media_types, playlist_track, playlists, sqlite_sequence, sqlite_stat1, tracks
```

Para enlazar una tabla:

```
invoices <- tbl(chinook, "invoices")
invoices
```

```
Source: table<invoices> [?? x 9]
Database: sqlite 3.36.0 [D:\OneDrive - Universidade da
Coruña_Actual_IGE_book_notasr\datos\chinook.db]
InvoiceId CustomerId InvoiceDate BillingCity BillingState BillingCountry BillingPostalCode Total
<int> <int> <chr> <chr> <chr> <chr> <chr> <dbl>
1 1 2 2009-01-02 Theodo~ Stuttg~ <NA> Germany 70174 1.98
2 2 4 2009-01-04 Ullevå~ Oslo <NA> Norway 0171 3.96
3 3 8 2009-01-08 Grétry~ Brusse~ <NA> Belgium 1000 5.94
4 4 14 2009-01-14 8210 1~ Edmont~ AB Canada T6G 2C7 8.91
5 5 23 2009-01-16 69 Sal~ Boston MA USA 2113 13.9
6 6 37 2009-01-17 Berger~ Frankf~ <NA> Germany 60316 0.99
7 7 38 2009-02-08 Barbar~ Berlin <NA> Germany 10779 1.98
8 8 40 2009-02-08 8, Rue~ Paris <NA> France 75002 1.98
9 9 42 2009-02-09 9, Pla~ Bordea~ <NA> France 33000 3.96
10 10 46 2009-02-03 3 Chat~ Dublin Dublin Ireland <NA> 5.94
... with more rows, and abbreviated variable names 1: InvoiceDate,
2: BillingAddress, 3: BillingCity, 4: BillingState, 5: BillingCountry,
6: BillingPostalCode
```

Ojo [?? x 9]: de momento no conoce el número de filas.

```
nrow(invoices)
```

```
[1] NA
```

Podemos mostrar la consulta SQL correspondiente a una operación:

```
show_query(head(invoices))
```

```
<SQL>
SELECT *
FROM `invoicess`
LIMIT 6
str(head(invoicess))
```

Al trabajar con bases de datos, dplyr intenta ser lo más vago posible:

- No exporta datos a R a menos que se pida explícitamente (`collect()`).
- Retrasa cualquier operación lo máximo posible: agrupa todo lo que se desea hacer y luego hace una única petición a la base de datos.

```
invoicess %>% head %>% collect
```

```
A tibble: 6 x 9
InvoiceId CustomerId InvoiceDate Billi~1 Billi~2 Billi~3 Billi~4 Billi~5 Total
<int> <int> <chr> <chr> <chr> <chr> <chr> <chr> <dbl>
1 1 2 2009-01-01~ Theodo~ Stuttg~ <NA> Germany 70174 1.98
2 2 4 2009-01-02~ Ullevå~ Oslo <NA> Norway 0171 3.96
3 3 8 2009-01-03~ Grétrý~ Brusse~ <NA> Belgium 1000 5.94
4 4 14 2009-01-06~ 8210 1~ Edmont~ AB Canada T6G 2C7 8.91
5 5 23 2009-01-11~ 69 Sal~ Boston MA USA 2113 13.9
6 6 37 2009-01-19~ Berger~ Frankf~ <NA> Germany 60316 0.99
... with abbreviated variable names 1: BillingAddress, 2: BillingCity,
3: BillingState, 4: BillingCountry, 5: BillingPostalCode
```

```
invoicess %>% count # número de filas
```

```
Source: lazy query [?? x 1]
Database: sqlite 3.36.0 [D:\OneDrive - Universidade da
Coruña__Actual__IGE__book_notasr\datos\chinook.db]
n
<int>
1 412
```

Por ejemplo, para obtener el importe mínimo, máximo y la media de las facturas:

```
res <- invoicess %>% summarise(min = min(Total, na.rm = TRUE),
 max = max(Total, na.rm = TRUE), med = mean(Total, na.rm = TRUE))
show_query(res)
res %>% collect
```

```
A tibble: 1 x 3
min max med
<dbl> <dbl> <dbl>
1 0.99 25.9 5.65
```

Para obtener el total de las facturas de cada uno de los países:

```
res <- invoicess %>% group_by(BillingCountry) %>%
 summarise(n = n(), total = sum(Total, na.rm = TRUE))
show_query(res)
res %>% collect
```

```
A tibble: 24 x 3
BillingCountry n total
<chr> <int> <dbl>
1 Argentina 7 37.6
2 Australia 7 37.6
3 Austria 7 42.6
4 Belgium 7 37.6
```

```
5 Brazil 35 190.
6 Canada 56 304.
7 Chile 7 46.6
8 Czech Republic 14 90.2
9 Denmark 7 37.6
10 Finland 7 41.6
... with 14 more rows
```

Para obtener un listado con Nombre y Apellidos de cliente y el importe de cada una de sus facturas (Hint: WHERE customer.CustomerID=invoices.CustomerID):

```
customers <- tbl(chinook, "customers")
tbl_vars(customers)
```

```
<dplyr:::vars>
[1] "CustomerId" "FirstName" "LastName" "Company" "Address"
[6] "City" "State" "Country" "PostalCode" "Phone"
[11] "Fax" "Email" "SupportRepId"
```

```
res <- customers %>% inner_join(invoices, by = "CustomerId") %>% select(FirstName, LastName, Country)
show_query(res)
```

```
<SQL>
SELECT `FirstName`, `LastName`, `Country`, `Total`
FROM (SELECT `LHS`.`CustomerId` AS `CustomerId`, `FirstName`, `LastName`, `Company`, `Address`, `City`, `State`, `Country`, `PostalCode`, `Phone`, `Fax`, `Email`, `SupportRepId`
FROM `customers` AS `LHS`
INNER JOIN `invoices` AS `RHS`
ON (`LHS`.`CustomerId` = `RHS`.`CustomerId`))
)
```

```
res %>% collect
```

```
A tibble: 412 x 4
FirstName LastName Country Total
<chr> <chr> <chr> <dbl>
1 Luís Gonçalves Brazil 3.98
2 Luís Gonçalves Brazil 3.96
3 Luís Gonçalves Brazil 5.94
4 Luís Gonçalves Brazil 0.99
5 Luís Gonçalves Brazil 1.98
6 Luís Gonçalves Brazil 13.9
7 Luís Gonçalves Brazil 8.91
8 Leonie Köhler Germany 1.98
9 Leonie Köhler Germany 13.9
10 Leonie Köhler Germany 8.91
... with 402 more rows
```

Para listar los 10 mejores clientes (aquellos a los que se les ha facturado más cantidad) indicando Nombre, Apellidos, País y el importe total de su facturación:

```
customers %>% inner_join(invoices, by = "CustomerId") %>% group_by(CustomerId) %>%
 summarise(FirstName, LastName, country, total = sum(Total, na.rm = TRUE)) %>%
 arrange(desc(total)) %>% head(10) %>% collect
```

```
A tibble: 10 x 5
CustomerId FirstName LastName Country total
<int> <chr> <chr> <chr> <dbl>
1 6 Helena Holý Czech Republic 49.6
2 26 Richard Cunningham USA 47.6
3 57 Luis Rojas Chile 46.6
4 45 Ladislav Kovács Hungary 45.6
```

|    |    |    |        |            |         |      |
|----|----|----|--------|------------|---------|------|
| ## | 5  | 46 | Hugh   | O'Reilly   | Ireland | 45.6 |
| ## | 6  | 28 | Julia  | Barnett    | USA     | 43.6 |
| ## | 7  | 24 | Frank  | Ralston    | USA     | 43.6 |
| ## | 8  | 37 | Fynn   | Zimmermann | Germany | 43.6 |
| ## | 9  | 7  | Astrid | Gruber     | Austria | 42.6 |
| ## | 10 | 25 | Victor | Stevens    | USA     | 42.6 |

Al finalizar hay que desconectar la base de datos:

```
DBI::dbDisconnect(chinook)
```



# Referencias

- Fernández-Casal R., Costa J. y Oviedo de la Fuente, M. (2021). *Aprendizaje Estadístico*. github.
- Fernández-Casal R., Roca-Pardiñas J., Costa J. y Oviedo-de la Fuente M. (2023). *Introducción al Análisis de Datos con R*. ISBN: 978-84-09-41823-7. github.
- Grolemund, G. (2014). *Hands-on programming with R: Write your own functions and simulations*, O'Reilly.
- Kuhn, M. y Silge, J. (2022). *Tidy Modeling with R*. O'Reill.
- Matloff, N. (2011). *The art of R programming: A tour of statistical software design*, No Starch Press.
- Wickham, H. (2015). *R packages: organize, test, document, and share your code* (actualmente 2ª edición en desarrollo con H. Bryan), O'Reilly, 1ª edición.
- Wickham, H. (2019). *Advanced R, 2ª edición*, Chapman & Hall, 1ª edición..
- Wickham, H., y Grolemund, G. (2016). *R for data science: import, tidy, transform, visualize, and model data*, online-castellano, O'Reilly.
- NOTA: En la bibliografía complementaria se incluyen algunas de estas referencias, y una selección de libros en abierto, organizados por temas.

## Bibliografía por temas

### *En preparación...*

A continuación se muestra una selección de **libros en abierto** (y algún manual) que considero que pueden resultar de utilidad. Para referencias adicionales recomiendo consultar:

- Baruffa, O. (2022). *Big Book of R: Your last-ever bookmark (hopefully...)*.

### Iniciación a la programación en R

- Wickham, H., y Grolemund, G. (2016). *R for data science: import, tidy, transform, visualize, and model data*, online-castellano, O'Reilly.
- Grolemund, G. (2014). *Hands-on programming with R: Write your own functions and simulations*, O'Reilly.
- Fernández-Casal, R., Roca-Pardiñas, J., Costa, J. y Oviedo de la Fuente, M. (2022). *Introducción al Análisis de Datos con R*. github.
- Peng, R.D. (2022). *R Programming for Data Science*, Leanpub.

### Programación avanzada en R

- Wickham, H. (2019). *Advanced R, 2ª edición*, Chapman & Hall, 1ª edición.
- Grosser, M., Bumann, H., Wickham, H. (2021). *Advanced R Solutions*. Chapman & Hall/CRC.

- Wickham, H. (2015). *R packages: organize, test, document, and share your code* (actualmente 2ª edición en desarrollo con H. Bryan), O'Reilly, 1ª edición.
- Gillespie, C. y Lovelace, R. (2016). *Efficient R programming*. O'Reilly.

## Rmarkdown y Shiny

- Xie, Y., Allaire, J.J. y Golemund, G. (2022): *R Markdown: The Definitive Guide*. Chapman & Hall/CRC.
- Xie, Y., Dervieux, C., Riederer, E. (2021). *R Markdown Cookbook*. Chapman & Hall/CRC.
- Fernández-Casal, R. y Cotos-Yáñez, T.R. (2018). *Escritura de libros con bookdown*, github.
- Xie, Y. (2016): *Authoring Books and Technical Documents with R Markdown*. Chapman & Hall/CRC.
- Wickham, H. (2021). *Mastering Shiny*. O'Reilly.
- Sievert, C. (2020). *Interactive Web-based Data Visualization with R, Plotly, and Shiny*. Chapman & Hall/CRC.
- Granjon, D. (2022). *Outstanding User Interfaces with Shiny*. Chapman & Hall/CRC.
- Rochette, S., Fay, C., Girard, C., Guyader, V. (2021). *Engineering Production-Grade Shiny Apps*. Chapman & Hall/CRC.

## Gráficos

- Chang, W. (2023). *The R Graphics Cookbook*. O'Reilly.
- Wickham, H. (2016). *ggplot2: Elegant graphics for Data Analysis* (3ª edición, en desarrollo junto a Navarro, D. y Pedersen, T.L.). Springer.
- Wilke, C. (2019). *Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures*. O'Reilly.

## Regresión y aprendizaje estadístico

- Fernández-Casal, R., Costa, J. y Oviedo de la Fuente, M. (2021). *Aprendizaje Estadístico*. github.
- Kuhn, M., y Silge, J. (2022). *Tidy Modeling with R*, O'Reilly.
- Luraschi, J., Kuo, K. y Ruiz, E. (2019). *Mastering Spark with R*. O'Reilly.
- Irizarry, R.A. (2020). *Introduction to Data Science: Data Analysis and Prediction Algorithms with R*. Chapman & Hall/CRC.
- Roback, P. y Legler, J. (2021). *Beyond Multiple Linear Regression: Applied Generalized Linear Models and Multilevel Models in R*. Chapman & Hall/CRC.
- Greenwell, B.M. y Boehmke, B. (2019). *Hands-On Machine Learning with R*. Chapman & Hall/CRC.
- Ismay, C. y Kim, A. Y. (2019). *Statistical Inference Via Data Science: A Modern Dive Into R and the Tidyverse*. Chapman & Hall/CRC.
- Silge, J. y Robinson, D. (2017). *Text Mining with R*. O'Reilly.
- Voigt, S., Scheuch, C. y Weiss, P. (2023). *Tidy Finance with R*. Chapman & Hall/CRC.
- Fernández-Casal, R., Cao, R. y Costa, J. (2023). *Técnicas de Simulación y Remuestreo* (github). La anterior edición (Fernández-Casal R. y Cao R., 2022, *Simulación Estadística*) está disponible en la rama *primera\_edicion*.
- García-Portugués, E. (2023). *Notes for Predictive Modeling*.



- García-Portugués, E. (2023). *Notes for Nonparametric Statistics*.
- Hanck, c., Arnold, M., Gerber, A. y Schmelzer, M. (2023). *Introduction to Econometrics with R*.
- Fieberg, J. (2022). *Statistics for Ecologists: A Frequentist and Bayesian Treatment of Modern Regression Models*.

## Datos temporales y espaciales

- Fernández-Casal, R. y Cotos-Yáñez, T.R. (2021). *Estadística Espacial con R*. github.
- Hyndman, R.J., y Athanasopoulos, G. (2021). *Forecasting: principles and practice*. OTexts.
- Lovelace, R., Nowosad, J., y Muenchow, J. (2019). *Geocomputation with R*. CRC.
- Moraga, P. (2019). *Geospatial health data: Modeling and visualization with R-INLA and shiny*, CRC.
- Pebesma, E., y Bivand, R. (2023). *Spatial Data Science: with applications in R*. Chapman & Hall/CRC.
- Wikle, C.K., Zammit-Mangion, A. y Cressie, N. (2019). *Spatio-temporal Statistics with R*. Chapman & Hall/CRC.
- Gomez-Rubio, V. (2020). *Bayesian inference with INLA*. Chapman & Hall/CRC.
- Haddon, M. (2020). *Using R for Modelling and Quantitative Methods in Fisheries*. Chapman & Hall/CRC.
- J. Brus, D.J. (2023). *Spatial sampling with R*. Chapman & Hall/CRC.
- Holmes, E.E., Scheuerell, M.D. y Ward, E.J. (2021). *Applied Time Series Analysis for Fisheries and Environmental Sciences*.
- *Spatial Data Science with R and “terra”*.

## Datos faltantes

- Buuren, S. (2018). *Flexible Imputation of Missing Data*, Chapman & Hall.

## Manuales oficiales R

<https://cran.r-project.org/manuals.html>

- Intro to R
- R Data Import/Export
- R Installation and Administration
- Writing R extensions
- The R language definition
- R Internals

## Enlaces

**Repositorio:** [rubenfcasal/book\\_notasr](https://github.com/rubenfcasal/book_notasr)

**Recursos para el aprendizaje de R:** En este post se muestran algunos recursos que pueden ser útiles para el aprendizaje de R y la obtención de ayuda.

**Bookdown:**

- Notas de programación en R

- Introducción a RMarkdown.

*Posit (RStudio)*

- Blog
- Videos
- Chuletas (Cheatsheets)
- *tidyverse*:
  - dplyr
  - tibble
  - tidyr
  - stringr
  - readr
  - Best Practices in Working with Databases
- tidymodels
- sparklyr
- shiny

# Apéndice A

## Pandoc

Pandoc es un conversor de documentos libre y de código abierto, Pandoc puede leer archivos en distintos formatos, incluyendo:

- Distintos dialectos de Markdown
- HTML
- LaTeX
- Microsoft Word docx
- LibreOffice ODT
- EPUB

Puede convertir los documentos de entrada a muchos otros formatos, incluyendo Office Open XML, OpenDocument, HTML, Wiki markup, InDesign ICML, ebooks, OPML, y varios formatos basados en TeX (desde donde se puede producir un PDF). En la web oficial <https://pandoc.org> hay un listado completo de los formatos soportados. Pandoc también proporciona distintas extensiones de Markdown para que admita resultados más complejos.

Pandoc es una herramienta independiente de línea de comandos (sin interfaz gráfica), que se instala automáticamente con RStudio porque el paquete `rmarkdown` la emplea para generar los documentos de salida a partir de documentos Markdown (por ejemplo, en Windows en `C:\Program Files\RStudio\bin\pandoc\pandoc.exe`).

### A.1 Conversión de documentos con Pandoc

La sintaxis del comando es `pandoc [opciones] [ficheros]`. Si se ejecuta `pandoc --help`, en la ventana de comandos o en la pestaña *Terminal* de RStudio, se obtiene un listado detallado de las opciones. También se puede consultar el manual de Pandoc <https://pandoc.org/MANUAL.html>.

Si Pandoc no está configurado en la ruta de búsqueda, habrá que reemplazar `pandoc` por la ruta completa al ejecutable. Por ejemplo, para emplear la versión instalada con RStudio en Windows habrá que introducir `"C:\Program Files\RStudio\bin\pandoc\pandoc"` si la versión de RStudio es anterior a *v2022.07* y `"C:\Program Files\RStudio\bin\quarto\bin\tools\pandoc.exe"` en caso contrario (ver Hello, Quarto).

Podemos emplear Pandoc para convertir contenido escrito en otros formatos a Markdown, por ejemplo:

- Un fichero word a markdown:

```
"C:\Program Files\RStudio\bin\pandoc\pandoc" fichero.docx -f docx -t markdown
--extract-media . -o fichero.Rmd
```

Ver `docx2md.bat` y `alldocx2md.bat` (o `alldocx2md2.bat` para RStudio  $\geq$  *v2022.07* - *Quarto*).

- Un fichero LaTeX a markdown:

```
"C:\Program Files\RStudio\bin\pandoc\pandoc" fichero.tex -f latex -t markdown
-o fichero.Rmd
```

Ver tex2md.bat y alltex2md.bat (o alltex2md2.bat para RStudio  $\geq$  v2022.07 - *Quarto*).

- Una web a markdown:

```
"C:\Program Files\RStudio\bin\pandoc\pandoc" http://url.org -f html -t markdown
-o fichero.Rmd
```

Ver web2md.bat (o web2md2.bat para RStudio  $\geq$  v2022.07 - *Quarto*).

Por defecto *pandoc* produce en algunos casos un fragmento de documento (cuando el formato de salida no es markdown). Para obtener un documento independiente (e.g. un fichero HTML válido incluyendo `<head>` y `<body>`), habrá que emplear la opción `-s` o `--standalone`.

## A.2 Pandoc y RMarkdown

Como ya se comentó, el paquete `rmarkdown` llama a *pandoc* para renderizar un documento RMarkdown<sup>1</sup>, y esta llamada se muestra en la consola (o en la correspondiente pestaña de RStudio):

```
"C:/Program Files/RStudio/bin/pandoc/pandoc" +RTS -K512m -RTS Informes.utf8.md --to html4
--from markdown+autolink_bare_uris+ascii_identifiers+tex_math_single_backslash
--output Informes.html --smart --email-obfuscation none --self-contained --standalone
--section-divs --table-of-contents --toc-depth 3 --variable toc_float=1
--variable toc_selectors=h1,h2,h3 --variable toc_collapsed=1 --variable toc_smooth_scroll=1
--variable toc_print=1 --template "C:\PROGRA~1\R\R-35~1.1\library\RMARKD~1\rmd\h\DEFAULT~1.HTM"
--no-highlight --variable highlightjs=1 --variable "theme:bootstrap" --include-in-header
"C:\Users\RUBEN~1\FCA\AppData\Local\Temp\RtmpkntXD8\rmarkdown-str2084caf51da.html" --mathjax
--variable "mathjax-url:https://mathjax.rstudio.com/latest/MathJax.js?config=TeX-AMS-MML_HTMLorMML"
```

Output created: Informes.html

La mayoría de los campos de la cabecera YAML de un fichero RMarkdown se traducen en las opciones de Pandoc. Por ejemplo, la cabecera:

```

output:
 html_document:
 number_sections: yes

```

(que produce la numeración de secciones y subsecciones), se corresponde con la opción `--number-sections` de *pandoc*. También se puede establecer cualquier opción de Pandoc en la cabecera YAML mediante el campo `pandoc_args`, por ejemplo:

```

output:
 html_document:
 pandoc_args: ["--number-offset", "4,0", "--number-sections"]

```

(en este caso la numeración comenzaría en 4).

---

<sup>1</sup>Desde la versión 2, antes se utilizaba `knitr` y `markdown`.

# Bibliografía

Xie, Y. (2021). *knitr: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.33.

Xie, Y., Allaire, J., and Golemund, G. (2018). *R Markdown: The Definitive Guide*. CRC Press.