

# CACCS: Secuencia de taller de RStudio – Parte 1

Rashid C.J. Marcano Rivera

4 de oct. de 2024

## Contents

<b>Sobre aprender R</b>	<b>2</b>
Usando RStudio (y presentando por encima, RMarkdown) . . . . .	3
<b>Interacción con la consola de R</b>	<b>5</b>
Asignando valores a objetos . . . . .	6
Funciones . . . . .	8
<b>Importando datos</b>	<b>9</b>
Tipos de datos . . . . .	11
El operador de acceso \$ . . . . .	12
<b>Entendiendo y manipulando datos: Tidyverse</b>	<b>18</b>
Modificando conjuntos de datos: creando una variable . . . . .	20
El operador pipe %>% o  > . . . . .	21
Resumiendo datos explorativamente con Tidy . . . . .	22
<b>Recapitulando</b>	<b>24</b>
Qué aprendimos en este taller inicial . . . . .	24
Continuamos el próximo viernes . . . . .	24

## Introducción a la computación estadística en R

Este taller está basado en notas de Paul Thibodeau y revisiones de profesores del departamento de psicología de BYU (vea esa versión [aquí](#)). Adaptado al español y usando ejemplos del libro de Rafael Irizarry disponible [aquí](#).

*Si aún no has instalado R, está aquí. Acto seguido, baja RStudio. Puedes también ir a la nube en Posit Cloud.*

## Sobre aprender R

R es un lenguaje creado por estadísticos como ambiente interactivo para análisis de datos. En R pueden guardar su trabajo como una secuencia de comandos, conocida como un *script*, que se pueden ejecutar fácilmente en cualquier momento, con portabilidad. Estos *scripts* sirven como un registro del análisis que realizaron, una característica clave que facilita el trabajo *reproducible*. Si bien es un programa poderoso y flexible, es ciertamente más complicado que programados como el que puedan encontrar en SPSS o STATA, donde pueden señalar con el ratón una opción y ejecutarla. Por otro lado, R es gratuito y de código abierto. En adición es modular y las funcionalidades añadidas complementariamente por terceros también son gratuitos, incluyendo acceso temprano a los métodos y herramientas más recientes que se desarrollan para una amplia variedad de disciplinas, incluyendo la economía, ciencias políticas, sociología, planificación, ecología, geografía o la biología molecular, entre otros.

Mi idea tras esta secuencia de talleres sobre R es brindar una introducción para que puedan interactuar con R tal que los sentimientos de frustración o ansiedad que puedan surgir al interactuar con este lenguaje al usarlo en variedad de circunstancias. Este taller presume en general ciertos conocimientos básicos en estadística, pero abundará en cada parte que se trabaje aunque sea algo como la motivación para las funciones que ejecutaremos. So no tienen experiencia en programación computacional, es posible que partes de este taller resulten confusas o no tengan mucho sentido. Recomendando que aún así traten de entender lo que puedan, y luego al volver a este taller periódicamente al repasar el contenido, vayan comprendiendo (o trayendo preguntas que no se resuelvan) el material.

Es importante que sepan que la habilidad más útil de antemano es la de saber adónde ir cuando se atasquen. R ayuda bastante en esto. La consola de R, al escribir y ejecutar comandos, brinda retroalimentación inmediata. Hagan uso de esto de manera liberal mientras trabajan el taller, haciendo pequeñas modificaciones a los ejemplos que brinde, hasta que sientan que entienden qué está ocurriendo.

De una vez, cabe señalar que la consola brinda acceso a la función de buscar ayuda interna. Casi todas las funciones (y muchos de los datos pre-almacenados) tienen archivos que les acompañan describiendo qué son las funciones, y cómo usarlas. Pueden acceder a esta información escribiendo `help(función)` (o `?función`), sustituyendo «función» por el nombre de la función que quieras conocer. Es importante leer estos archivos detenidamente la primera vez que te encuentres con una función, pero *también* (posiblemente más) importante es consultarlos con frecuencia. Si tienes una idea de qué quieres hacer, pero no sabes ni recuerdas la función exacta para esto, puedes buscar en los archivos por el término usando doble signo de interrogación (`??regression`).

Hay mucha información adicional en internet. Es difícil buscar en google por R, ya que es una letra, pero sí se puede encontrar mucho sobre funcionalidades. Para búsqueda más certera se puede ir a StackOverflow, y buscar el tag de R (`[R]`) junto a tu pregunta o error. Relacionado está el StackExchange de estadísticas, que es como Stack Overflow pero enfocado en la madeja estadística.

Finalmente, antes de empezar, quiero hablar sobre errores. Tanto novatos como expertos encontrarán errores en su código de R. De suceder, al ir ejecutando un *script* o trabajo, cesará el proceso y saldrá impreso un mensaje de error en la consola. Esto puede ser frustrante, en especial al estar iniciando el aprendizaje, pues el error ocurre a menudo muy adentro de las especificidades de una función, y el mensaje de error no guarda relación con lo que el usuario quería hacer. Un error a evitar al empezar a aprender R es que el error es un disparate incomprensible, y resignarse a la frustración y desánimo. Resistan ese impulso; si bien el mensaje no será de inicio informativo, está diseñado para transmitir cierta información con claridad, y entender esa información es clave para resolver el problema (o cambiar de estrategias).

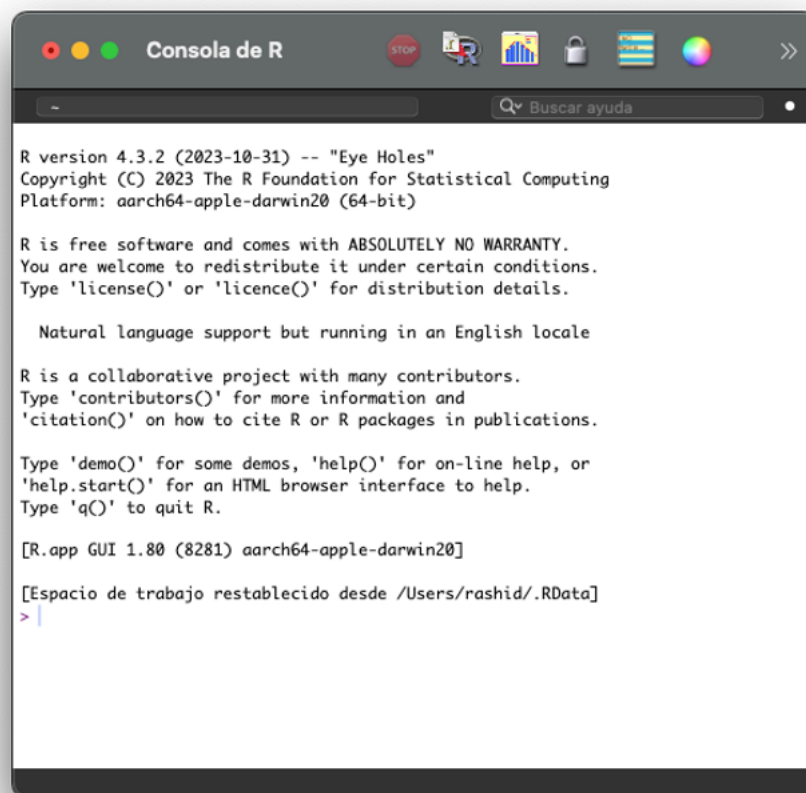
Esta es una secuencia de tres semanas, y en esta lección arrancaremos con lo básico de R, RStudio, y tenemos la meta hoy de que al culminar las primeras dos horas de esta secuencia podamos:

1. Entender la lógica de RStudio.
2. Importar, crear, leer y manipular datos y objetos.
3. Inicios de la visualización y análisis descriptivo.

Durante las próximas semanas abundaremos en esta base, y profundizaremos el viernes 11 de octubre que viene en visualización avanzada con datos censales así como de otros acercamientos, usando herramientas de `ggplot2`. Finalmente, la tercera semana, del 18 de octubre, entraremos más en *wrangling* de datos, así como en la inferencia estadística, con introducciones en R sobre modelos lineales, jerárquicos y longitudinales. Si bien hay mucho más que se puede hacer con R (e.g. *machine learning*, extracción de datos de la web, procesamiento de cadenas y minería de textos, interacción con Git, trabajos en Unix, o nuevos desarrollos en Quarto), la idea es iniciar la travesía en R, y señalar a dónde pueden buscar ahondar.

## Usando RStudio (y presentando por encima, RMarkdown)

¿Cómo difieren R y RStudio? RStudio es una interfaz que yace encima del programa base R. Es además mucho más amigable y llevadero que R.



```
R version 4.3.2 (2023-10-31) -- "Eye Holes"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: aarch64-apple-darwin20 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[R.app GUI 1.80 (8281) aarch64-apple-darwin20]

[Espacio de trabajo restablecido desde /Users/rashid/.RData]
> |
```

Figure 1: Consola R: ejecuta comandos a la medida que se escriben.

R, por su cuenta es una consola sencilla donde teclean y corren código.

RStudio tiene la *Consola* como panel, pero tiene otros paneles en adición que nos ayudan a ubicarnos, como *Environment* (Ambiente) - para que vean los objetos (conjuntos de datos y variables, o funciones) que hayan guardado o creado a través de la sesión - el de *Output* (Salida) - donde pueden ver sus archivos en el directorio, sus gráficos, acceder los paquetes y buscar ayuda - y *Source* (Fuente), donde pueden crear y editar tanto archivos o *scripts* de R, como de Markdown, Quarto y Shiny (para dashboards), entre otros formatos.

Estos paneles (y otras configuraciones) son editables, como el caso del R que manejo en mi Mac:

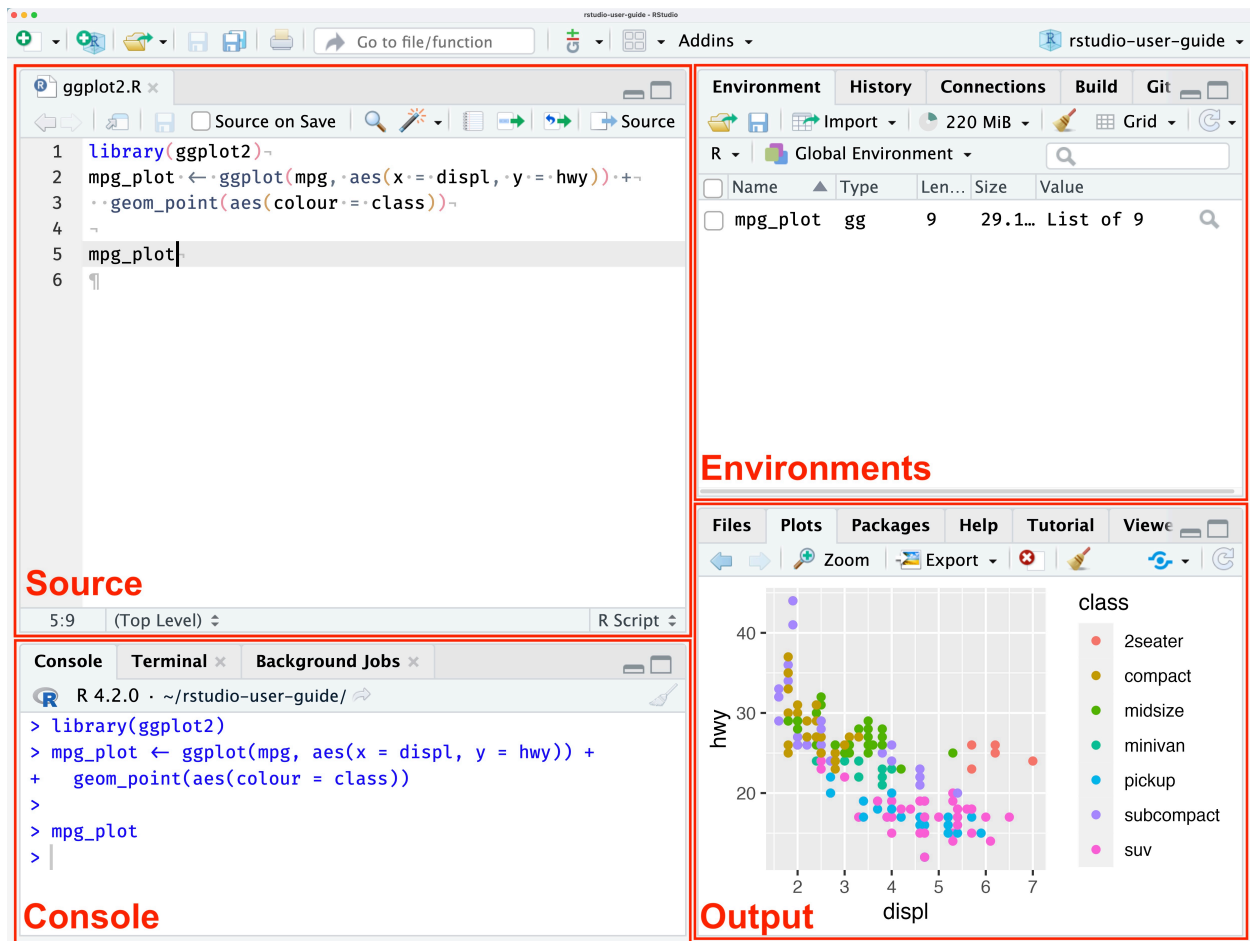


Figure 2: RStudio con varios paneles, incluyendo la consola inicial de R (abajo a la izquierda).

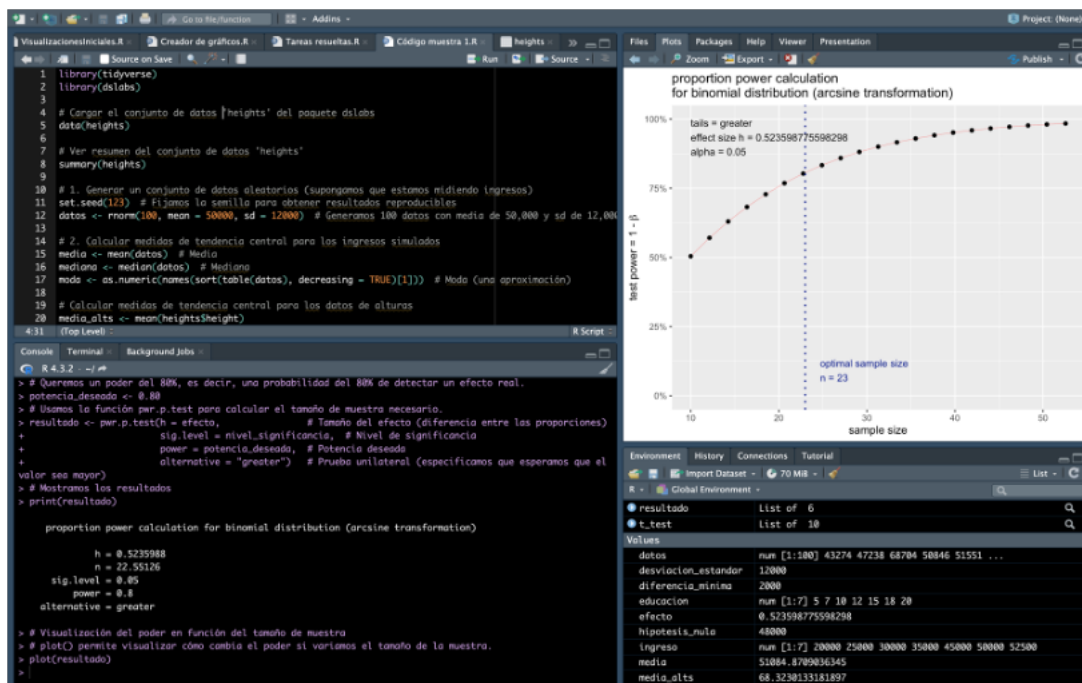


Figure 3: Otra forma posible de configuración.

A grosso modo, RMarkdown es un tipo de documento que permite la escritura de texto regular junto a secciones o pedazos de R. Esto es útil para explicar por ejemplo tu código, seguido por el código mismo para uso regular. También es útil porque Markdown se puede exportar como una página html así como un PDF, lo cual es útil para compartirlo fuera del esquema de R. Si les interesara más sobre RMarkdown, visiten esta página.

## Interacción con la consola de R

```
#install.packages("dslabs","wooldridge","tidyverse") #Si ya están instalados, comenten con '#' para des
```

En la forma básica, R es una calculadora para cálculos básicos. Escribe en R (o en un trozo o **chunk** en RMarkdown) y el resultado saldrá en la consola al ejecutar o correrlo. Un **chunk** se designa en Markdown con triple ``` al inicio y fin, así como `{r título de sección}`. Ahí va el código. Hoy nos enfocaremos en R, pero esta presentación se hizo en RMarkdown. Pueden correr líneas de su secuencia de códigos al usar Command+Return en Mac, o Control+Enter en la PC. También hay un botón para correr líneas o secciones enteras, así como el *script* entero.

```
1 + 2
```

```
## [1] 3
```

```
13 / 2
```

```
## [1] 6.5
```

```
2 ^ 6
```

```
## [1] 64
```

```
5 * (2 + 3)
```

```
## [1] 25
```

```
sqrt(81)
```

```
## [1] 9
```

## Asignando valores a objetos

Claro, R es mucho más que una calculadora básica. La computación con R incluye asignar valores a objetos, y esto se puede hacer con dos maneras básicamente equivalentes:

```
x = 4  
x <- 4
```

En ambos casos, `x` representará 4 y R guardará ese significado para las líneas subsiguientes, a menos que reasignes el valor de `x`.

```
x
```

```
## [1] 4
```

```
x + 2
```

```
## [1] 6
```

```
x = 8  
x
```

```
## [1] 8
```

Vale señalar que no se debe confundir la asignación de valor a un objeto (o variable) como una igualdad. Al pensar lo que vimos, debiéramos pensar que *asignamos 4 a x* o *x recibe 4* o *x tiene 4*, mas no *x es igual a 4*. Aunque `=` es consistente con otros lenguajes de programación, muchos preferimos `<-` al hacer la acción tomada a cabo más evidente. Si tenían curiosidad, se prueba la igualdad con doble signo de igualdad (`==`), y eso produce algo distinto:

```
2 == 2
```

```
## [1] TRUE
```

```
2 == 3
```

```
## [1] FALSE
```

Está bien usar nombres de variables como `x` para ejemplos matemáticos simples como los anteriores. Sin embargo, cuando escribas código para realizar análisis, debes tener cuidado de usar nombres descriptivos. El código donde las cosas se llaman `id_sujeto`, `condición` o `edad` serán más largos que `x`, `y` y `z`, pero tendrán **mucho** más sentido al volver a ellos meses después al hacer sus artículos o trabajos de investigación. Hay eso sí, ciertas reglas para nombres: pueden ser cualquier carácter alfanumérico, pero el primer carácter ha de ser una letra. No se permiten espacios: la computadora no entiende que quieres trabajar con dos palabras como si fuera un concepto; son dos términos separados. En ese caso considera unir palabras con `_` y `..`. R también puede trabajar con datos categóricos, no sólo numéricos:

```
y<-"Puerto Rico"  
y
```

```
## [1] "Puerto Rico"
```

Noten las comillas. ¿Qué pasa si no llevara comillas?

También podemos asignar valores lógicos como cierto, TRUE y falso, FALSE:

```
alive <- TRUE  
asleep <- FALSE
```

Pueden comparar también valores numéricos con `>`, `<`, `!=`, `<=` y `>=`, que devolverán valores lógicos TRUE o FALSE.

```
2 < 3
```

```
## [1] TRUE
```

```
3 <= 3
```

```
## [1] TRUE
```

```
3 != 4 #-> note: the sign "!=" means "not equal to"
```

```
## [1] TRUE
```

Una vez hayan asignado valor numérico a objetos o variables, podrán hacer cálculos:

```
psic <- 2  
soci <- 3  
econ <- 5  
cipo <- 5  
antr <- 3  
geog <- 2  
otro <- 4  
  
taller_n = psic + soci + econ + cipo + antr + geog + otro  
print(paste0("La cantidad de participantes en el taller hoy es ",taller_n)) # cantidad de participantes
```

```
## [1] "La cantidad de participantes en el taller hoy es 24"
```

## Funciones

Las **funciones** en R son útiles para operaciones complejas. Toman en sí insumos de *argumentos* o *parámetros*, hacen su operación y devuelven unas salidas o resultados. Ustedes *llaman* a la función al escribir el nombre seguido de paréntesis, con los argumentos necesarios. Por ejemplo `print()` y `paste0` arriba. También podemos crear funciones propias:

```
mediana_mín_máx <- function(x){  
  qs <- quantile(x, c(0.5, 0, 1))  
  data.frame(mediana = qs[1], mín = qs[2], máx = qs[3])  
}
```

Hay varias funciones en R básico que son útiles en matemáticas:

```
abs(-4)
```

```
## [1] 4
```

```
sqrt(64)
```

```
## [1] 8
```

```
log(1.75)
```

```
## [1] 0.5596158
```

Normalmente usaremos `c()`, la función de concatenación. Esta toma una secuencia de argumentos y la encadena en un **vector**. La mayoría de las funciones de estadísticas descriptivas esperan recibir al menos un vector, o algo similar a ello.

```
a <- c(2, 5, 7)  
print(a)
```

```
## [1] 2 5 7
```

```
cat(a) #concatena e imprime, menos complejo que print al no dar line feeds a menos que se explicita fil
```

```
## 2 5 7
```

```
sum(a) #suma
```

```
## [1] 14
```

```
mean(a) #media
```

```
## [1] 4.666667
```



```
sd(a) #desviación estándar
```

```
## [1] 2.516611
```

## Importando datos

Al recolectar datos en nuestros estudios, o al recibir datos pre-existentes de archivos estadísticos, es probable que nos encontremos con un archivo .csv, donde cada fila tenga la respuesta de un participante o un país, y una columna represente una variable, concepto o pregunta. Queremos importar esto a R para manipular estos datos (renombrarlos, crear nuevas variables de las existentes, fusionar conjuntos de datos que tengan puntos en común) y generar posiblemente estadísticas que resuman la información, así como analizar y visualizar las tendencias halladas.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
## v dplyr      1.1.4      v readr      2.1.5
## v forcats    1.0.0      v stringr   1.5.1
## v ggplot2    3.5.1      v tibble    3.2.1
## v lubridate  1.9.3      v tidyr     1.3.1
## v purrr      1.0.2
## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
# escribiendo un csv
write_csv(mtcars, "mtcars.csv")
```

```
# load a csv file
d <- read_csv("mtcars.csv")
```

```
## Rows: 32 Columns: 11
## -- Column specification -----
## Delimiter: ","
## dbl (11): mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear, carb
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Esta importación es sencilla también con Stata o SPSS usando el paquete **haven**:

```
library(haven)
```

```
haven::write_dta(mtcars, "mtcars.dta")
d2 <- haven::read_dta("mtcars.dta")
```

```
haven::write_sav(mtcars, "mtcars.sav")
d3 <- haven::read_sav("mtcars.sav")
```

Una vez hayan importado los datos es probable que quieran echarle un vistazo a los datos, asegurarse que todo entró adecuadamente:

```
#mira los nombres de las variables
names(d)
```

```
## [1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear"
## [11] "carb"
```

```
#recoge los datos básicos de las variables en el conjunto (e.g. observaciones, datos ausentes, mínimo,
summary(d)
```

```
##      mpg      cyl      disp      hp
##  Min.   :10.40  Min.   :4.000  Min.   : 71.1  Min.   : 52.0
## 1st Qu.:15.43  1st Qu.:4.000  1st Qu.:120.8  1st Qu.: 96.5
## Median :19.20  Median :6.000  Median :196.3  Median :123.0
## Mean   :20.09  Mean   :6.188  Mean   :230.7  Mean   :146.7
## 3rd Qu.:22.80  3rd Qu.:8.000  3rd Qu.:326.0  3rd Qu.:180.0
## Max.   :33.90  Max.   :8.000  Max.   :472.0  Max.   :335.0
##      drat      wt      qsec      vs
##  Min.   :2.760  Min.   :1.513  Min.   :14.50  Min.   :0.0000
## 1st Qu.:3.080  1st Qu.:2.581  1st Qu.:16.89  1st Qu.:0.0000
## Median :3.695  Median :3.325  Median :17.71  Median :0.0000
## Mean   :3.597  Mean   :3.217  Mean   :17.85  Mean   :0.4375
## 3rd Qu.:3.920  3rd Qu.:3.610  3rd Qu.:18.90  3rd Qu.:1.0000
## Max.   :4.930  Max.   :5.424  Max.   :22.90  Max.   :1.0000
##      am      gear      carb
##  Min.   :0.0000  Min.   :3.000  Min.   :1.000
## 1st Qu.:0.0000  1st Qu.:3.000  1st Qu.:2.000
## Median :0.0000  Median :4.000  Median :2.000
## Mean   :0.4062  Mean   :3.688  Mean   :2.812
## 3rd Qu.:1.0000  3rd Qu.:4.000  3rd Qu.:4.000
## Max.   :1.0000  Max.   :5.000  Max.   :8.000
```

```
#las primeras filas
head(d)
```

```
## # A tibble: 6 x 11
##   mpg   cyl  disp    hp  drat    wt  qsec    vs  am  gear  carb
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  21     6   160   110   3.9   2.62  16.5     0     1     4     4
## 2  21     6   160   110   3.9   2.88  17.0     0     1     4     4
## 3  22.8    4   108    93   3.85   2.32  18.6     1     1     4     1
## 4  21.4    6   258   110   3.08   3.22  19.4     1     0     3     1
## 5  18.7    8   360   175   3.15   3.44  17.0     0     0     3     2
## 6  18.1    6   225   105   2.76   3.46  20.2     1     0     3     1
```

```
#las últimas filas
tail(d)
```

```
## # A tibble: 6 x 11
```

```
##      mpg    cyl  disp    hp  drat    wt   qsec    vs    am  gear  carb
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1   26      4  120.    91  4.43  2.14  16.7    0     1     5     2
## 2  30.4      4  95.1   113  3.77  1.51  16.9    1     1     5     2
## 3  15.8      8 351     264  4.22  3.17  14.5    0     1     5     4
## 4  19.7      6 145     175  3.62  2.77  15.5    0     1     5     6
## 5   15      8 301     335  3.54  3.57  14.6    0     1     5     8
## 6  21.4      4 121     109  4.11  2.78  18.6    1     1     4     2
```

```
#los tipos de variable
str(d)
```

```
## spc_tbl_ [32 x 11] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
## $ mpg : num [1:32] 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
## $ cyl : num [1:32] 6 6 4 6 8 6 8 4 4 6 ...
## $ disp: num [1:32] 160 160 108 258 360 ...
## $ hp : num [1:32] 110 110 93 110 175 105 245 62 95 123 ...
## $ drat: num [1:32] 3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
## $ wt : num [1:32] 2.62 2.88 2.32 3.21 3.44 ...
## $ qsec: num [1:32] 16.5 17 18.6 19.4 17 ...
## $ vs : num [1:32] 0 0 1 1 0 1 0 1 1 1 ...
## $ am : num [1:32] 1 1 1 0 0 0 0 0 0 0 ...
## $ gear: num [1:32] 4 4 4 3 3 3 3 4 4 4 ...
## $ carb: num [1:32] 4 4 1 1 2 1 4 2 2 4 ...
## - attr(*, "spec")=
## .. cols(
## .. mpg = col_double(),
## .. cyl = col_double(),
## .. disp = col_double(),
## .. hp = col_double(),
## .. drat = col_double(),
## .. wt = col_double(),
## .. qsec = col_double(),
## .. vs = col_double(),
## .. am = col_double(),
## .. gear = col_double(),
## .. carb = col_double()
## .. )
## - attr(*, "problems")=<externalptr>
```

Recuerden que para datos pre-existentes en R o paquetes, pueden buscar más información sobre los datos al poner `?[nombredatos]` en la consola. `mtcars` es uno de estos, así que podríamos ir a verificar en la pestaña de ayuda sobre los datos y el significado de estos.

```
help(mtcars)
```

## Tipos de datos

Hay cuatro tipos de datos en R. Sabiendo de ellos podemos entender las limitaciones de análisis de cada uno, y también podrán entender mejor algunos errores que podrían recibir. Estos son:

1. Numéricos (números, enteros, dobles (aceptan los decimales))

2. Caracterers (strings)
3. Lógicos (C/F)
4. Factores (niveles discretos; e.g., categorías)

Si ponemos la función `str(d)` notarán que cada variable tiene una asignatura de tipo. Pueden cambiar el tipo de datos, por ejemplo hacia categórico usando la función `as.factor()`. Ejemplo:

```
str(d)

## spc_tbl_ [32 x 11] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
## $ mpg : num [1:32] 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
## $ cyl : num [1:32] 6 6 4 6 8 6 8 4 4 6 ...
## $ disp: num [1:32] 160 160 108 258 360 ...
## $ hp  : num [1:32] 110 110 93 110 175 105 245 62 95 123 ...
## $ drat: num [1:32] 3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
## $ wt  : num [1:32] 2.62 2.88 2.32 3.21 3.44 ...
## $ qsec: num [1:32] 16.5 17 18.6 19.4 17 ...
## $ vs  : num [1:32] 0 0 1 1 0 1 0 1 1 1 ...
## $ am  : num [1:32] 1 1 1 0 0 0 0 0 0 0 ...
## $ gear: num [1:32] 4 4 4 3 3 3 3 4 4 4 ...
## $ carb: num [1:32] 4 4 1 1 2 1 4 2 2 4 ...
## - attr(*, "spec")=
## .. cols(
## ..   mpg = col_double(),
## ..   cyl = col_double(),
## ..   disp = col_double(),
## ..   hp = col_double(),
## ..   drat = col_double(),
## ..   wt = col_double(),
## ..   qsec = col_double(),
## ..   vs = col_double(),
## ..   am = col_double(),
## ..   gear = col_double(),
## ..   carb = col_double()
## .. )
## - attr(*, "problems")=<externalptr>

d$am <- as.factor(d$am)
str(d$am)
```

```
## Factor w/ 2 levels "0","1": 2 2 2 1 1 1 1 1 1 1 ...
```

En estos datos `am` nos informa si el vehículo usa transmisión automática (0) o manual (1). Estas son categorías en realidad, representadas por una variable dummy, pues es mejor reclasificarlas de ser números continuos a categorías: al usar `as.factor()` le dijimos a R que `am` era un factor. Puedes verificar manualmente el estado o transformación de una variable al usar `str(nombre_de_variable)`.

## El operador de acceso `$`

A menudo queremos no entender todos los datos (pueden ser muchos) sino entender algunas variables en específico. Podemos usar un código que permite acceder a la variable dentro de un objeto: `datos$nombre_var`—el conjunto de datos, un signo de peso o dólar, y el nombre de la variable. Por ejemplo, `d$cyl` es decirle a R “la variable `cyl` dentro del conjunto de datos `d`”. Es importante en este punto especificar el conjunto que queremos acceder, pues tenemos varias opciones.

## Pegándonos de un conjunto de datos

Es posible que quieras trabajar con sólo uno o mayormente uno de estos conjuntos. Ahí podríamos usar la opción de `attach(datos)` mientras operas con esos datos, y luego `detach()` al terminar con ellos. En este caso, le decimos a R que asuma que al llamar la variable, nos referimos a los datos que ‘pegamos’ a la memoria.

*Si bien esto puede ser conveniente, podría llevar a ciertos errores por olvido o por cambios en los datos, debe ser usado sin mucha frecuencia.*

Aquí ‘pegamos’ ese conjunto de datos y hacemos una operación con la variable de automático o manual `am`:

```
attach(d)
```

```
## The following object is masked from package:ggplot2:
##
##      mpg
```

```
am <- as.integer(am)
str(am)
```

```
##  int [1:32] 2 2 2 1 1 1 1 1 1 1 ...
```

```
mean(am)
```

```
## [1] 1.40625
```

```
sd(am)
```

```
## [1] 0.4989909
```

```
range(am)
```

```
## [1] 1 2
```

```
am <- as.factor(am)
str(am)
```

```
##  Factor w/ 2 levels "1","2": 2 2 2 1 1 1 1 1 1 1 ...
```

```
#sd(am) #dará error: no se puede aplicar esto a categórico
```

¡Asegúrense de ‘despegar’ el conjunto de datos al terminar las operaciones que iban a usar!

```
detach(d)
```

## Verificando las variables

Ahora que podemos acceder a las variables, podemos explorarlas con varias funciones existentes en R y paquetes adicionales que podamos usar. También podríamos crear nuestras propias funciones. Abajo unas funciones

```
tapply(d$mpg, d$gear, mean) #aplica la función de promedio a la variable de mpg para verlos según la ca
```

```
##          3          4          5
## 16.10667 24.53333 21.38000
```

```
sapply(d, mean) # promedia cada variable del conjunto en lista o vector
```

```
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
```

```
##      mpg      cyl      disp      hp      drat      wt      qsec
## 20.090625  6.187500 230.721875 146.687500  3.596563  3.217250 17.848750
##      vs      am      gear      carb
##  0.437500      NA  3.687500  2.812500
```

```
summary(d$mpg) #estadísticas resumidas básicas
```

```
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
##    10.40   15.43   19.20   20.09   22.80   33.90
```

```
table(d$carb, d$am) #tabla de frecuencias
```

```
##
##      0 1
##    1 3 4
##    2 6 4
##    3 3 0
##    4 7 3
##    6 0 1
##    8 0 1
```

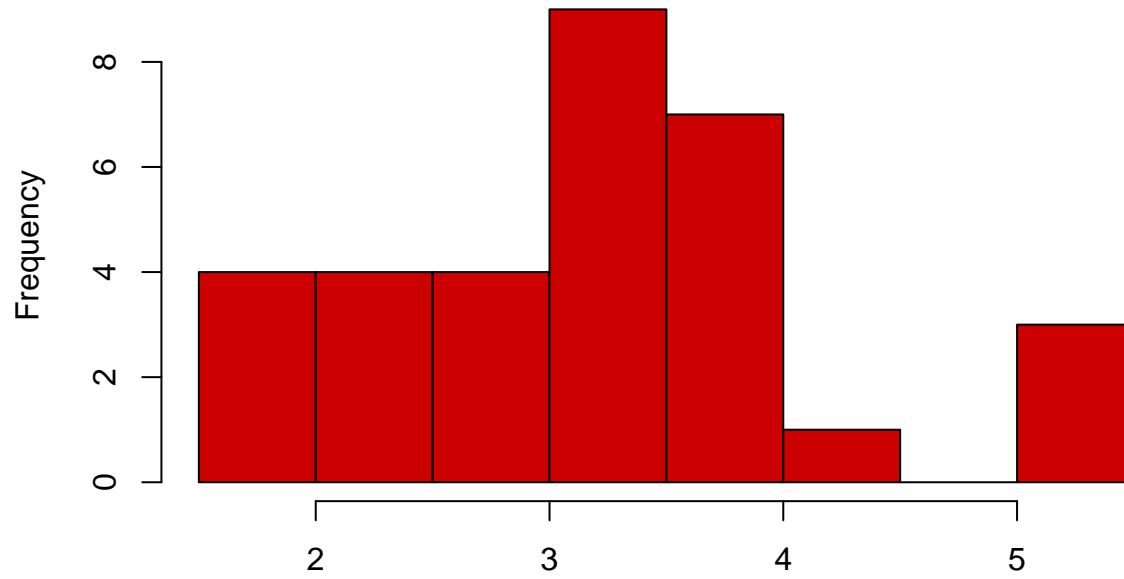
```
addmargins(table(d$carb, d$am, dnn=c('núm. de carburadores','transmisión'))) #añade información adicional
```

```
##
##      transmisión
## núm. de carburadores  0  1 Sum
##                    1  3  4  7
##                    2  6  4 10
##                    3  3  0  3
##                    4  7  3 10
##                    6  0  1  1
##                    8  0  1  1
##                    Sum 19 13 32
```

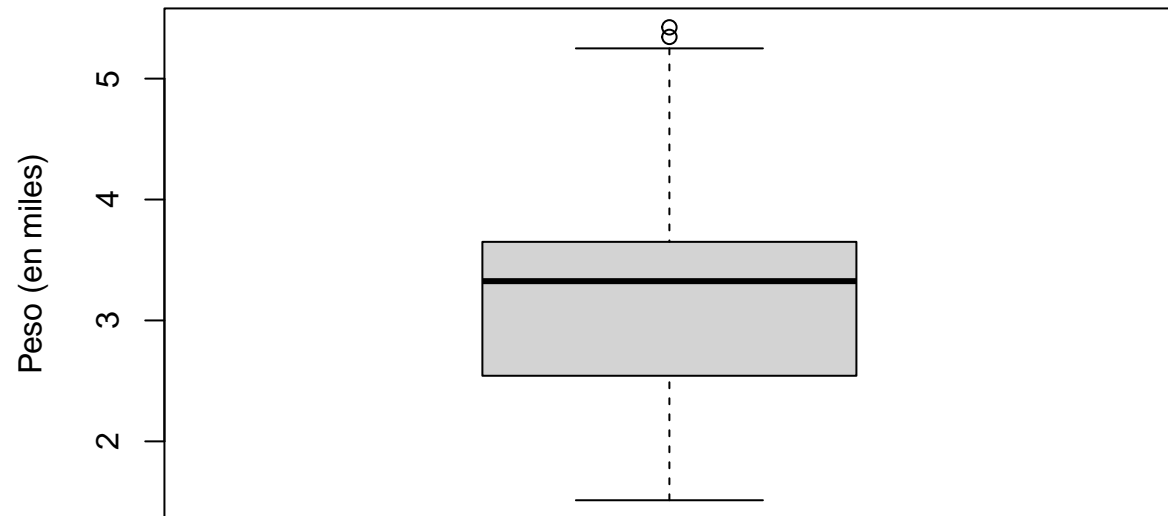
```
# Visualizando la distribución
```

```
hist(d$wt, col = 'red3', xlab = NA, main = 'Distribución del peso')
```

## Distribución del peso

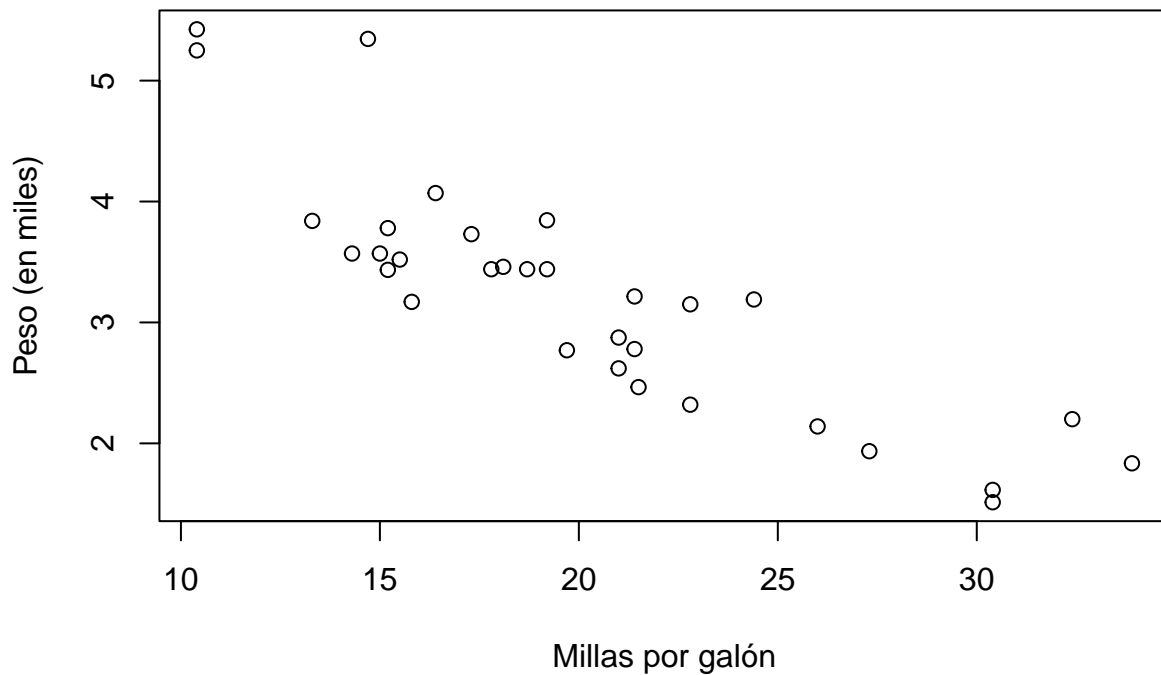


```
boxplot(d$wt, ylab = 'Peso (en miles)')
```



```
plot(d$wt~d$mpg,xlab="Millas por galón",ylab='Peso (en miles)')
```





Nota que para estas funciones lo que se *requiere* en realidad es el nombre de la variable como argumento. ¿Qué creen hacen las opciones `col =`, `xlab =`, `main =`, and `ylab =` mean?

Podríamos también agrupar la información como ocurre en estos casos abajos del paquete `psych`.

```
#install.packages("psych")
library(psych)
```

```
## Warning: package 'psych' was built under R version 4.3.3
```

```
##
## Attaching package: 'psych'
```

```
## The following objects are masked from 'package:ggplot2':
##
##   %+%, alpha
```

```
describe(d$hp) # la función "describe" viene en el paquete de psych; da datos adicionales
```

```
##      vars  n  mean    sd median trimmed  mad min max range skew kurtosis   se
## X1      1 32 146.69 68.56    123  141.19  77.1  52 335   283  0.73    -0.14 12.12
```

```
describeBy(d$mpg, d$am) # del paquete "psych"
```

```
##
## Descriptive statistics by group
## group: 0
##   vars  n mean   sd median trimmed  mad min  max range skew kurtosis   se
## X1     1 19 17.15 3.83   17.3   17.12 3.11 10.4 24.4    14 0.01    -0.8 0.88
## -----
## group: 1
##   vars  n mean   sd median trimmed  mad min  max range skew kurtosis   se
## X1     1 13 24.39 6.17   22.8   24.38 6.67  15 33.9  18.9 0.05    -1.46 1.71
```

Una forma más complicada pero útil de verificar estadísticas es la función `xtabs()`, es decir tabulación cruzada. El código pide una fórmula que contraste las variables. El código es un poco más complicado pero se puede interpretar como “la suma de `mpg`, para cada grupo de `cyl` y `am`”. Ya que `cyl` tiene tres grupos y `am` tiene dos grupos, `xtabs()` devuelve una tabla de seis celdas con las sumas de `mpg` para cada uno de estos grupos; así podríamos por ejemplo verificar como cambian las medias dadas ciertas categorías. Noten que hemos usado, sin `attach()` el nombre de las variables, ya que le dijimos a la función que se enfocara en las variables dentro del objeto definido en la opción `data = conjunto_de_datos` (aquí, `data = d`).

```
xtabs(mpg ~ cyl + am, data = d)
```

```
##      am
## cyl    0     1
##   4 68.7 224.6
##   6 76.5  61.7
##   8 180.6  30.8
```

Si queríamos el promedio de millas por galón por categoría hacemos lo siguiente:

```
# Sumamos mpg por categorías de cyl y am
suma_mpg <- xtabs(mpg ~ cyl + am, data = d)

# Sacamos la cantidad de observaciones por categorías de cyl y am obviando el lado izquierdo de la fórmula
cantidad <- xtabs(~ cyl + am, data = d)

# Media de mpg por categorías de cyl y am
media_mpg <- suma_mpg / cantidad

media_mpg
```

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

En este caso hemos obtenido la media de millas por galón según la cantidad de cilindros del vehículo, y si la transmisión era manual o automática.

## Entendiendo y manipulando datos: Tidyverse

Si bien podríamos usar esta información previa para entender los datos. Digamos que queremos saber qué carro es el mejor en estos datos en cuestión de millas por galón. Podríamos hacer:

```
library(ggplot2)
data("mpg")
summary(mpg)
```

```
## manufacturer      model      displ      year
## Length:234        Length:234    Min.   :1.600  Min.   :1999
## Class :character   Class :character  1st Qu.:2.400  1st Qu.:1999
## Mode  :character   Mode  :character  Median :3.300  Median :2004
##                                     Mean   :3.472  Mean   :2004
##                                     3rd Qu.:4.600  3rd Qu.:2008
##                                     Max.   :7.000  Max.   :2008
##      cyl          trans      drv          cty
## Min.   :4.000    Length:234    Length:234    Min.   : 9.00
## 1st Qu.:4.000    Class :character  Class :character  1st Qu.:14.00
## Median :6.000    Mode  :character  Mode  :character  Median :17.00
## Mean   :5.889                                     Mean   :16.86
## 3rd Qu.:8.000                                     3rd Qu.:19.00
## Max.   :8.000                                     Max.   :35.00
##      hwy          fl          class
## Min.   :12.00    Length:234    Length:234
## 1st Qu.:18.00    Class :character  Class :character
## Median :24.00    Mode  :character  Mode  :character
## Mean   :23.44
## 3rd Qu.:27.00
## Max.   :44.00
```

```
max(mpg$hwy) #sin embargo esto no nos lleva a mucha más información
```

```
## [1] 44
```

```
i_max<-which.max(mpg$hwy)
mpg$model[i_max]
```

```
## [1] "jetta"
```

El vehículo con mejor millaje por galón fue el Jetta. Igualmente si trabajamos datos de criminalidad en EEUU, podríamos tener

```
library(dslabs)
data(murders)
summary(murders)
```

```
##      state      abb      region      population
## Length:51      Length:51    Northeast : 9    Min.   : 563626
## Class :character  Class :character  South     :17    1st Qu.: 1696962
## Mode  :character  Mode  :character  North Central:12  Median : 4339367
##                                     West      :13    Mean   : 6075769
##                                     3rd Qu.: 6636084
##                                     Max.   :37253956
##      total
```

```
## Min.    : 2.0
## 1st Qu.: 24.5
## Median : 97.0
## Mean    : 184.4
## 3rd Qu.: 268.0
## Max.    : 1257.0
```

```
min(murders$total) #queremos igual aquí saber el estado que menos tuvo
```

```
## [1] 2
```

```
i_min<-which.min(murders$total)
murders$state[i_min]
```

```
## [1] "Vermont"
```

Y tendríamos este resultado, con Vermont como el que menos asesinatos de arma de fuego registrara en 2010.

Sin embargo esto no es suficiente quizás para todo lo que queremos hacer y se ve demasiado laborioso o largo en relación a lo que obtenemos. En el caso de los datos de asesinatos por arma de fuego, hemos buscado el estado con menos asesinatos, y vemos que hay un rango enorme entre ése y el mayor, pero ¿estamos comparando chinasy con chinasy?

## Modificando conjuntos de datos: creando una variable

Podríamos crear una tasa de asesinatos para este último conjunto de datos, con la información ya suministrada.

```
murders$tasa_100k<-murders$total/murders$population *10^5
head(murders)
```

```
##      state abb region population total tasa_100k
## 1  Alabama AL  South   4779736   135  2.824424
## 2  Alaska  AK   West    710231    19  2.675186
## 3  Arizona AZ   West   6392017   232  3.629527
## 4  Arkansas AR  South   2915918    93  3.189390
## 5 California CA  West  37253956  1257  3.374138
## 6  Colorado CO   West   5029196    65  1.292453
```

```
#en tidyverse esto se puede hacer con la función mutate()
murders<-mutate(murders,tasa=total/population*10^5)
head(murders)
```

```
##      state abb region population total tasa_100k      tasa
## 1  Alabama AL  South   4779736   135  2.824424  2.824424
## 2  Alaska  AK   West    710231    19  2.675186  2.675186
## 3  Arizona AZ   West   6392017   232  3.629527  3.629527
## 4  Arkansas AR  South   2915918    93  3.189390  3.189390
## 5 California CA  West  37253956  1257  3.374138  3.374138
## 6  Colorado CO   West   5029196    65  1.292453  1.292453
```

Digamos que ahora queremos ver una lista más informativa e intuitiva de los datos; por ejemplo, qué otros carros figuran con buen millaje por galón:

```
filter(mpg, hwy>=35)
```

```
## # A tibble: 8 x 11
##   manufacturer model      displ  year   cyl trans  drv      cty   hwy fl      class
##   <chr>          <chr>    <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
## 1 honda         civic      1.8  2008     4 auto(~ f      25    36 r      subc~
## 2 honda         civic      1.8  2008     4 auto(~ f      24    36 c      subc~
## 3 toyota        corolla    1.8  1999     4 manua~ f      26    35 r      comp~
## 4 toyota        corolla    1.8  2008     4 manua~ f      28    37 r      comp~
## 5 toyota        corolla    1.8  2008     4 auto(~ f      26    35 r      comp~
## 6 volkswagen    jetta      1.9  1999     4 manua~ f      33    44 d      comp~
## 7 volkswagen    new beetle 1.9  1999     4 manua~ f      35    44 d      subc~
## 8 volkswagen    new beetle 1.9  1999     4 auto(~ f      29    41 d      subc~
```

En este caso hemos seleccionado las filas para visualizar que tienen millaje por galón superior a 35: hemos creado un **subconjunto**.

## El operador pipe %>% o |>

Digamos que realmente queremos reducir la cantidad de columnas en las que queremos enfocarnos, pues los datos de mpg tienen demasiadas. Podríamos crear otro subconjunto:

```
tabla_nueva_mpg<-select(mpg,model,year,cty,hwy)
filter(tabla_nueva_mpg, hwy>=35)
```

```
## # A tibble: 8 x 4
##   model      year   cty   hwy
##   <chr>    <int> <int> <int>
## 1 civic      2008    25    36
## 2 civic      2008    24    36
## 3 corolla    1999    26    35
## 4 corolla    2008    28    37
## 5 corolla    2008    26    35
## 6 jetta      1999    33    44
## 7 new beetle 1999    35    44
## 8 new beetle 1999    29    41
```

Y nos quedamos con cuatro columnas dándonos información puntual si esto era lo que nos interesaba. Sin embargo, podríamos evitarnos la creación de objetos intermedios al usar la función que canaliza los datos en secuencia funcional así:

```
mpg %>% select(model,year,cty,hwy) |> filter(hwy>=35)
```

```
## # A tibble: 8 x 4
##   model      year   cty   hwy
##   <chr>    <int> <int> <int>
## 1 civic      2008    25    36
## 2 civic      2008    24    36
```

```
## 3 corolla      1999      26      35
## 4 corolla      2008      28      37
## 5 corolla      2008      26      35
## 6 jetta        1999      33      44
## 7 new beetle   1999      35      44
## 8 new beetle   1999      29      41
```

Vale la pena señalar que el pipe funcionará bien con las funciones donde el primer argumento sean los datos de entrada, que canalizamos. Las funciones de *tidyr* y *dplyr* operan así y se acoplan al pipe.

## Resumiendo datos explorativamente con Tidy

En esta sección cubriremos dos funciones de tidyverse, en dplyr, **summarise** y **group\_by**. La primera, **summarise**, ofrece el cálculo de estadísticas de resumen con un código legible e intuitivo. El segundo agrupa y resume los datos por categorías inherente en las variables existentes. Por ejemplo, quizás quisiéramos calcular el promedio y desviación estándar de los vehículos en los datos de millaje por galón, pero queremos agruparlos por fabricante, o tipo de transmisión, o cualquier otro tipo de variable:

```
mpg %>%
  group_by(manufacturer) |>
  summarise(mpg_grupal=mean(hwy),
            ds_grupal=sd(hwy)) #por fabricante
```

```
## # A tibble: 15 x 3
##   manufacturer mpg_grupal ds_grupal
##   <chr>          <dbl>     <dbl>
## 1 audi           26.4       2.18
## 2 chevrolet      21.9       5.11
## 3 dodge          17.9       3.57
## 4 ford           19.4       3.33
## 5 honda          32.6       2.55
## 6 hyundai        26.9       2.18
## 7 jeep           17.6       3.25
## 8 land rover     16.5       1.73
## 9 lincoln        17         1
## 10 mercury       18         1.15
## 11 nissan         24.6       5.09
## 12 pontiac       26.4       1.14
## 13 subaru        25.6       1.16
## 14 toyota        24.9       6.17
## 15 volkswagen    29.2       5.32
```

```
mpg |>
  group_by(trans) |>
  summarise(mpg_grupal=mean(hwy),
            ds_grupal=sd(hwy)) #por año de manufactura
```

```
## # A tibble: 10 x 3
##   trans      mpg_grupal ds_grupal
##   <chr>          <dbl>     <dbl>
## 1 auto(av)      27.8       2.59
## 2 auto(l3)      27         4.24
```

```
## 3 auto(l4)          22.0      5.64
## 4 auto(l5)          20.7      6.04
## 5 auto(l6)          20       2.37
## 6 auto(s4)          25.7      1.15
## 7 auto(s5)          25.3      6.66
## 8 auto(s6)          25.2      3.99
## 9 manual(m5)        26.3      5.99
## 10 manual(m6)        24.2      5.75
```

Podemos también aplicar una función creada previamente:

```
mpg |> group_by(manufacturer) |> summarise(media_n_mín_máx(hwy))
```

```
## # A tibble: 15 x 4
##   manufacturer media_n mín máx
##   <chr>          <dbl> <dbl> <dbl>
## 1 audi           26     23    31
## 2 chevrolet      23     14    30
## 3 dodge          17     12    24
## 4 ford           18     15    26
## 5 honda          32     29    36
## 6 hyundai        26.5    24    31
## 7 jeep           18.5    12    22
## 8 land rover     16.5    15    18
## 9 lincoln        17     16    18
## 10 mercury        18     17    19
## 11 nissan          26     17    32
## 12 pontiac        26     25    28
## 13 subaru         26     23    27
## 14 toyota         26     15    37
## 15 volkswagen     29     23    44
```

Digamos que queremos obtener las tasas medias de regiones específicas en los EEUU. En este caso entra más complicación pero en Tidyverse podemos usar tanto el pipe `%>%` como la `%in%` (el `%in%` es una función que busca parear la entrada de un valor mapeado en otro):

```
murders %>%
  mutate(group = case_when(
    abb %in% c("ME", "NH", "VT", "MA", "RI", "CT") ~ "Nueva Inglaterra",
    abb %in% c("WA", "OR", "CA") ~ "Costa del Pacífico",
    region == "South" ~ "el Sur",
    TRUE ~ "Otras regiones")) %>%
  group_by(group) %>%
  summarise(tasa_100k = sum(total) / sum(population) * 10^5)
```

```
## # A tibble: 4 x 2
##   group          tasa_100k
##   <chr>          <dbl>
## 1 Costa del Pacífico    2.90
## 2 Nueva Inglaterra     1.72
## 3 Otras regiones       2.71
## 4 el Sur               3.63
```

## Recapitulando

### Qué aprendimos en este taller inicial

Hoy aprendimos varias cosas para empezar a usar R:

- Hablamos sobre cómo descargar e instalar R y RStudio
- Funcionalidades posibles, como Markdown
- Funciones y jerga específica de R
- Cómo cargar datos y visualizarlos
- Cómo buscar estadísticas básicas
- Cómo llevar a cabo operaciones transformadoras a los datos
- Empezamos a usar la sintaxis de tidyverse

### Continuamos el próximo viernes

En los talleres que vienen continuaremos ahondando en operaciones estadísticas, visualización más avanzada, así como estadística inferencial. Gracias por asistir hoy.