R <- world #intro

Enric Escorsa O'Callaghan

1/4/2017

Contents

Chapter 1

Prefacio

> "Trabajando sobre los datos inmediatos de la realidad, nuestra consciencia elabora el universo en el que vivimos realmente"
>
> — Aldous Huxley

Este libro es una introducción muy básica y breve en español al lenguaje de programación estadística **R**. Está dirigido a no programadores (con ánimo de que empiecen a serlo un poquito cuando lo hayan leído).

R permite obtener todo tipo de datos (estructurados, no estructurados, numéricos, textuales, etc.) de dondequiera que estén (bases de datos, tablas, páginas web, etc.) y aplicarles tratamientos y métricas estadísticas, así como generar todo tipo de visualizaciones que nos ayuden a describirlos y entenderlos mejor; pero también automatizar estos procesos, integrando distintas funciones, llegando a generar modelos para, por ejemplo, detectar patrones y asociaciones interesantes que estaban escondidas o hasta lograr predecir que ocurrirá, en base a determinadas tendencias.

Todo ello, se puede hacer en R de una forma ágil, amigable, gratuita, fresca y no encorsetada, en el contexto de una enorme y dinámica comunidad global de usuarios que favorece la colaboración y el aprendizaje continuo, la usabilidad y reproducibilidad de los resultados que se generan y la incorporación de los últimos avances científicos en análisis de datos y aprendizaje automático (*Machine Learning*).

Este libro se escribió usando R -sí, ¡con R también podrás escribir libros!- mediante el método de escritura **Markdown**, que permite ir creando los contenidos sin preocuparte demasiado del formato final y luego exportar fácilmente a los formatos deseados (por ejemplo: PDF, ePub, HTML para publicarlo en una web, etc.)

Si quisieras escribir un libro como este desde R, lo puedes hacer instalándote el paquete **bookdown** (Xie, 2016). Ya hablaremos de qué son los paquetes - *packages* en inglés- y que nos permiten; empecemos.

1.1 Indice de contenidos

El libro se estructura en los siguientes capítulos (si estás leyendo la versión web del libro a la izquierda de la pantalla -o desplegándola desde arriba- verás una barra de navegación que puedes usar para para avanzar en los contenidos):

- Capítulo 1: **Prefacio**
- Capítulo 2: **Introducción a R**
- Capítulo 3: **R para explorar**
- Capítulo 4: **R para predecir**

- Capítulo 5: **R para visualizar**
- Capítulo 6: **R que R**

Chapter 2

Intro

R (R Core Team, 2013) es un lenguaje de programación estadística open source, versátil y potente. Puede descargarse desde la web del proyecto **R-project** https://www.r-project.org.

Una vez instalado uno puede ya empezar a usarlo directamente desde su consola, pero la opción más conveniente para trabajar con R es instalarse acto seguido el Interfaz o entorno de desarrollo integrado (IDE) desarrollado por **RStudio** https://www.rstudio.com (también gratuito), que facilita mucho trabajar con distintos archivos, descargarse paquetes para facilitar la realización de todo tipo de funciones, generar visualizaciones y tenerlo todo ordenado y a mano.

2.1 Manejarse en el entorno de R

¿Te lo descargaste ya? ¡fantástico! ábrelo pues y una vez en la consola puedes escribir líneas de código después del símbolo `>` o *command prompt* y ejecutarlas una a una dándole a `Enter`

Si pruebas a escribir una operación sencilla como `3+4` y lo ejecutas obtienes el resultado.

```
3+4
```

```
## [1] 7
```

¡Bien! También es conveniente usar la **flecha hacia arriba del teclado** para ir a comandos escritos previamente (y así poder recuperar cosas para no tenerlas que volver a escribir)

Luego puedes guardar las líneas de código que hayas escrito en un archivo con extension `.R` para abrirlo en otra ocasión desde el mismo R -o para enviarle a un amig@ y que lo abra en el suyo- y así poder ejecutarlas todas a la vez. Del mismo modo puedes buscar en internet código que otros hayan escrito y reusarlo. ¿no es mágico?

Desde un archivo puedes poner el cursor delante de una línea y ejecutarla con `ctrl + Enter` o bien seleccionar unas cuantas líneas de código con el cursor y hacer lo mismo `ctrl + Enter` (o desde el menú hacer clic en `Run`)

Con `Ctrl + A` puedes seleccionarlo todo y con `Ctrl + R` ejecutarlo todo.

R distingue mayúsculas y minúsculas (algo que no conviene olvidar).

2.2 Variables

En R puedo crear todo tipo de **variables** asignando algo a un nombre mediante `<-`.

Por ejemplo puedo crear una variable numérica escribiendo `x <- 1` u otro tipo de variable que puede ser, por ejemplo de caracteres textuales (*strings*) escribiendo `z <- "pepito"` Nótese que en este caso la pongo entrecomillada. Nótese, asimismo, que el nombre de las variables que creo lo asigno yo (les he llamado `X` y `Z` como podría haberles llamado `numerito` o `palabrita`).

Ojo: los hay, no tan puristas con la gramática de R, que usan para asignar objetos `=` en vez de `<-`. En ambos casos funciona igual.

2.3 Vectores

Un **vector** es un objeto de un solo tipo de datos (puede ser númerico o de caracteres o de otras variables categóricas tales como factores, pero todos los elementos deben ser del mismo tipo)

Puedo crear un vector usando `c()` (combine)

```
c(1,2)
```

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

O también puedo crear un vector de números entre 10 y 20 usando `:`

```
c(10:20)
```

```
##  [1] 10 11 12 13 14 15 16 17 18 19 20
```

O un vector de nombres:

```
c("pepito","grillo", "ternera")
```

```
## [1] "pepito"  "grillo"  "ternera"
```

Como he hecho antes, puedo asignar un vector a una variable `x <- (213:221)` y preguntar por sus propiedades: por ejemplo ¿qué clase de vector es? con la función `class()`

```
x <- (213:221)
class(x)
```

```
## [1] "integer"
```

`X` es un vector de números enteros (*integers*).

Nota: en R es importante tener en cuenta qué tipo de objetos manejamos, porque según qué tipo tenemos podemos hacer unas cosas u otras. Lo iremos viendo. Sigamos.

```
#longitud (número de elementos que contiene)
length(x)
```

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

Contiene 9 elementos.

Nota: La almohadilla `#` se usa para comentar cosas que no queremos que se ejecuten en el código (todo lo que pongamos después de ella no se ejecutará). Comentar nuestros scripts de código es útil para que otros -¡incluidos nosotros mismos en el futuro!- entiendan al leerlo qué pretendíamos hacer.

Va, pongamos otra cita aquí; esta que me gusta:

> "Cualquier tonto puede escribir código que un ordenador entienda. Sólo los buenos programadores escriben código que los humanos entienden."
>
> — Martin Fowler

Sigamos preguntando cosas a una variable; por ejemplo si es numérica:

```
#¿es numérico?
is.numeric(x)
```

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

Verdadero, sí es numérico.

La expresión `x[2]` me dará el segundo elemento del vector `x`

```
## [1] 214
```

Incluso puedo crear una variable nueva (`y`) que tenga los elementos del cuarto al octavo de `x`:

```
y <- x [4:8]
y
```

```
## [1] 216 217 218 219 220
```

2.4 Tablas

También puedo crear tablas mediante `data.frame` y luego ir listando las columnas de la tabla separadas por comas. Una columna sería: `nombre_variable=c(caso1, caso2, etc.)`. Una tabla con varias columnas se armaría del siguiente modo:

```
df<- data.frame( ID=1:4,
     Nombre=c("Pepito","Juanito","Carlitos","Pedrito"),
     Cromos=c(35,15,3,21),
     Edad=c(12,13,9,10),
     Deporte=c("Waterpolo","Futbol","Waterpolo","Petanca"))
df
```

```
##   ID   Nombre Cromos Edad   Deporte
## 1  1   Pepito     35   12 Waterpolo
## 2  2  Juanito     15   13    Futbol
## 3  3 Carlitos      3    9 Waterpolo
## 4  4  Pedrito     21   10   Petanca
```

Entonces puedo preguntar a una tabla:

```
# muéstrame la primera fila
df[1,]
```

```
##   ID Nombre Cromos Edad   Deporte
## 1  1 Pepito     35   12 Waterpolo
```

```
# muéstrame la tercera columna
df[,3]
```

```
## [1] 35 15  3 21
```

que también puedo expresar como `df[[3]]`.

Para ver una columna de una tabla también puedo usar el nombre de la tabla, seguido del símbolo del dólar `$`, seguido del nombre de la columna que me interesa:

```
# muéstrame la columna Edad
df$Edad
```

```
## [1] 12 13  9 10
```

Ahora quiero fijarme sólo en los cromos y la edad de Pepito y Juanito:

```
# filas 1 y 2, columnas 3 y 4
df[1:2,3:4]
```

```
##   Cromos Edad
## 1     35   12
## 2     15   13
```

O si sólo me interesan unas filas determinadas:

```
# las filas 1 y 3`
df[c(1,3),]
```

```
##   ID   Nombre Cromos Edad   Deporte
## 1  1   Pepito     35   12 Waterpolo
## 3  3 Carlitos      3    9 Waterpolo
```

Estas opciones de filtrado son útiles porque me permiten por ejemplo crear una nueva tabla limpia -llamémosle `df2`- con sólo las filas que quiero de `df` (lo haria con la expresión `df2 <- df[c(1,3),]`).

También puedo usar operadores:

- `==` igual que;
- `>=` igual o mayor que;
- `>` mayor que;
- `&` AND;
- `|` OR;
- `!` NOT.

A ver ¿quién tiene más de 20 cromos?

```
df[df$Cromos>20,2]
```

```
## [1] Pepito  Pedrito
## Levels: Carlitos Juanito Pedrito Pepito
```

(nótese que abajo del resultado me indica que estos nombres son considerados como 4 valores distintos o niveles (`levels`); se trata, por tanto, de variables categóricas, también llamadas `Factores`)

Vemos pues que los **data frames** son tablas formadas por vectores y/o factores de la misma longitud. Tan sólo mencionar, por ahora, que en R, existen otros tipos de tablas; por ejemplo **arrays** (tablas con K dimensiones) o **matrices** (array de 2 dimensiones). Otros objetos interesantes son las **listas** (que pueden contener cualquier objeto, por ejemplo, varios vectores diferentes). Dejemoslo aquí de momento.

2.5 Funciones

Hemos visto cómo usar funciones tales como por ejemplo `length()` para calcular el número de elementos que contiene una variable.

Si quisiera cambiar el tipo de variable usaria funciones como `as.numeric()` para cambiar a variable numérica o `as.character()` para cambiar a variable de carácteres.

Como vemos, una función se expresa con el nombre de la función seguida de los datos y las variables dónde queremos aplicar la función (los `argumentos`) que ponemos dentro de un paréntesis.

Otras funciones útiles son:

`nrow(df)` me dice el numero de filas de la tabla df

`ncol(df)` me dice el numero de columnas de la tabla df

Alternativamente, puedo usar la función `dim()` para preguntar por ambas dimensiones (filas y columnas):

```
dim(df)
```

```
## [1] 4 5
```

Con `dim` nos hemos informado de que la tabla `df` tiene 4 filas y 5 columnas.

`summary(df)` me sirve para obtener un descripción de los principales parámetros estadisticos de `df`:

```
summary(df)
```

```
##        ID            Nombre      Cromos          Edad            Deporte
##  Min.   :1.00   Carlitos:1   Min.   : 3.0   Min.   : 9.00   Futbol   :1
##  1st Qu.:1.75   Juanito :1   1st Qu.:12.0   1st Qu.: 9.75   Petanca  :1
##  Median :2.50   Pedrito :1   Median :18.0   Median :11.00   Waterpolo:2
##  Mean   :2.50   Pepito  :1   Mean   :18.5   Mean   :11.00
##  3rd Qu.:3.25                3rd Qu.:24.5   3rd Qu.:12.25
##  Max.   :4.00                Max.   :35.0   Max.   :13.00
```

`head()` me permite ver las primeras 6 filas de una tabla

`tail()` las 6 últimas

Para obtener la mediana de Edad (cuarta columna en este caso) de `df` podriamos hacer:

```
mean(df[,4])
```

```
## [1] 11
```

Para repetir algo un número de veces, uso `rep()`:

```
rep("algo", times=3)
```

```
## [1] "algo" "algo" "algo"
```

Nota: Por defecto R transforma los carácteres de una tabla en factores (categorias) al leerla; esto puede causar cierta confusión porque hay casos, por ejemplo, en que el texto de una columna sí queremos que se entienda como una categoria (p.ej. hombre/mujer, región, tipología, etc.), pero otras veces podemos querer importar el texto como tal sin categorizarlo. Esto lo podemos controlar con el argumento `stringsAsFactors`; si queremos que nuestros valores sigan siendo carácteres y no queremos que se nos cambien a factores, lo debemos indicar específicamente con `stringsAsFactors = False`. (muchos errores en R se deben a no tener claro qué tipo datos tratamos).

2.6 Paquetes

Cuando uno crea en R una serie de funciones para hacer algo, lo puede guardar como un **paquete** para que otros puedan reusarlo. Los conceptos de reutilización y reproducibilidad en un lenguaje de acceso abierto como R son sumamente importantes. Hay infinidad de paquetes disponibles que podemos utilizar para un sinfín de aplicaciones. Ante un problema analítico específico que debamos resolver, es muy probable que exista ya un paquete que resuelva ese mismo problema en el repositorio oficial de R, llamado CRAN o bien en otro repositorio importante de paquetes -más dedicado a tratamientos bioestadísticos- llamado Bioconductor

Para usar un paquete en R, debo primero tenerlo instalado. Lo hago escribiendo:

`install.packages("nombre_del_paquete")`

y luego puedo cargarlo en cada sesión con:

`library(nombre_del_paquete)`

Chapter 3

R para Explorar

> "Aquí tengo todos los datos que demuestran incuestionablemente que los datos no demuestran NADA…."
>
> — Miguel Brieva

La exploración de datos implica normalmente poder leer datos de fuentes y formatos diversos y acomodarlos según nos convenga. Estas tareas son, de hecho, las que suelen tomar más tiempo, de todo el proceso de análisis y asimilación de la información.

3.1 Leer datos

Nos puede interesar leer y analizar datos de varios lados: de un archivo, una página web, una base de datos, etc.

3.1.1 Leer datos de un archivo

Si queremos leer los datos de una tabla, por ejemplo, un archivo en formato `.csv` (valores delimitados por comas) que tenemos en nuestra carpeta o directorio, lo podemos hacer mediante el paquete **readr** (Wickham, 2017b)

```
library(readr)
misdatos<- read_csv("_bookdown_files/cromos.csv")
misdatos
```

```
## # A tibble: 13 x 10
##      Nombre Cromos  Edad   Deporte Poblacion      Pais numerodehermanos
##       <chr>  <int> <int>     <chr>     <chr>     <chr>            <int>
##  1   Pepito     35    12 Waterpolo Barcelona    España                1
##  2  Juanito     15    13    Futbol    Madrid    España                3
##  3 Carlitos      3     9 Waterpolo     Ceuta    España                2
##  4  Pedrito     21    10   Petanca  Mallorca    España                2
##  5  Mariela     40    12    Futbol Barcelona    España                4
##  6   Vicent     21    10    Futbol  Valencia    España                2
##  7    Pablo      0    13   Basquet    Madrid    España                1
##  8   Tanita     33     8 Waterpolo    Murcia    España                0
##  9  Antonio     26     9    Futbol     Porto  Portugal                2
## 10     Marc     31    10   Basquet   Andorra   Andorra                2
```

```
## 11    Marie     11    12    Ballet    Rennes   Francia               3
## 12     Anas     14    10    Futbol     Rabat Marruecos              1
## 13  Alfredo      9    10    Futbol     Quito   Ecuador              4
## # ... with 3 more variables: Juegopreferido <chr>,
## #   Fechadenacimiento <chr>, Sexo <chr>
```

Nota: ojo al detalle del formato usado para indicar la ubicación del archivo: las barras son del tipo `/` (si usamos \ no nos funcionará)

De modo similar, leería los datos de un archivo `.txt` (datos no tabulares) usando la variante de la función de lectura `read_file`en mi expresión: `misdatos <- read_file("path/miarchivo.txt")`

Habrá que fijarse bien y cuando sea necesario usar las distintas variantes de la función de lectura de **readr** según cómo estén delimitados los datos en el archivo que queremos leer: sea por comas, por punto y coma, por separador (tab) o por cualquier otro delimitador.

Mírate esta chuleta de Rstudio para entender todas las opciones de importación de datos: https://github.com/rstudio/cheatsheets/raw/master/data-import.pdf

En caso de que no estén por defecto en la función de lectura que usamos estas opciones las deberemos especificar nosotros en los argumentos de la función; así como si tienen título o no, si queremos considerar los datos faltantes o no, etc. Siempre nos aseguraremos de que los datos se nos importan en el formato correcto.

3.1.2 Leer datos contenidos en un excel

Si me interesan algunos datos que tengo en un excel; los puedo seleccionar, copiar (en el portapapeles) y luego voy a R y escribo: `misdatos <-read.delim ("clipboard")`.

Ahora cuando lo que quiero es leer archivos enteros de excel desde R, usaré -de modo similar a cómo hemos visto para leer archivos csv- paquetes como **xlsx** (Dragulescu, 2015) o **readxl** (Bryan and et al., 2017) (paquete que es parte del conjunto integrado de paquetes **Tidyverse**) y sus respectivas funciones para leer los datos de las tablas de un excel: `read.xlsx ()` y `read_excel()`.

3.1.3 Leer datos de una página web

Pongamos que me interesa una tabla en csv disponible en una página web. La puedo importar también con `misdatos <- read.csv("http://www..../archivo.csv")`

Otro caso seria recuperar una parte de contenido que se encuentra en formato HTML en una página web a un formato estructurado que podamos analizar. La técnica asociada a ese fin se conoce como `webscrapping`. En R hay un paquete específico para hacer webscraping llamado **rvest** (Wickham, 2016)

Por ejemplo, imaginemos que soy un fan del waterpolo y quiero recuperar los resultados de la competición de la categoría de másteres que se encuentran en la página de la federación catalana de natación:

```
#cargo el paquete rvest
library(rvest)
#asigno un nombre a la URL de la página que me interesa
clasificacion_waterpolo <-
  read_html("http://aquatics.cat/competicio/informacioCompeticio/2016/9/49/0")

#recupero la segunda tabla que aparece en esta página indicando ".[[2]]" en la expresión:
clasificacion_waterpolo %>%
  html_nodes("table") %>%
  .[[2]] %>%
  html_table()
```

```
##    CLASSIFICACIÓ   CLASSIFICACIÓ CLASSIFICACIÓ CLASSIFICACIÓ CLASSIFICACIÓ
## 1             NA           Equip             P            PJ            PG
## 2              1 CN L'Hospitalet            32            18            15
## 3              2       CN Rubí B            29            18            14
## 4              3    CN Vallirana            22            18            10
## 5              4        CN Badia            20            18            10
## 6              5 CW Sant Adrià A            18            18             8
## 7              6    Poble Nou CN            17            18             8
## 8              7 CN Granollers B            17            18             6
## 9              8    CN Martorell            11            18             5
## 10             9           GEIEG             9            18             3
## 11            10      CN Sallent             5            18             1
##    CLASSIFICACIÓ CLASSIFICACIÓ CLASSIFICACIÓ CLASSIFICACIÓ
## 1             PE            PP            GF            GC
## 2              2             1           197           124
## 3              1             3           209           133
## 4              2             6           168           160
## 5              0             8           201           183
## 6              2             8           159           149
## 7              1             9           168           170
## 8              5             7           130           135
## 9              1            12           142           183
## 10             3            12           122           175
## 11             3            14           128           212
```

Ya tengo los resultados.

3.2 Acomodar datos

Muy a menudo, sin embargo, los datos se nos presentan de forma bruta o imperfecta (formatos raros, valores que faltan, duplicidades, etc) o simplemente no están de la forma que queremos. Por ello es importante acomodarlos de un modo que sea adecuado o conveniente para su análisis.

3.2.1 Funciones base en R

En R es posible usar funciones como `apply, lapply, sapply, tapply, etc.` para realizar operaciones comunes en el trabajo con matrices y tablas.

- **apply**, permite aplicar funciones a filas o columnas de una matriz

Por ejemplo, aplicaríamos la funcion suma a las columnas de la matriz `df` con la expresión: `apply (df, 2, sum)` (aquí `1` se usa para indicar fila y `2` para indicar columna)

- **lapply**, extrae una parte de una matriz o tabla como una lista

Por ejemplo, para extraer la primera fila de la tabla `df` usariamos: `lapply (df,"[", 1, )` y para extraer la segunda columna de la tabla `df`: `lapply (df,"[", , 2)`

- **tapply**, permite aplicar funciones a una variable, desglosada en base a otra.

Por ejemplo, en el caso de nuestros datos, para calcular la mediana del número de cromos por sexo:

```
tapply(misdatos$Cromos, misdatos$Sexo, mean)
```

```
##    f    m
## 28.0 17.5
```

Las chicas tienen de media más cromos que los chicos.

Otras funciones disponibles en R base (sin necesidad de cargar paquetes adicionales) útiles para ordenar, filtrar o hacer transformaciones a nuestros datos son `sort()`, `subset()` o `transform()`.

Prueba a buscar ayuda sobre estas funciones (por ejemplo escribiendo `?sort`).

3.2.2 Paquete dplyr

Más recientemente, en el marco de herramientas integradas que facilitan acomodar datos (tidyverse), podemos usar varias funciones incluidas en el paquete **dplyr** (Wickham, 2017a). Las más importantes son:

- **filter()** que permite filtrar observaciones por sus valores.
- **arrange()** para reordenar filas por variables
- **select()** para tomar variables por sus nombres.
- **mutate()** para canviar variables o crear nuevas variables con funciones de variables existentes.
- **summarise()** para sumarizar muchos valores en uno solo.

Por ejemplo, para eliminar una columna de una tabla puedo usar `select()` y el operador negativo `-`:

`df %>% select(-variable_inutil)` (siendo `variable_inutil` el nombre de la columna de `df` que no quiero)

Para eliminar filas duplicadas puedo usar la función `distinct()`:

`df_limpia <- df %>% distinct()` elimina filas duplicadas de `df` y asigna una nueva tabla (`df_limpia`) sin ellas (`distinct` és útil por tanto para hacer recuentos de variables descartando las que aparecen más de una vez)

Con `filter()` (función similar a `subset` en R base) puedo por ejemplo retener determinadas observaciones de una columna (p.ej. un determinado rango de la variable `Edad`):

`df %>%  filter(Edad > 18)`

Con `arrange()` (similar a `sort` en R base) ordenar datos en orden ascendente: `arrange(Edad)` o descendente : `arrange(desc(Edad))`

Con `mutate()` (similar a `transform` en R base) puedo, por ejemplo, crear una nueva variable en base a otras:

`mutate(nueva_columna = col1 + col2)`

Nota: vemos que una forma de expresar varias funciones a la vez de un modo eficiente y entendedor es usando el operador `%>%` (conocido como `pipe` y disponible con los paquetes **magritte** y **tidyverse**). Consiste básicamente en que, en vez de expresar una función que opere sobre unos datos como `f(datos)`, lo puedo hacer como `datos %>% f` y ello me permite ir concatenando varias funciones que quiera aplicar sobre esos mismos datos y que sea fácil de entender.

Veámoslo en un ejemplo:

```
library(dplyr)
misdatos %>%
  #hago filtro de los que no son hijos únicos
  filter(numerodehermanos > 1) %>%
  #creo nueva variable resultante de dividir Cromos por Edad
  mutate(RatioCromosEdad = Cromos/Edad) %>%
  #creo nueva variable (Edaden5) sumándole 5 años a la variable Edad
  transform(Edaden5 = Edad+5) %>%
```

```
  #ordeno por numero de cromos en orden descendente
  arrange(desc(Cromos))%>%
  #mostrar las primeras 5 filas de la tabla resultante
  head(5)
```

```
##    Nombre Cromos Edad Deporte Poblacion     Pais numerodehermanos
## 1 Mariela     40   12  Futbol Barcelona   España                4
## 2    Marc     31   10 Basquet   Andorra  Andorra                2
## 3 Antonio     26    9  Futbol     Porto Portugal                2
## 4 Pedrito     21   10 Petanca  Mallorca   España                2
## 5  Vicent     21   10  Futbol  Valencia   España                2
##   Juegopreferido Fechadenacimiento Sexo RatioCromosEdad Edaden5
## 1         Farcry         12-4-2004    f        3.333333      17
## 2     GuitarHero         28-9-2005    m        3.100000      15
## 3      Minecraft         12-7-2007    m        2.888889      14
## 4    ClashRoyale         30-5-2006    m        2.100000      15
## 5      Minecraft          1-9-2005    m        2.100000      15
```

3.2.3 Limpiar textos

Para limpiar textos a menudo nos interesa reemplazar carácteres de nuestros datos brutos por otros más sencillos. Para ello suelen ser muy útiles funciones como **gsub**. La expresamos como:

`gsub(pattern, replacement, x, ignore.case = TRUE)`

Dónde:

- `pattern`: carácteres a cambiar
- `replacement`: carácteres para remplazar
- `x`: nuestros datos
- `ignore.case`: ponemos TRUE cuando queremos ignorar si el patrón es mayúsculas o minúsculas

Por ejemplo:

```
x <- "danger"
gsub("d","",x)
```

```
## [1] "anger"
```

La función **substr** la podemos usar, por ejemplo, para eliminar n carácteres de un elemento. Si sólo queremos los carácteres del primero al onceavo:

```
brutos <- "loquequieroloquenoquiero"
limpios <- substr(brutos, 1, 11)
print(limpios)
```

```
## [1] "loquequiero"
```

Esta misma expresión la podemos opcionalmente formar junto con `nchar` para indicar el número de carácteres que no queremos (empezando en este caso por atrás):

```
brutos <- "loquequieroloquenoquiero"
limpios <- substr(brutos, 1, nchar (brutos)-13)
print(limpios)
```

```
## [1] "loquequiero"
```

Otro paquete **stringr** (Wickham, 2017c) es también útil para manipular textos; de modo similar me permite tomar sólo los últimos carácteres de algo:

```
library (stringr)
brutos <- "smart"
limpios <- str_sub(brutos,-3)
print(limpios)
```

```
## [1] "art"
```

O extrer las palabras unidas por guiones bajos en una frase

```
frase <- "el_último_de_la_fila"
 str_split(frase,"_")
```

```
## [[1]]
## [1] "el"     "último" "de"     "la"     "fila"
```

O considerar los NA que aparecen a menudo en una tabla o un vector de textos como datos faltantes y reemplazarlos por carácteres (entrecomillados) sin eliminarlos.

```
vector <- c("una_cosa",NA,"otra_cosa")
str_replace_na(vector)
```

```
## [1] "una_cosa"  "NA"        "otra_cosa"
```

Nota: Quizás lo estés pensando; pues sí, en R una misma cosa puede hacerse de muchos modos distintos y usando paquetes distintos. Hay algunas funciones comunes y paquetes muy usados -las que estamos tratando de mostrar aquí- pero existen innumerables funciones para resolver problemas similares y uno siempre encontrará un paquete que haga la misma cosa pero de otro modo.

Chapter 4

R para Predecir

> "The problem with experts is that they don't know what they don't know…."
>
> — Nassim Taleb

Con R es también sencillo aplicar modelos de regresión y correlación a nuestros datos, modelos estadísticos predictivos, etc.

4.1 Correlación

No hemos comentado todavía que en el repositorio de R existen varios datasets de ejemplo con los que podemos trabajar y que podemos invocar con sólo llamarlos por su nombre. Uno de ellos es *mtcars*. Se trata de una base de datos de modelos de coches y sus distintas prestaciones o características técnicas expresadas como variables.

Echémosle un vistazo:

```
head(mtcars)
```

```
##                   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
```

Veamos,por ejemplo, con la ayuda de la función **cor()** si existen correlaciones entre las variables *mpg* (miles per galon), *cyl* (cilindrada), etc.:

```
# ponemos las variables mpg, cyl y disp como filas de una matriz:
x <- mtcars[1:3]
# y hp, drat y wt como columnas de la matriz:
y <- mtcars[4:6]
#la funcion cor() nos da la correlacion en cada caso:
cor(x, y)
```

```
##              hp       drat         wt
## mpg  -0.7761684  0.6811719 -0.8676594
## cyl   0.8324475 -0.6999381  0.7824958
## disp  0.7909486 -0.7102139  0.8879799
```

Valores cercanos a 1 indican correlación positiva, cercanos a -1 correlación negativa y cercanos a 0 poca correlación.

4.2 Regresión

Usamos regresión cuando tenemos un atributo `X` y queremos predecir una Respuesta o Variable de Salida `Y`, por ejemplo para saber cuanto mide alguien en función de cuanto pesa.

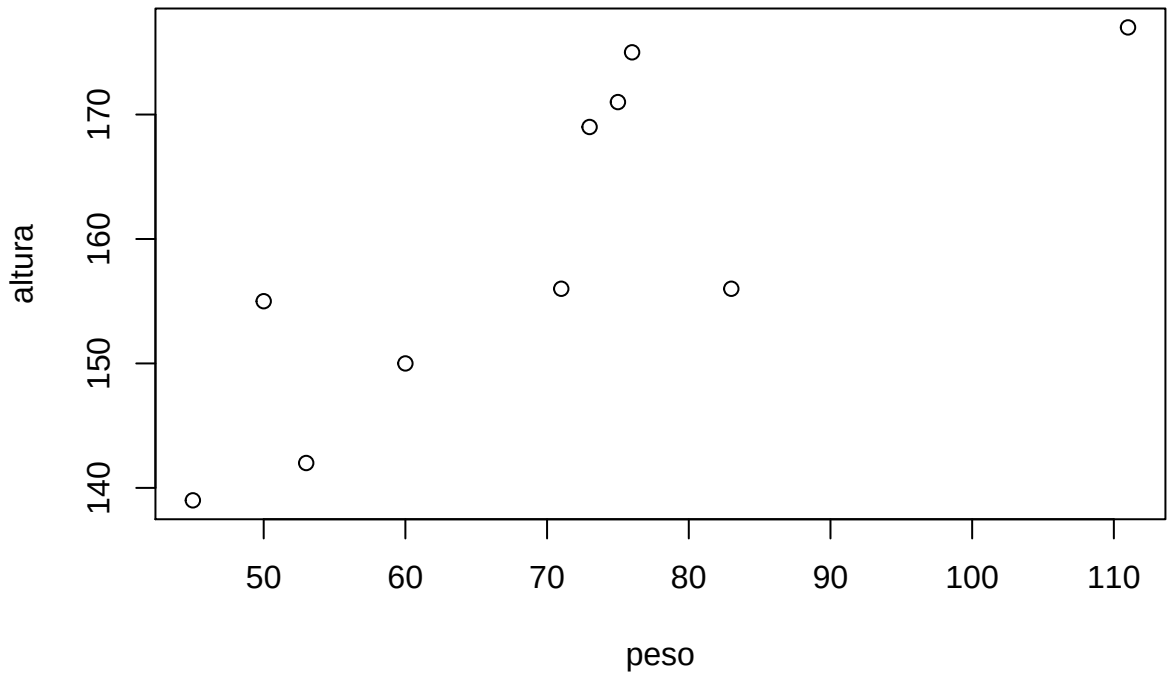
4.2.1 Regresión lineal

Si tenemos sólo una variable o atributo hablamos de Regresión Linear Simple. Cuando tenemos múltiples atributos (ej. x1, x2, x3,..) sería Regresión Linear Múltiple.

Se aplica la fórmula de la ecuación lineal: `y = a + b * x` y se trata de determinar los coeficientes `a` (determina dónde la línea intersecta con el eje Y) y `b` (pendiente de la línea).

Veamos este caso sencillo y tratemos de trazar la línea de regresión en base a alturas y pesos de una población de 10 personas:

```
altura <- c(156, 155, 142, 177, 139, 156, 171, 169, 150, 175)
peso <- c(83, 50, 53, 111, 45, 71, 75, 73, 60, 76)
plot(peso, altura)
```

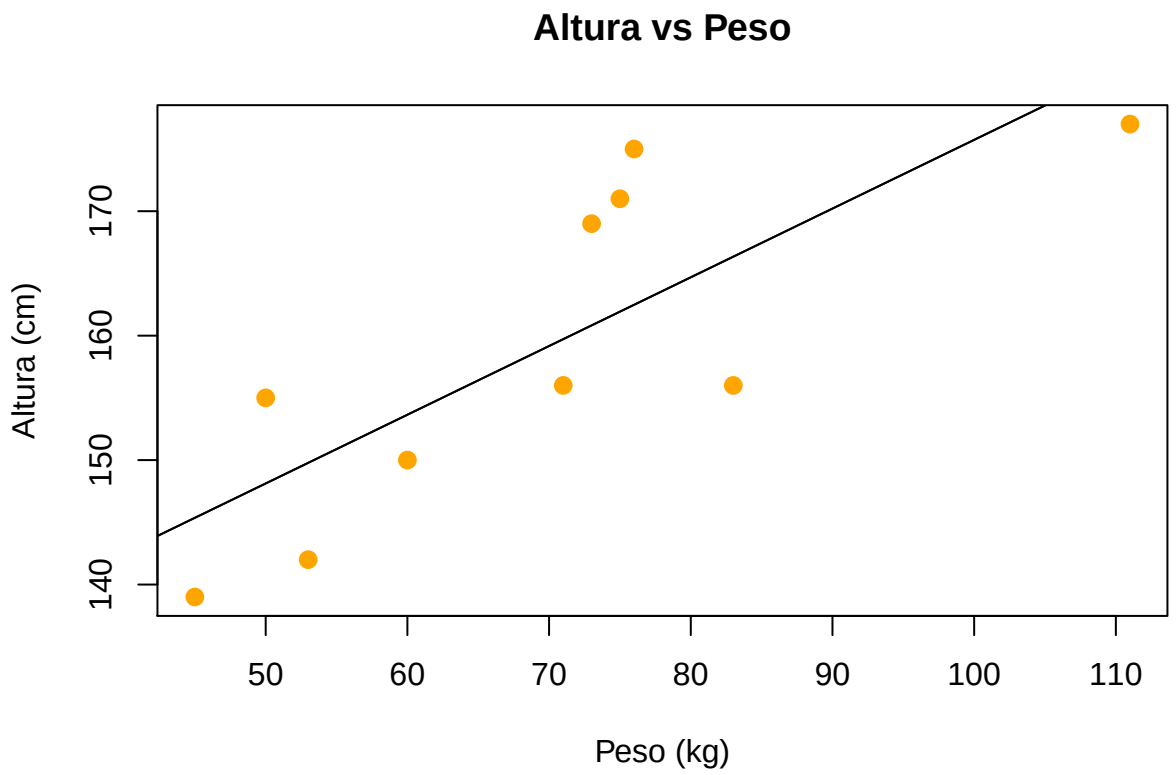



```
plot(peso, altura, pch = 16, cex = 1.3, col = "orange",
     main = "Altura vs Peso",
```

```
     xlab = "Peso (kg)",
     ylab = "Altura (cm)")
#modelo lineal
lm(altura ~ peso)
```

```
##
## Call:
## lm(formula = altura ~ peso)
##
## Coefficients:
## (Intercept)         peso
##    120.5135       0.5522
```

```
#vemos que el intercept es 120.5135 y el pendiente 0.5522.
#Entonces finalmente trazamos la linea que mejor se ajusta (linea de regresión)
#en nuestro plot:
abline(120.5135, 0.5522)
#o sino también podemos visualizar la linea de regresion con:
abline(lm(altura ~ peso))
```



4.2.2 Regresión logística

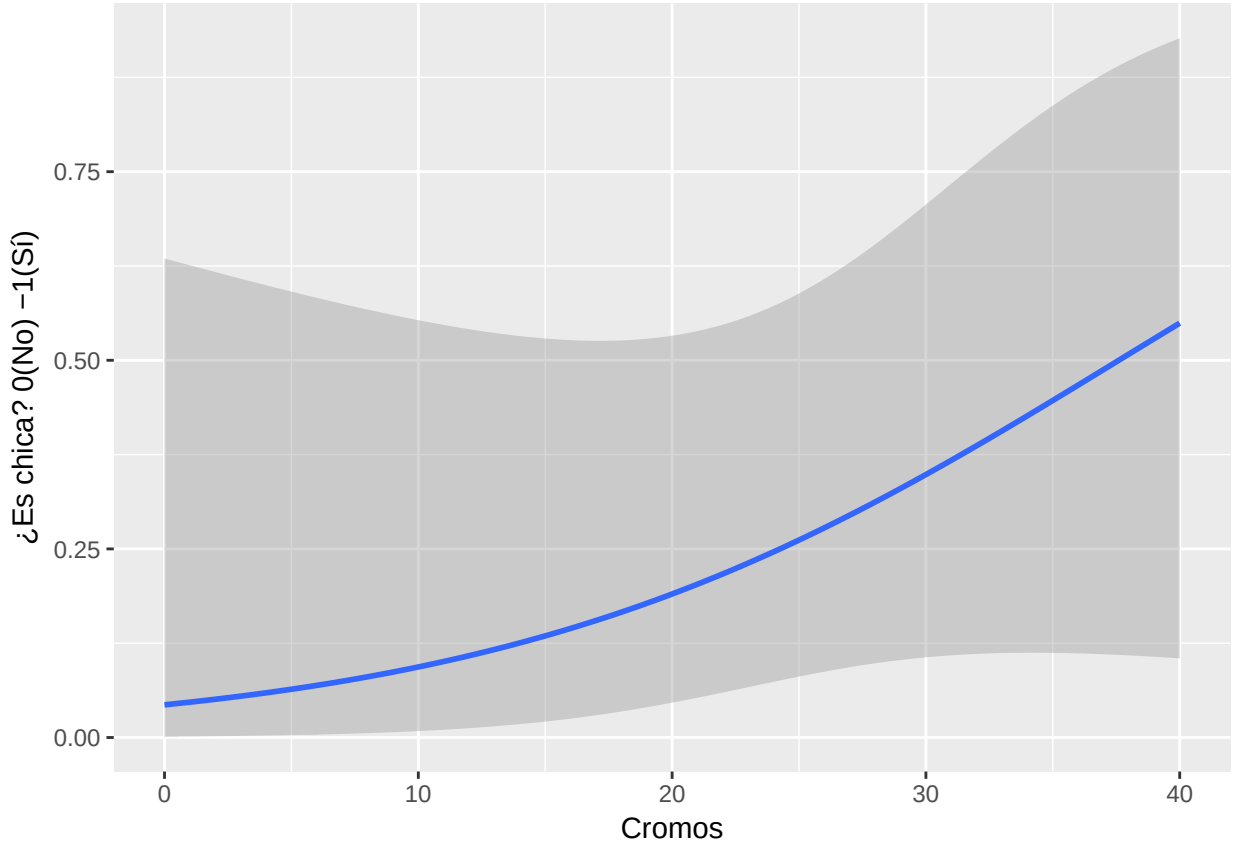
Otro tipo de regresión que nos es útil a menudo es la regresión logística. Se trata de un modelo de regresión dónde la variable dependiente es categórica. Por ejemplo: la probabilidad de Aprobar (SI o NO) un examen en función del número de horas estudiadas, o la determinación de si un mail es *SPAM* o no en función de

varios atributos o variables independientes (p.ej. el número de palabras, si contiene imágenes, links, etc.)

El modelo estima, por tanto, la probabilidad de una respuesta binaria (categórica) en base a uno o más predictores (o variables independientes) mediante una función logística.

Probemos con un ejemplo tonto en base a nuestros datos: probabilidad de ser chica en función del número de cromos que tiene una persona.

```
library(tidyverse)
library(ggplot2)
misdatos<-misdatos%>%
#creo la variable Es_Chica
  mutate(Es_chica=as.numeric(Sexo=="f"))
#uso geom_smooth para definir la curva de tendencia de la regresión logística
regresion_logistica<-ggplot(data= misdatos,aes(x= Cromos,y= Es_chica)) +
  geom_smooth(method="glm",method.args=list(family="binomial")) +
  ylab("¿Es chica? 0(No) -1(Sí)")
regresion_logistica
```



4.3 Proyectar (Forecast)

Veamos ahora otro caso; supongamos que hemos ido anotando desde el año 2000 en un archivo las ventas de cada mes de nuestro negocio. Estaria bien poder predecir cuáles van a ser las de los proximos meses.

Podemos usar el paquete **Forecast** (Hyndman, 2017) desarrollado y mantenido por Rob Hyndman, para aplicar modelos proyectivos conocidos (tales como el modelo ARIMA)

Lo haríamos del siguiente modo:

Leemos los datos a partir de nuestro archivo ventas.csv.

```
df <- read.csv("C:/.../ventas.csv")
```

Transformamos los datos en un objeto temporal (*time series object*) del tipo `ts` (indicamos que los datos son mensuales y que el periodo de inicio es Enero del 2000).

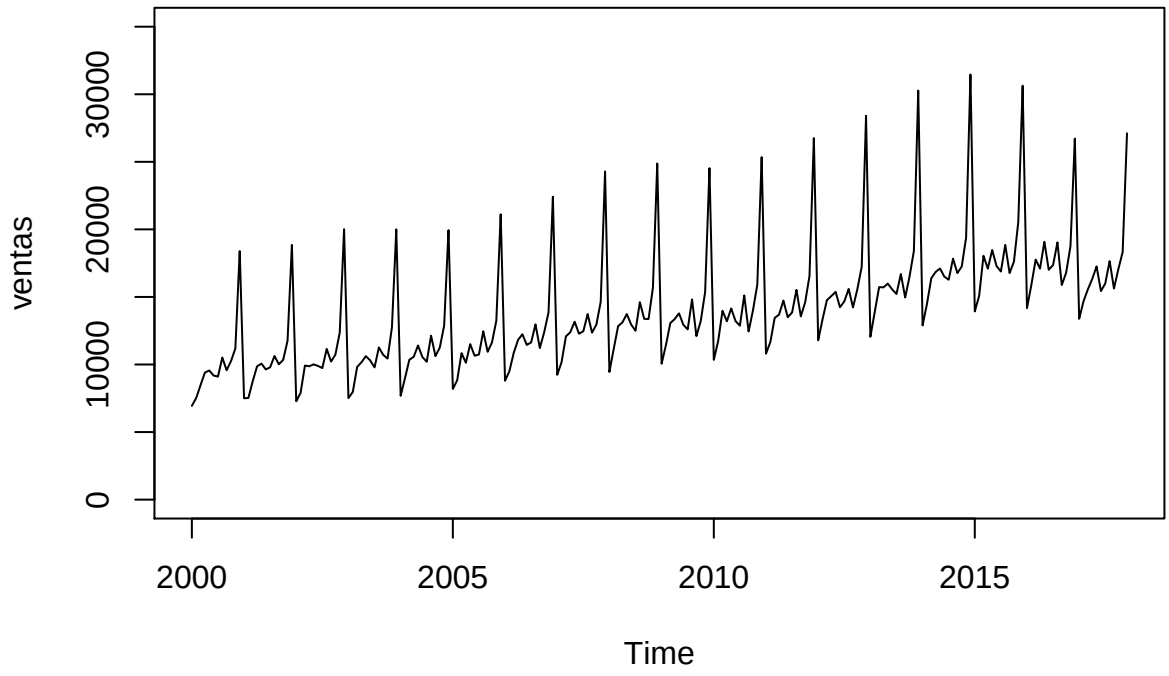
Mostramos los datos:

```
df <- read.csv("_bookdown_files/ventas.csv")
series <- ts(df, frequency = 12, start = c(2000,1))
print(series)
```

```
##        Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov
## 2000  6938  7524  8475  9401  9558  9182  9103 10513  9573 10254 11187
## 2001  7502  7524  8766  9867 10063  9635  9794 10628 10013 10346 11760
## 2002  7280  7902  9921  9869 10009  9893  9735 11157 10217 10730 12354
## 2003  7518  7961  9815 10168 10620 10301  9784 11264 10710 10439 12751
## 2004  7684  8991 10349 10570 11405 10554 10202 12134 10623 11250 12875
## 2005  8194  8835 10840 10131 11505 10654 10734 12461 10942 11635 13244
## 2006  8800  9499 10863 11825 12239 11451 11633 12971 11214 12384 13854
## 2007  9237 10171 12081 12386 13167 12280 12461 13734 12357 12948 14643
## 2008  9447 11170 12841 13124 13735 12953 12500 14610 13375 13369 15675
## 2009 10060 11450 13067 13362 13787 12935 12600 14818 12104 13218 15352
## 2010 10344 11730 13977 13195 14150 13210 12873 15113 12445 14006 15911
## 2011 10804 11662 13452 13691 14730 13496 13854 15522 13567 14601 16555
## 2012 11790 13344 14760 15058 15379 14237 14667 15588 14224 15570 17230
## 2013 12046 13878 15727 15708 15989 15559 15218 16697 14960 16509 18402
## 2014 12893 14474 16386 16848 17103 16505 16275 17832 16767 17253 19391
## 2015 13927 15077 18045 17096 18474 17289 16883 18850 16765 17614 20550
## 2016 14170 15877 17764 17098 19081 17006 17366 19038 15881 16791 18753
## 2017 13382 14681 15560 16334 17260 15429 16002 17650 15624 17046 18324
##        Dec
## 2000 18395
## 2001 18851
## 2002 20016
## 2003 20002
## 2004 19944
## 2005 21118
## 2006 22418
## 2007 24286
## 2008 24875
## 2009 24534
## 2010 25350
## 2011 26760
## 2012 28406
## 2013 30276
## 2014 31462
## 2015 30635
## 2016 26718
## 2017 27110
```

Ploteamos la serie temporal. Nos aseguramos que el eje `y` empieza por cero.

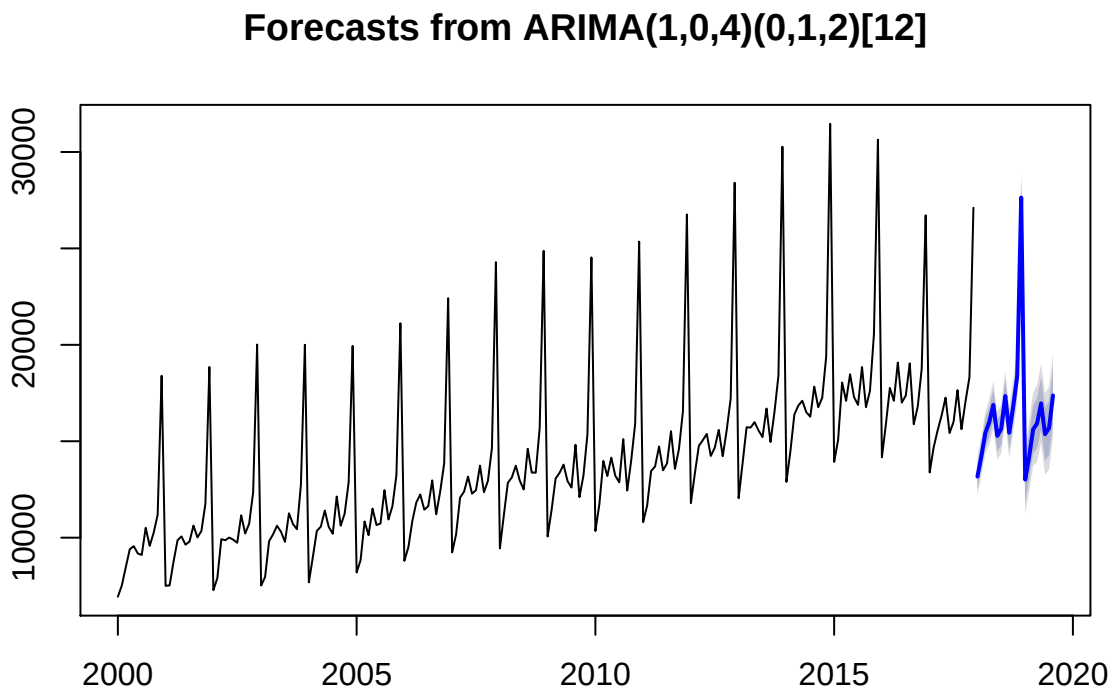
```
plot(series, ylim=c(0, 35000))
```



Generemos ahora una prevision para los próximos 20 periodos en base al modelo ARIMA. Lo hacemos en dos pasos: 1) creamos un modelo usando la función `auto.arima` del paquete `forecast` 2) generamos una proyección en base al modelo usando la función `forecast`

Representamos gráficamente la previsión:

```
require(forecast)
model_arima <- auto.arima(series)
fcast_arima <- forecast(model_arima, h = 20)
plot(fcast_arima)
```


Forecasts from ARIMA(1,0,4)(0,1,2)[12]
30000
20000
10000
2000
2005
2010
2015
2020

Chapter 5

R para Visualizar

> "There is no such thing as information overload. There is only bad design…."
>
> — Edward Tufte

En este capítulo veremos algunas opciones básicas para visualizar convenientemente nuestros datos.

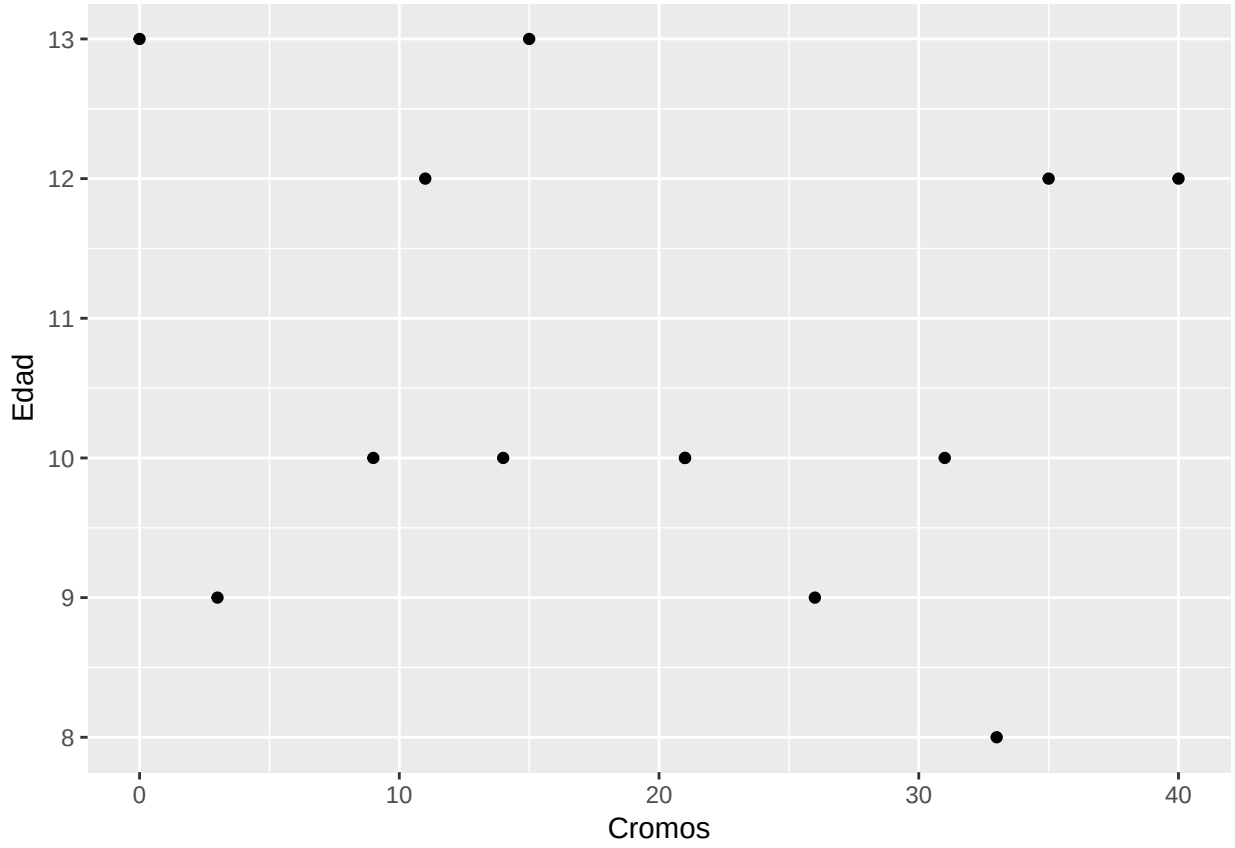
5.1 Gráficos de dispersión

Una primera forma de visualización, útil en muchos casos son los **gráficos de dispersión** o *scatterplots* en inglés. Para generar muchos tipos de visualizaciones nos será sumamente útil el paquete **ggplot2** (Wickham, 2009) creado por el señor Hadley Wickham. Lo primero, cargémoslo:

```
library(ggplot2)
```

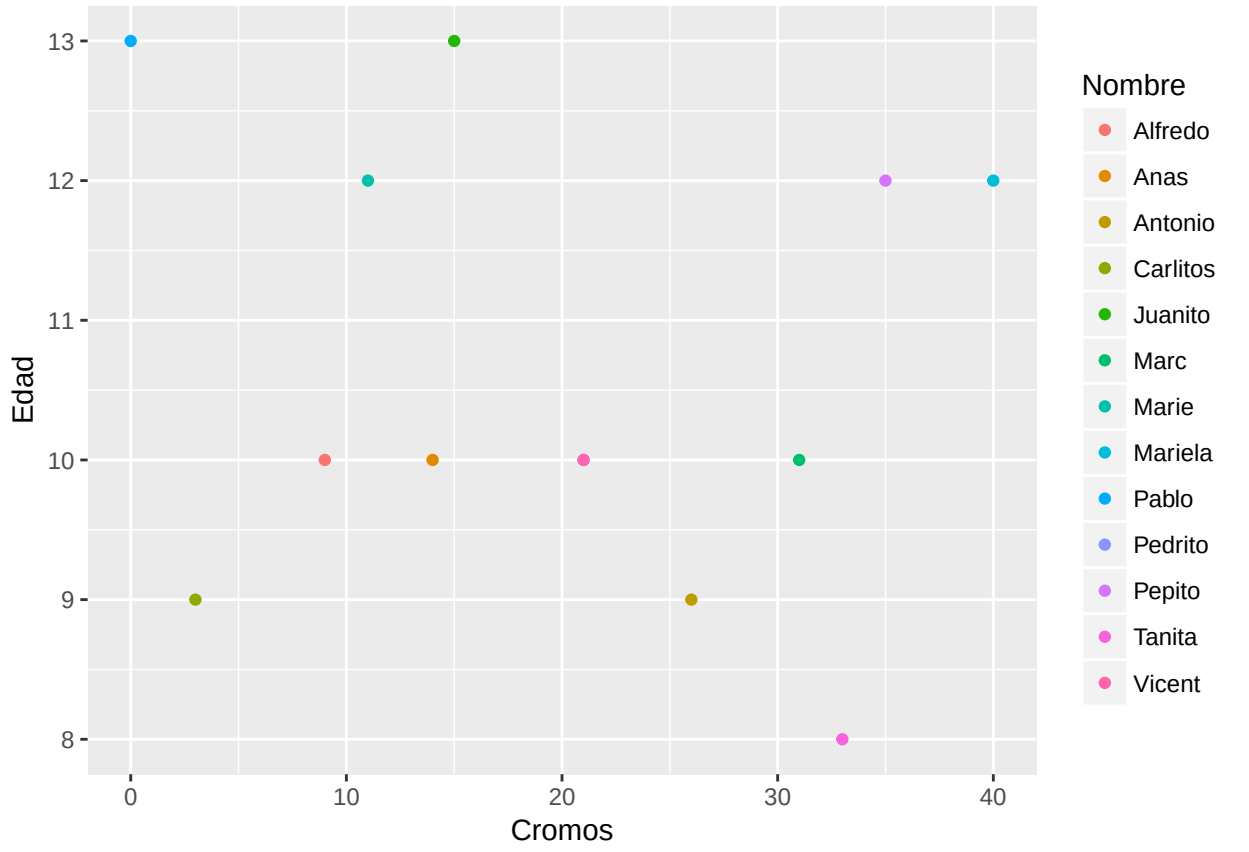
La opción más sencilla para visualizar nuestros datos es usar la función `qplot` dónde en los argumentos de la función pongo: la variable que quiero en el eje X, la que quiero en el eje Y, seguido de la tabla de los datos que estoy analizando (misdatos):

```
qplot(Cromos, Edad, data=misdatos)
```



Puedo añadir una tercera variable (Nombre) para representarla por ejemplo mediante color:

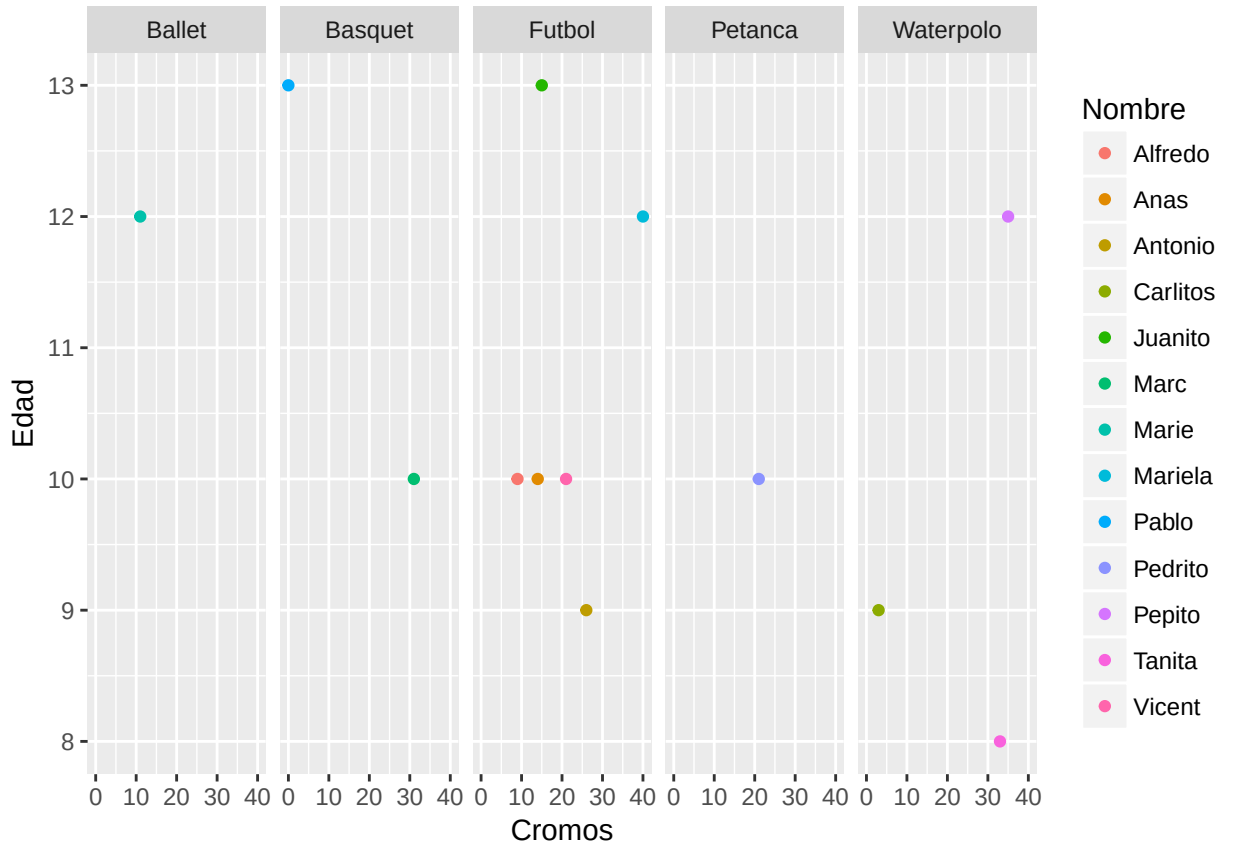
```
qplot(Cromos, Edad, data=misdatos, color=Nombre)
```



(nótese que la leyenda se me crea automáticamente).

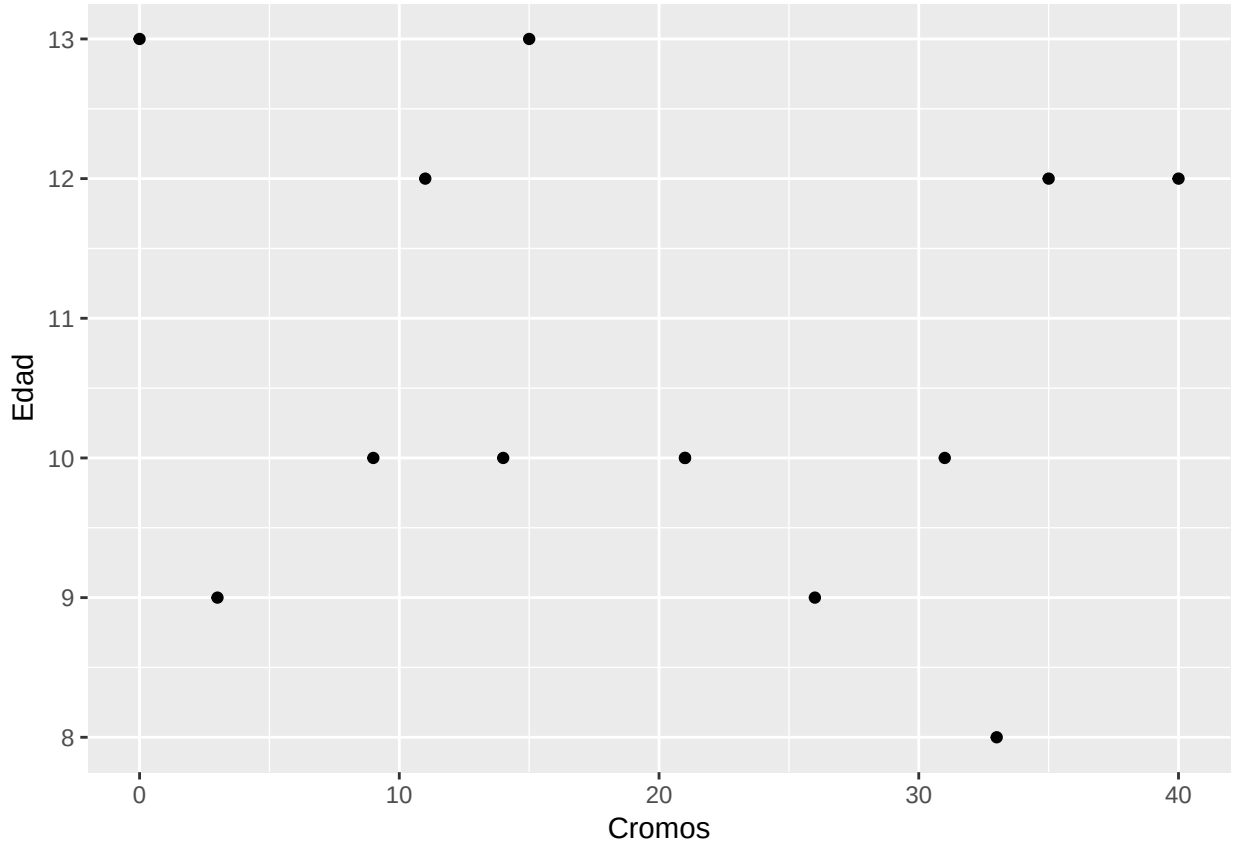
Otra utilidad conveniente a menudo a la hora de visualizar nuestros datos es hacer varias capas o particiones (**facets**) para mostrar los datos en función de determinadas variables. Veámoslos por ejemplo en función del deporte practicado.

```
qplot(Cromos, Edad, data=misdatos, color=Nombre, facets=.~Deporte)
```



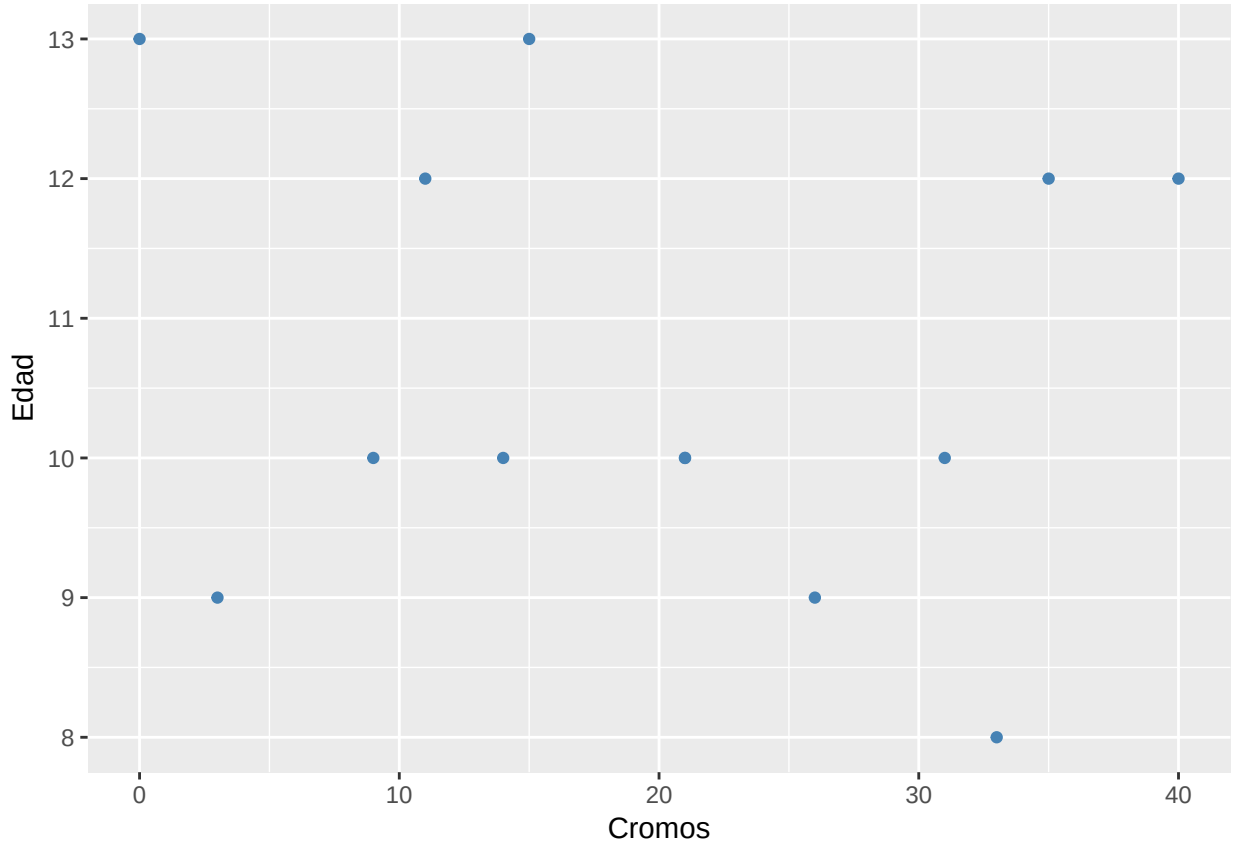
La opción `qplot` está bien para generar visualizaciones rápidas de nuestros datos, pero cuando queremos tener más control estético podemos usar la expresión `ggplot` seguida de los datos y a continuación los parámetros estéticos que deseemos (que indicamos con `aes()`), más la forma (`geom`) de los datos (sean puntos, barras u otras formas)

```
plot <- ggplot(misdatos, aes (Cromos, Edad)) + geom_point()
plot
```



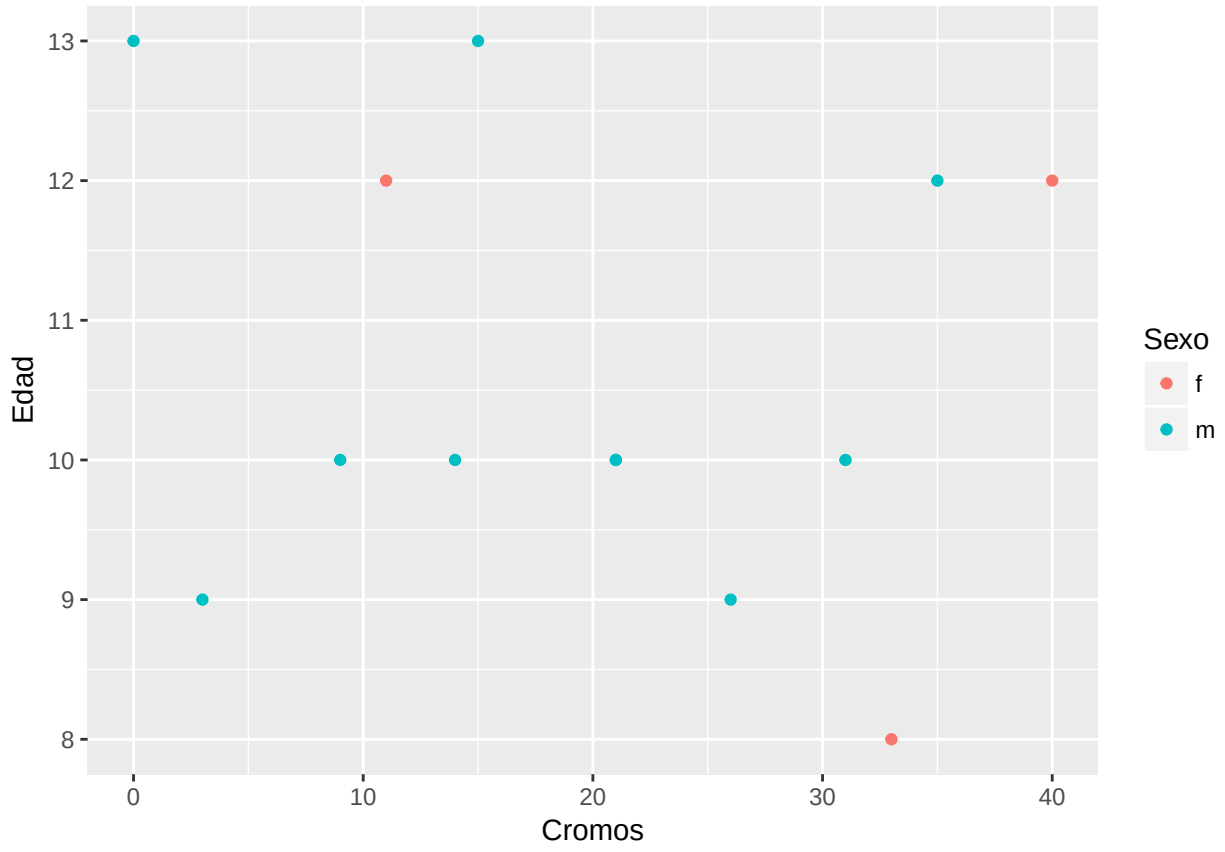
Así, por ejemplo puedo añadir estética de color a los puntos, bien como una constante:

```
plot <- ggplot(misdatos, aes (Cromos, Edad)) + geom_point(color = "steelblue")
plot
```



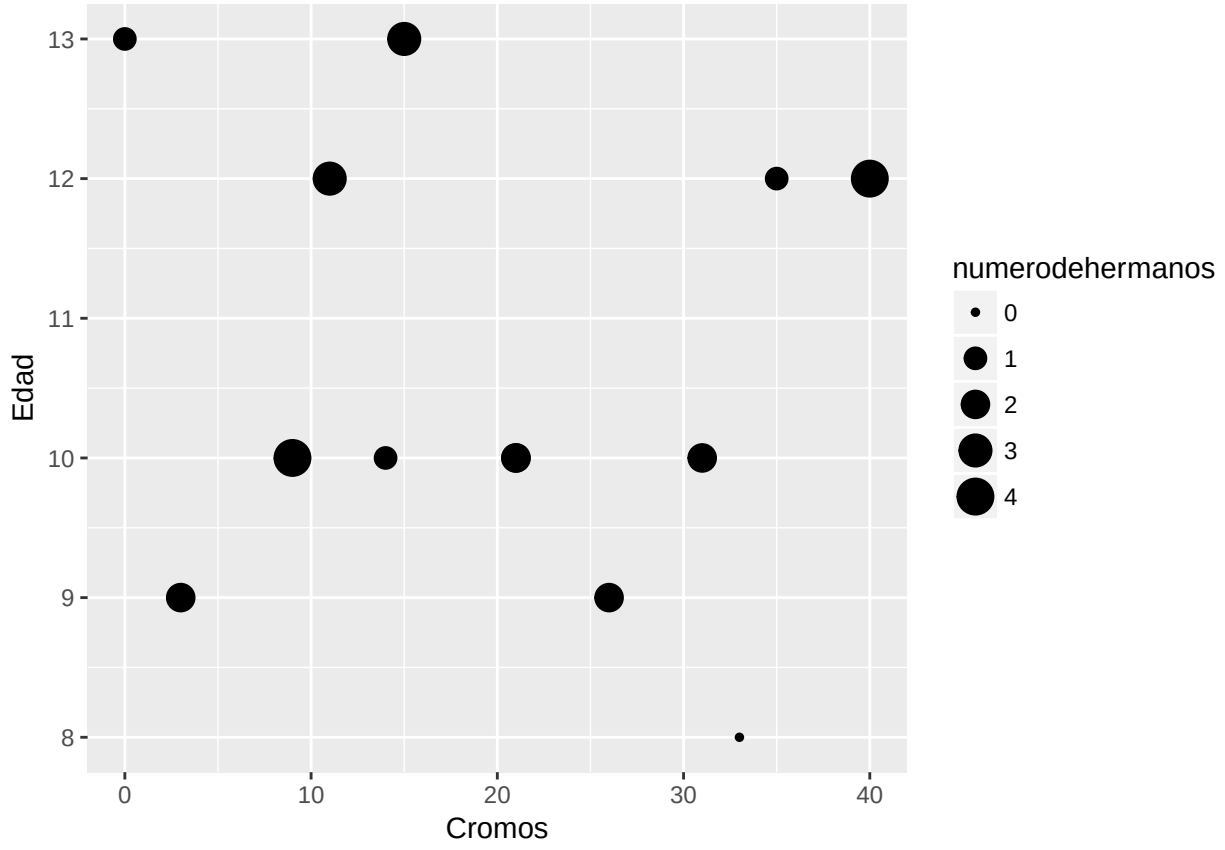
o bien como una variable (dónde el color es función del Sexo):

```
plot <- ggplot(misdatos, aes (Cromos, Edad)) + geom_point(aes(color=Sexo))
plot
```



O el tamaño (size) es función del número de hermanos:

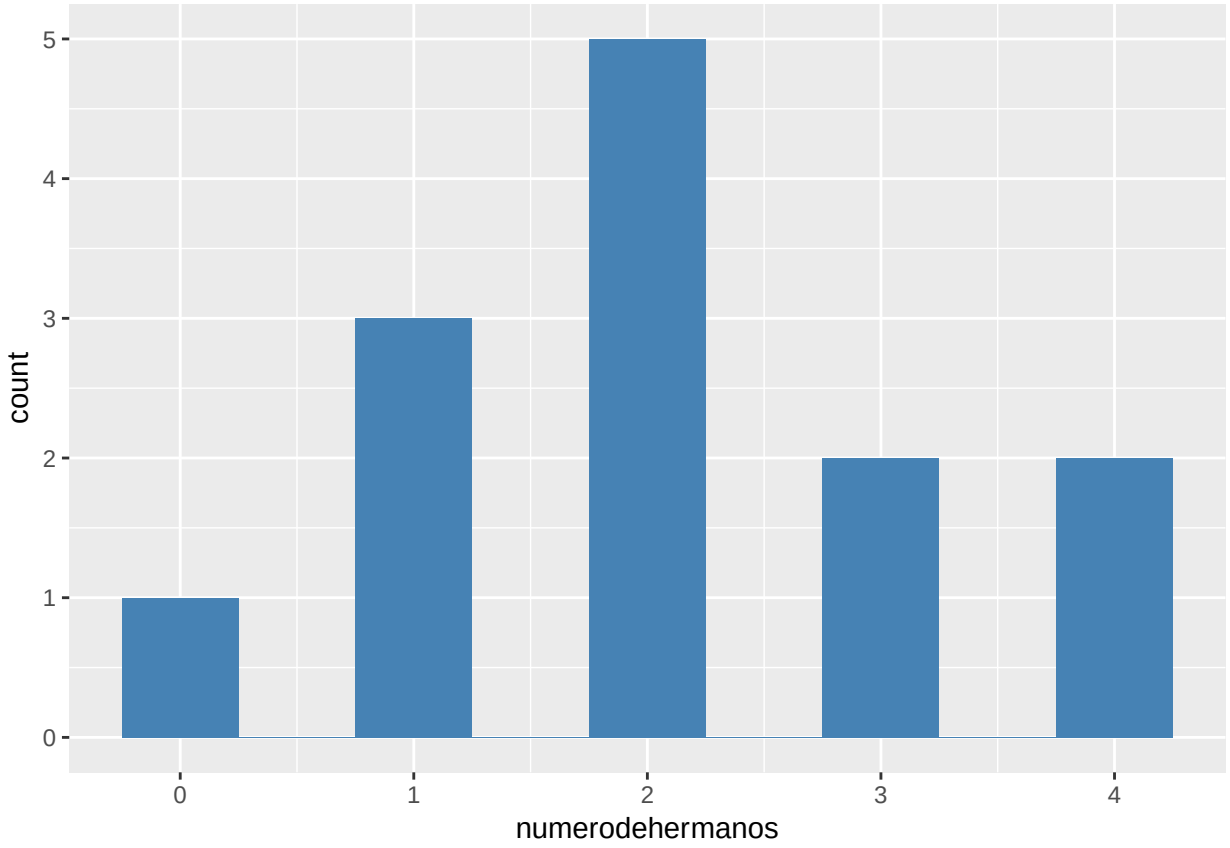
```
plot <- ggplot(misdatos, aes (Cromos, Edad)) + geom_point(aes(size=numerodehermanos))
plot
```



5.2 Histogramas

Para representar un histograma simplemente usariamos en este caso en nuestra expresión la forma estética `geom_histogram`.

```
library(ggplot2)
ggplot(misdatos, aes(numerodehermanos)) +
  geom_histogram(binwidth=.5, fill="Steelblue", show.legend = FALSE)
```



Nótese que a efectos estéticos hemos representado el histograma ajustando el ancho de las barras (con `binwidth`) y rellendando de color (con `fill`).

5.3 Redes

Eventualmente puede ser interesante visualizar datos que describen una red y percibir su estructura relacional.

Con el paquete **igraph** (Csardi and Nepusz, 2006) podemos leer una red cuya información tengamos expresada en forma de nodos y enlaces.

En nuestro caso partimos de un arxivo de nodos que es una tabla de nombres de personas y un archivo de enlaces que es una tabla con tres columnas (nombre de una persona, nombre de otra persona con quien interacciona y número de cromos intercambiados).

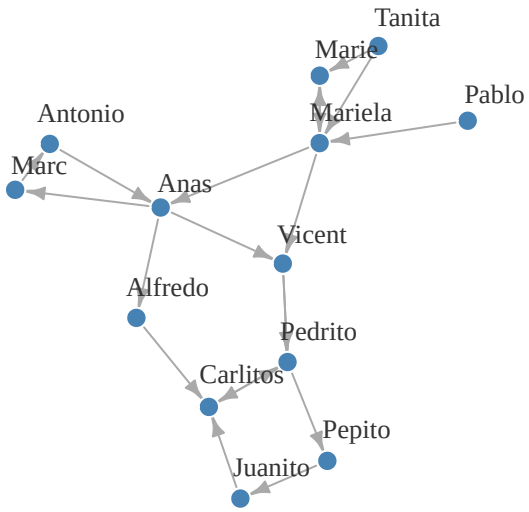
```
#cargamos el paquete igraph
library(igraph)

#leemos archivos de nodos y de enlaces que tenemos en nuestro directorio
nodos <- read.csv("_bookdown_files/grafo_nodos.csv", header=T, as.is=T)
enlaces <- read.csv("_bookdown_files/grafo_enlaces.csv", header=T, as.is=T)
```

Ahora podemos visualizar la red con el paquete `igraph`.

```
#trasformamos los datos en objetos de red mediante la función graph.data.frame
red <-graph.data.frame(enlaces, directed=TRUE, vertices = nodos)
```

```
#representamos el grafo especificando varios criterios estéticos
plot(red, layout=layout.fruchterman.reingold, vertex.size=9,
     vertex.label.color="grey20", vertex.label.dist=0.9,
     vertex.color="Steelblue", vertex.frame.color ="white",
     edge.arrow.size=0.5, edge.curved=0, vertex.label.font=9,
     vertex.label.cex=0.8)
```



Apreciamos un grafo resultante cuyos nodos son los nombres de las personas y los enlaces los vínculos que éstas establecen (en este caso intercambio de cromos).

Podemos llegar a hacer grafos mucho más sofisticados que este: reflejar varias variables en ellos, distintos layouts, aplicar conceptos de la teoría de grafos (centralidad, cercanía, intermediacion) para entenderlos mejor, etc.

Para aprender más acerca de representaciones de redes con R recomiendo leer los excelentes materiales al respecto de Katya Ognyanova.

5.4 Mapas geográficos

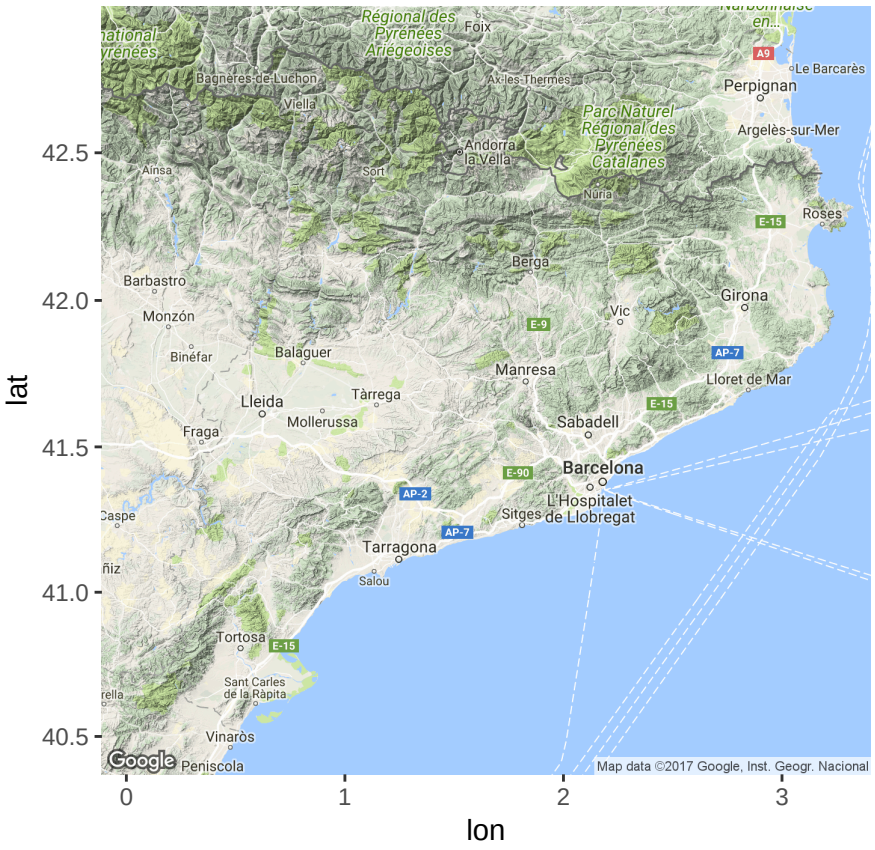
También podemos representar en R mapas geográficos de varios modos; usamos paquetes como **ggmap** (Kahle, 2016), **tmap** (Tennekes, 2017) para representar los mapas y luego con paquetes como por ejemplo **rgdal** (Bivand, 2017) podemos importar perímetros geoespaciales de áreas geográficas o países (en forma de *shapefiles*) en R y localizarlos en el mapa con nuestros datos.

Por ejemplo, podemos generar rápidamente un mapa de Europa con `tmap` con sólo indicar una región geográfica:

```
library(tmap)
#el paquete contiene una tabla con datos de los países europeos
data(Europe)
qtm(Europe)
```

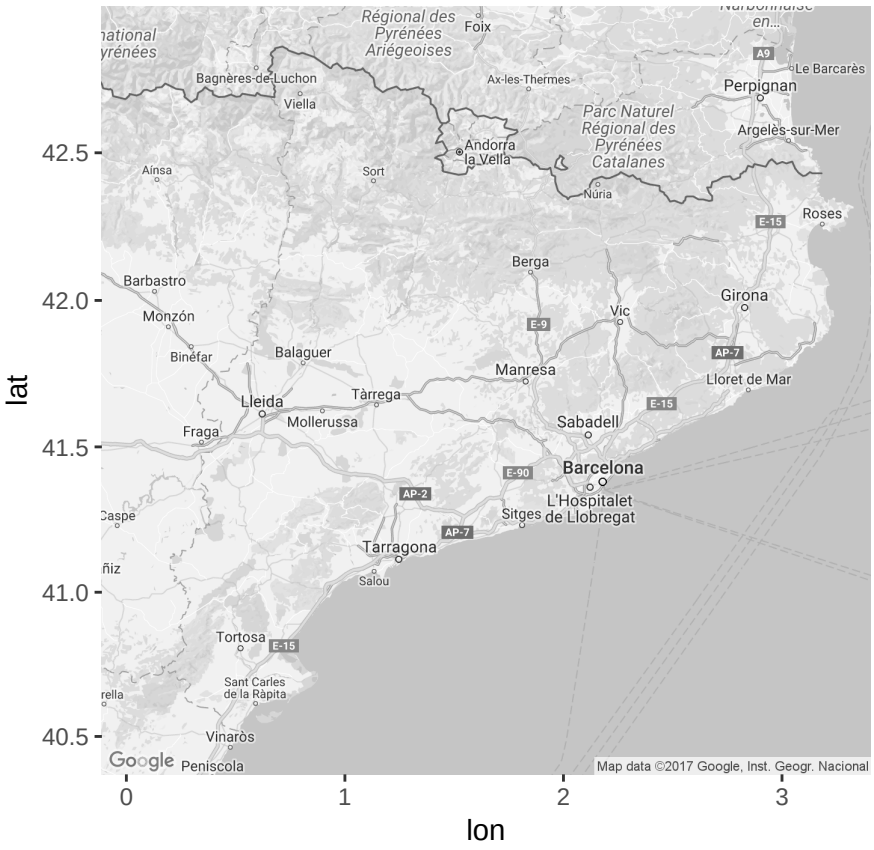
Alternativamente, con ggmap podríamos generar el mapa de una localización específica del siguiente modo:

```
library(ggmap)
#localizamos con longitud y latitud la zona que queremos
cat <- c(lon = 1.6430518, lat = 41.6960344)
#generamos el mapa indicando la fuente (google) y el zoom que queremos
map <- get_map(location=cat, source="google", zoom=8)
ggmap(map)
```



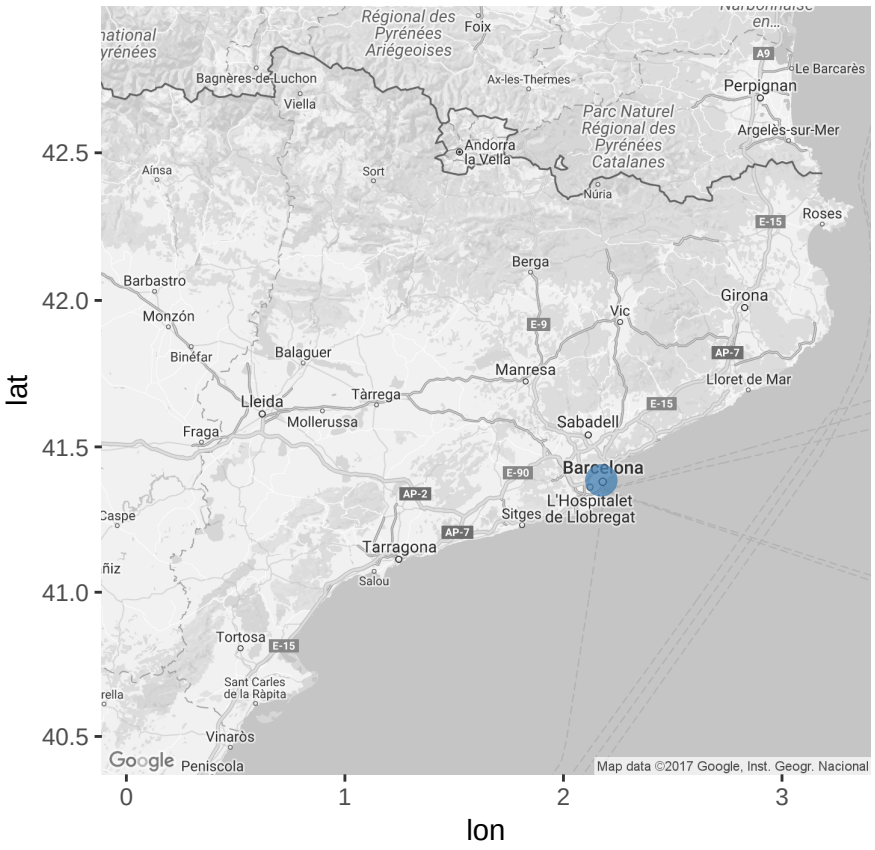
Si quisieramos otro tipo de representación (maptype) y color (por ejemplo en blanco y negro) indicaríamos:

```
#el tipo de mapa de google puede ser "roadmap", "terrain", "satellite" o "hybrid"
map <- get_map(location=cat, source="google", zoom=8, maptype="roadmap", color="bw")
ggmap(map)
```



Si ahora quiero añadir un punto en el mapa (por ejemplo mi ciudad, Barcelona):

```
#añado la long/lat de Barcelona
ggmap(map)+ geom_point (aes (x = 2.1734, y = 41.3851),
#e indico la transparencia (alpha), color y tamaño del punto
alpha = .3, color="steelblue", size = 5)
```



Incluso puedo añadir datos al mapa; por ejemplo, los distritos de la ciudad.

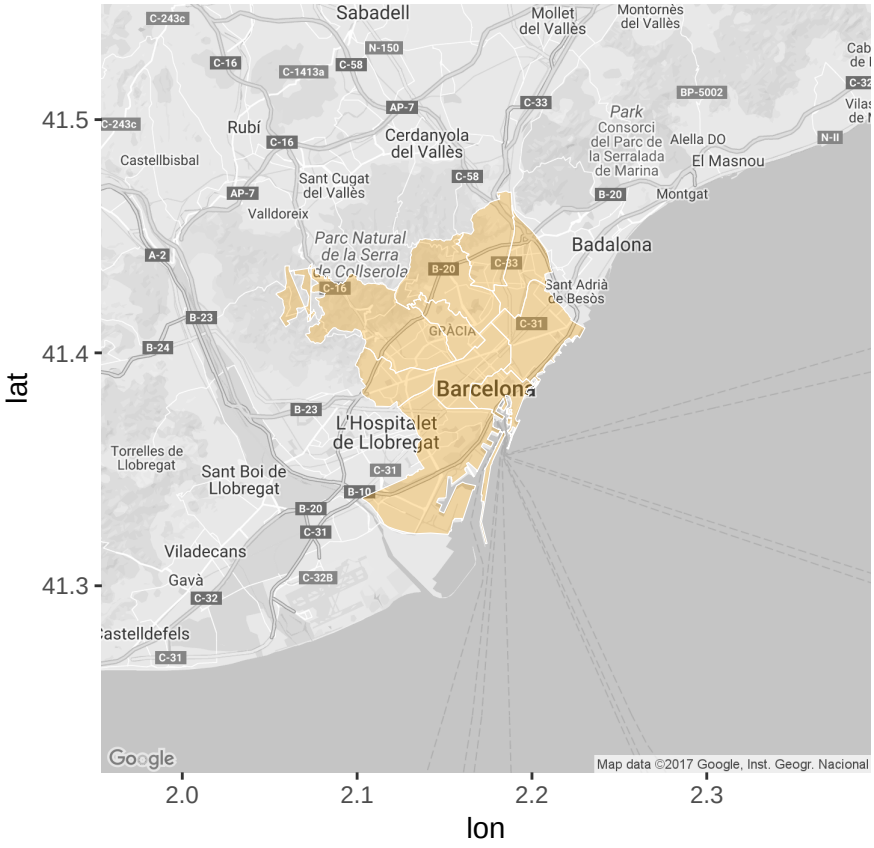
```
#primero encuentro las shapefiles de los distritos de Barcelona en datos abiertos:
#https://laura-an.carto.com/tables/shapefile_distrito_barcelona/public
#me las descargo en mi directorio de trabajo

#focalizo el mapa en Barcelona
bcn <- c(lon = 2.1734, lat = 41.3851)
map <- get_map(location=bcn, source="google", zoom=11, maptype="roadmap", color="bw")

#cargo el paquete rgdal para importar las shapefiles de los distritos al mapa
library(rgdal)
shapefiles<-readOGR(dsn="_bookdown_files/shapefile_distrito_barcelona.shp",
                    layer="shapefile_distrito_barcelona")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "_bookdown_files/shapefile_distrito_barcelona.shp", layer: "shapefile_distrito_barcelona"
## with 10 features
## It has 12 fields
```

```
#lo represento en el mapa:
ggmap(map)+ geom_polygon(aes(x = long, y = lat, group=id),
data = shapefiles, color = "white", fill = "orange",
alpha = .3, size = .2)
```


Sabadell
Mollet del Vallès
Montornès del Vallès
Rubí
Cerdanyola del Vallès
Castellbisbal
Sant Cugat del Vallès
Valldoreix
Parc Natural de la Serra de Collserola
Park Consorci del Parc de la Serralada de Marina
Alella DO
El Masnou
Montgat
Badalona
Sant Adrià de Besòs
GRÀCIA
Barcelona
L'Hospitalet de Llobregat
Torrelles de Llobregat
Sant Boi de Llobregat
Viladecans
Gavà
Castelldefels
C-243c
C-16
C-1413a
C-58
N-150
AP-7
C-33
BP-5002
N-II
B-20
A-2
B-23
B-24
C-31
C-32B
C-32
Google
Map data ©2017 Google, Inst. Geogr. Nacional
41.5
41.4
41.3
lat
2.0
2.1
2.2
2.3
lon

Chapter 6

R que R

He intentado mostrar en esta brevísima introducción a R, las funcionalidades básicas de este lenguaje de Programación estadística y algunas de las aplicaciones posibles. Esto ha sido sólo una muestra para despertar el apetito. Uno de los aspectos más positivos de este lenguaje es, precisamente, la activa comunidad de usuarios que existe a su entorno, gracias a la cual es muy fácil obtener soporte para problemas específicos y seguir aprendiendo mediante infinitud de recursos, muchos de ellos gratuitos.

A continuación os dejo una lista de recursos para seguir aprendiendo R que R.

6.1 Tutoriales

Algunos tutoriales interesantes para seguir aprendiendo son:

- An Introduction to Statistical Learning with Applications in R (ISLR) de los profesores de Stanford **Trevor Hastie** y **Rob Tibshirani**. Mucho conocimiento transmitido de un modo muy cálido y relajado.
- El libro R for Data Science de **Hadley Wickham**, experto neozelandés que trabaja en RStudio. Toda una referencia actual en el desarrollo de este lenguage de programación y creador e integrador de gran cantidad de paquetes que facilitan todo el proceso analítico.
- R-bloggers, blog curado por **Tal Galili**, Doctor de la Universidad de Tel Aviv. Enrorme repositorio de posts de miembros de de la comunidad de R a nivel mundial.
- twotorials de **Anthony Damico** una introducción muy ágil y divertida a R (videos de 2 minutos usando R en bruto ¡sin Rstudio!).
- Data Science Central de **Vincent Granville**
- KDnuggets de **Gregory Piatetsky**
- Kirk Borne científico de datos principal en la consultora pionera BoozAllen.
- Analytics Vidhya blog de un equipo muy activo de analistas de Bombay, liderados por **Kunal Jail** y que son además excelentes comunicadores y están siempre a la última de los avances en análisis de datos y Machine Learning, lenguajes de programación, etc. Como la gente de Rstudio y Datacamp, también publican útiles "chuletas" o Cheat Sheets Para principiantes pero también para usuarios más avanzados.
- Variance Explained de David Robinson videos cortos, super claros y explicativos.
- El libro Tidy Textmining, escrito por Julia Silge y David Robinson.

- François Husson, análisis de datos multivariantes y técnicas de clusterización con R.
- Para trabajar con predicciones en series temporales es muy útil el trabajo del profesor Rob J Hyndman de la Monash University, creador del paquete Forecast

Otros recursos abiertos y gratuitos útiles son:

- Los excelentes materiales del curso CSI E-109 del Harvard Distance Education, impartido por los porfesores de Harvard _professors **Joe Blitzstein**, **Hanspeter Pfister** y **Verena Kaynig-Fittkau**.
- Inventarios de recursos intersantes como Data Science Masters de **Clare Corthell** de Summer.ai
- Learn Data Science de **Nitin Borwankar**.
- Kaggle Plataforma, red social de científicos de datos y entorno para competiciones de proyectos de minería de datos, aprendizaje automático y modelos predicitivos, creada por Anthony Goldbloom.

Coursera, Udacity y Datacamp, entre otros ponen al alcance de todos el poder iniciarse en ciencia de datos y R.

Finalmente existen numerosos blogs personales de doctorandos, *practitioners* o usuarios apasionados de R que muestran modos de analizar datos para casos específicos. También es sumamente útil encontrar **notebooks** donde se conjugan scripts de código y explicaciones de los mismos.

6.2 Obtener ayuda

Para obtener ayuda en R sobre una función -por ejemplo sobre `lm`- puedo hacer `?lm`, `help(lm)` o `help ("lm")` y me aparecerá la ayuda en el visor del entorno de R.

También puede ser conveniente buscar ejemplos de esa función con `example (lm)`.

Cuando al ejecutar comandos nos aparece un error (mensaje de texto en rojo en la consola de R), es importante leerlo para aprender de él (¡como en la vida!)

Otra gran opción que tenemos para resolver errores es buscar en la web de preguntas y respuestas **stackoverflow** https://stackoverflow.com. Si simplemente copiamos y pegamos el error que nos aparece en R en Google, es muy probable que nos dirija a stackoverflow. Y es muy probable también que en uno de sus foros alguien haya solucionado ya el problema que buscamos u otro similar que nos ayude a entender. De igual modo podemos contribuir allí y aclarar dudas a otros. Estos foros de preguntas y respuestas son, por lo general, una enorme fuente de aprendizaje y una inyección de optimismo en la humanidad.

Chapter 7

Sumario

Espero que este libro te haya servido para iniciarte en el análisis de datos con R y que hayas podido entrever sus grandes posibilidades analíticas.

No hemos hablado de otras muchas técnicas estadísticas posibles en R, desde opciones gráficas (incluso interactivas) a opciones para procesar contenidos textuales que en R pueden implementarse mediante paquetes como `tm`, `OpenNLP` o `tidytext` u otros, algoritmos de Aprendizaje automático, mediante paquetes como `e1071`, `randomForest` o `Caret`, entre muchos, muchos otros. Las aplicaciones son infinitas. Nos queda todavía mucho que aprender.

¿Te enganchó?

A seguir pues, **¡R que R!**

¡Muchas gracias!

Enric Escorsa O'Callaghan

Bibliography

Bivand, R. (2017). *rgdal: Bindings for the 'Geospatial' Data Abstraction Library*. R package version 1.2-13.

Bryan, J. and et al., H. W. (2017). *readxl: Read Excel Files*. R package version 1.2-13.

Csardi, G. and Nepusz, T. (2006). The igraph software package for complex network research. *InterJournal*, Complex Systems:1695.

Dragulescu, A. A. (2015). *xlsx: Read, write, format Excel 2007 and Excel 97/2000/XP/2003 files*. R package version 1.2-13.

Hyndman, R. (2017). *Forecasting Functions for Time Series and Linear Models*. R package version 8.1.

Kahle, D. (2016). *ggmap: Spacial visualization with ggplot2*. R package version 2.6.1.

R Core Team (2013). *R: A Language and Environment for Statistical Computing*.

Tennekes, M. (2017). *tmap; Thematic maps*. R package version 1.10.

Wickham, H. (2009). *ggplot2: Elegant Graphics for Data Analysis*. Springer Verlag New York.

Wickham, H. (2016). *rvest: Easily Harvest (Scrape) Web Pages*. R package.

Wickham, H. (2017a). *dplyr: A grammar of Data Manipulation*. R package version 0.7.2.

Wickham, H. (2017b). *readr: Read Rectangular Text Data*. R package version 1.1.1.

Wickham, H. (2017c). *stringr: Simple, Consistent Wrappers for Common String Operations*. R package version 1.2.0.

Xie, Y. (2016). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.3.