Introducción al Análisis de Datos con R

Rubén Fernández Casal (ruben.fcasal@udc.es)
Javier Roca-Pardiñas (roca@uvigo.es)
Julián Costa Bouzas (julian.costa@udc.es)
Manuel Oviedo de la Fuente (manuel.oviedo@udc.es)

Edición: Junio de 2022. Impresión: 2023-01-30. ISBN: 978-84-09-41823-7

Índice general

Prólogo

Este es un libro introductorio al análisis de datos con R.

En el Apéndice A se detallan los pasos para la instalación de `R` y el entorno de desarrollo RStudio. En la Sección Enlaces de las Referencias se incluyen recursos adicionales, incluyendo algunos que pueden ser útiles para el aprendizaje de R.

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

https://rubenfcasal.github.io/intror.

donde puede descargarse en formato pdf.

Para ejecutar los ejemplos mostrados en el libro sería necesario tener instalados los siguientes paquetes: `lattice`, `ggplot2`, `foreign`, `car`, `leaps`, `MASS`, `RcmdrMisc`, `lmtest`, `glmnet`, `mgcv`, `rmarkdown`, `knitr`, `dplyr`, `tidyr`. Por ejemplo mediante los siguientes comandos:

```
pkgs <- c("lattice", "ggplot2", "foreign", "car", "leaps", "MASS", "RcmdrMisc",
          "lmtest", "glmnet", "mgcv", "rmarkdown", "knitr", "dplyr", "tidyr")
install.packages(setdiff(pkgs, installed.packages()[,"Package"]), dependencies = TRUE)
```

(puede que haya que seleccionar el repositorio de descarga, e.g. *Spain (Madrid)*).

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

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

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

Para citar este libro se puede emplear la referencia:

Fernández-Casal R., Roca-Pardiñas J., Costa J. y Oviedo-de la Fuente M. (2022). *Introducción al Análisis de Datos con R*. ISBN: 978-84-09-41823-7. https://rubenfcasal.github.io/intror.

También puede resultar de utilidad la siguiente entrada BibTeX:

```
@book{fernandezetal2022,
    title        = {Introducción al Análisis de Datos con R},
    author       = {Fernández-Casal, R.; Roca-Pardiñas, J.; Costa, J.;Oviedo-de la Fuente, M.},
    year         = {2022},
    note         = {ISBN 978-84-09-41823-7},
```

```
    url           = {https://rubenfcasal.github.io/intror/}
}
```

Capítulo 1

Introducción

El entorno estadístico R puede ser una herramienta de gran utilidad a lo largo de todo el proceso de obtención de información a partir de datos (normalmente con el objetivo final de ayudar a tomar decisiones).

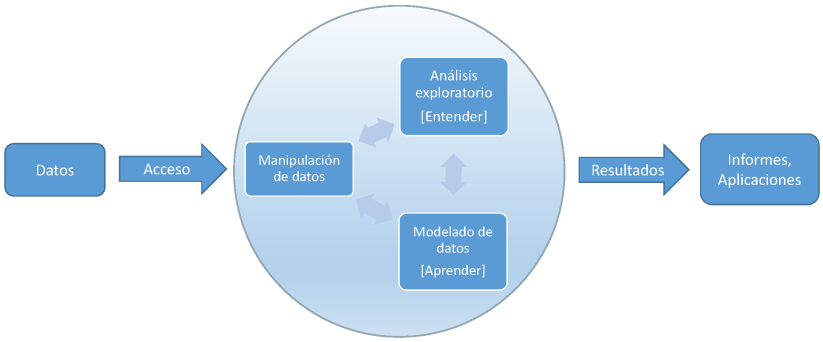


Figura 1.1: Etapas del proceso

1.1 El lenguaje y entorno estadístico R

R es un lenguaje de programación desarrollado específicamente para el análisis estadístico y la visualización de datos.

- El lenguaje R es interpretado (similar a Matlab o Phyton) pero orientado al análisis estadístico (fórmulas modelos, factores,…).
  - derivado del S (Laboratorios Bell).
- R es un **Software Libre** bajo las condiciones de licencia GPL de GNU, con código fuente de libre acceso.
  - Además de permitir crear **nuevas funciones**, se pueden examinar y modificar las ya existentes.
- Multiplataforma, disponible para los sistemas operativos más populares (Linux, Windows, MacOS X, …).

1.1.1 Principales características

Se pueden destacar las siguientes características del entorno R:

- Dispone de numerosos complementos (librerías, paquetes) que cubren "literalmente" todos los campos del análisis de datos.

- Repositorios:
  - CRAN (9705, 14972, 19122, …)
  - Bioconductor (1289, 1741, 2183, …),
  - GitHub, …
- Existe una comunidad de usuarios (programadores) muy dinámica (multitud de paquetes adicionales).
- Muy bien documentado y con numerosos foros de ayuda.
- Puntos débiles (a priori): velocidad, memoria, …

Aunque inicialmente fue un lenguaje desarrollado por estadísticos para estadísticos:

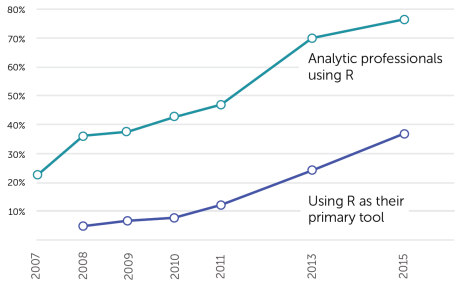


Figura 1.2: Rexer Data Miner Survey 2007-2015

Hoy en día es muy popular:

Figura 1.3: [IEEE Spectrum](https://spectrum.ieee.org) Top Programming Languages, 2019

`R` destaca especialmente en:

- Representaciones gráficas.
- Métodos estadísticos "avanzados":
  - *Data Science*: *Statistical Learning*, *Data Mining*, *Machine Learning*, *Business Intelligence*, …
  - Datos funcionales.
  - Estadística espacial.
  - …

- Análisis de datos “complejos”:
  - Big Data.
  - Lenguaje natural (*Text Mining*).
  - Análisis de redes.
  - ...

En el Apéndice A se detallan los pasos para la instalación de R y el entorno de desarrollo RStudio. En la Sección Enlaces de las Referencias se incluyen recursos adicionales, incluyendo algunos que pueden ser útiles para el aprendizaje de R.

1.2 Interfaz de comandos

Normalmente se trabaja en R de forma interactiva empleando una **interfaz de comandos** donde se teclean las instrucciones que se pretenden ejecutar. En Linux se trabaja directamente en el terminal de comandos y se inicia ejecutando el comando R. En Windows se puede emplear el menú de inicio para ejecutar R (e.g. abriendo *R x64 X.Y.Z*) y se mostrará una ventana de consola que permite trabajar de modo interactivo (ver Figura 1.4).

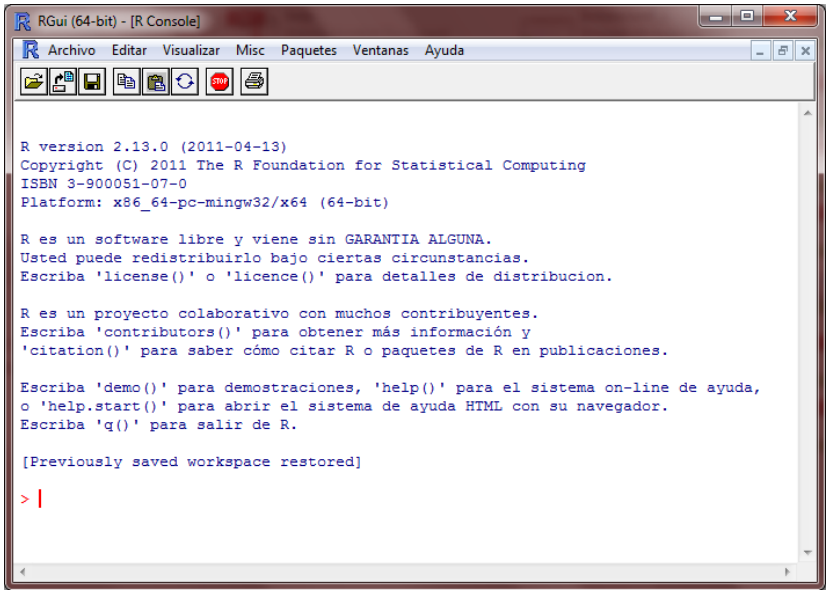


Figura 1.4: Consola de ‘R‘ en Windows (modo MDI).

En la línea de comandos R muestra el carácter > (el *prompt*) para indicar que está a la espera de instrucciones. Para ejecutar una línea de instrucciones hay que pulsar *Retorno* (y por defecto se imprime el resultado).

Por ejemplo, para obtener una secuencia de números desde el 1 hasta el 10, se utilizará la sentencia:

```
1:10
```

obteniéndose el resultado

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

Se pueden escribir varias instrucciones en una misma línea separándolas por “;”.

```
2+2; 1+2*4
```

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

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

Si no se completó algún comando, el prompt cambia a + (habría que completar la instrucción anterior antes de escribir una nueva, o pulsar *Escape* para cancelarla).

Se pueden recuperar líneas de instrucciones introducidas anteriormente pulsando la tecla *Arriba*, a fin de re-ejecutarlas o modificarlas.

La ventana consola ejecuta de forma automática cada línea de comando. Sin embargo, suele interesar guardar un conjunto de instrucciones en un único archivo de texto para formar lo que se conoce como un *script* (archivo de código). Las instrucciones del script se pueden pegar en la ventana de comandos para ser ejecutadas, pero también hay editores o entornos de desarrollo que permiten interactuar directamente con R.

Por ejemplo, en la consola de R en Windows se puede abrir una ventana de código seleccionando el menú *Archivo > Nuevo script*. Posteriormente se pueden ejecutar líneas de código pulsando *Ctrl+R*.

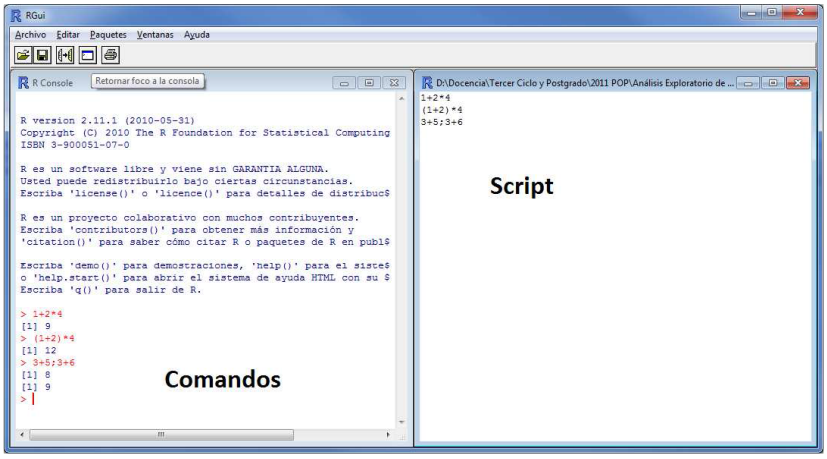


Figura 1.5: Ventanas de la consola y de comandos en Windows (modo MDI).

Sin embargo, nosotros recomendamos emplear *RStudio Desktop*.

1.3 El entorno de desarrollo RStudio Desktop

Al ejecutar RStudio se muestra la ventana principal:

Por defecto RStudio está organizado en cuatro paneles:

- Editor de código (normalmente un fichero *.R* o *.Rmd*).
- Consola de R (y terminal de comandos del sistema operativo).
- Explorador del entorno e historial.
- Explorador de archivos, visor de gráficos, ayuda y navegador web integrado.

Primeros pasos:

- Presionar *Ctrl-Enter* (*Command-Enter* en OS X) para ejecutar la línea de código actual o el código seleccionado (también se puede emplear el botón *Run* en la barra de herramientas del Editor o el menú *Code*).
- Presionar *Tab* para autocompletado.
- Pulsar en el nombre del objeto en la pestaña *Environment*, o ejecutar `View(objeto)` en la consola, para visualizar el objeto en una nueva pestaña del editor.

Información adicional:

- RStudio cheatsheet

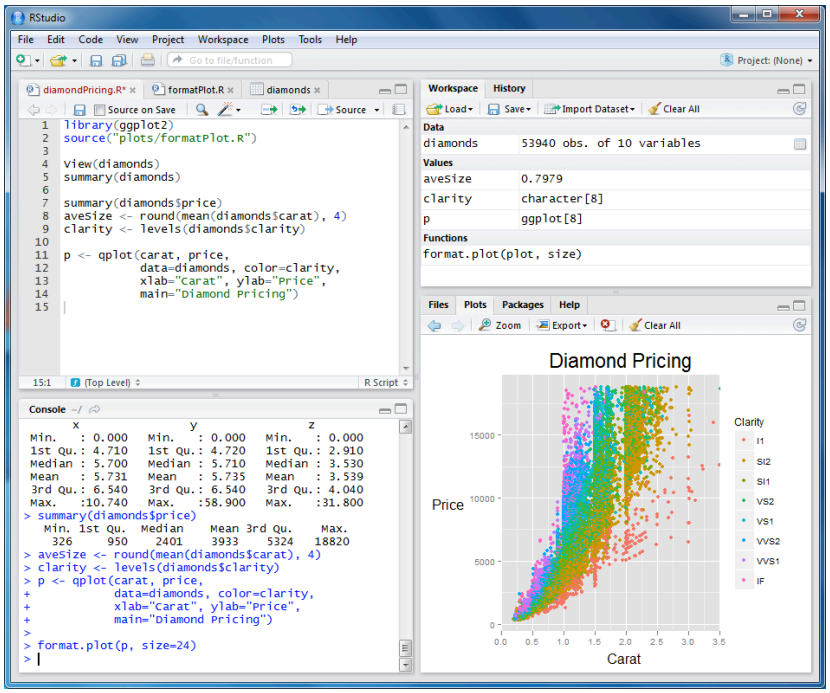


Figura 1.6: Ventana principal de RStudio.

- Using the RStudio IDE

1.4 Ayuda

Se puede acceder a la ayuda empleando el entorno de comandos o los menús correspondientes de la intefaz gráfica. Por ejemplo en RStudio se puede emplear el menú *Help*, y en la consola de `R` el menú *Ayuda > Manuales (en PDF)*. Para acceder a la ayuda desde la interfaz de comandos se puede ejecutar `help.start()` (también puede ser de interés la función `demo()`).

Todas las funciones de `R` están documentadas. Para obtener la ayuda de una determinada función se utilizará `help(función)` o de forma equivalente `?función`.

Por ejemplo, la ayuda de la función `rnorm` (utilizada para la generación de datos con distribución normal) se obtiene con el código

```
help(rnorm)
?rnorm
```

En muchas ocasiones no se conoce el nombre exacto de la función de la que queremos obtener la documentación. En estos casos, la función `help.search()` realiza búsquedas en la documentación en todos los paquetes instalados, estén cargados o no. Por ejemplo, si no conocemos la función que permite calcular la mediana de un conjunto de datos, se puede utilizar

```
help.search("median")
```

Para más detalles véase `?help.search`

1.5 Una primera sesión

Como ya se comentó, al emplear la interfaz de comandos, el usuario puede ir ejecutando instrucciones y se va imprimiendo el resultado. Por ejemplo:

```
3+5
```

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

```
sqrt(16) # raiz cuadrada de 16
```

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

```
pi # R reconoce el número pi
```

```
## [1] 3.141593
```

Nótese que en los comandos se pueden hacer comentarios utilizando el símbolo `#`.

Los resultados obtenidos pueden guardarse en objetos empleando el operador asignación `<-` (o `=`). Por ejemplo, al ejecutar

```
a <- 3 + 5
```

el resultado de la suma se guarda en el objeto `a` (se crea o se reescribe si ya existía previamente). Se puede comprobar si la asignación se ha realizado correctamente escribiendo el nombre del objeto (equivalente a ejecutar `print(a)`)

```
a
```

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

Es importante señalar que `R` diferencia entre mayúsculas y minúsculas, de modo que los objetos `a` y `A` serán diferentes.

```
a <- 1:10 # secuencia de números
A <- "casa"
a
```

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

```
A
```

```
## [1] "casa"
```

Nota: Habitualmente no habrá diferencia entre la utilización de las asignaciones hechas con `=` y `<-` (aunque nosotros emplearemos el segundo). Las diferencias aparecen a nivel de programación y se tratarán en el Capítulo 11.

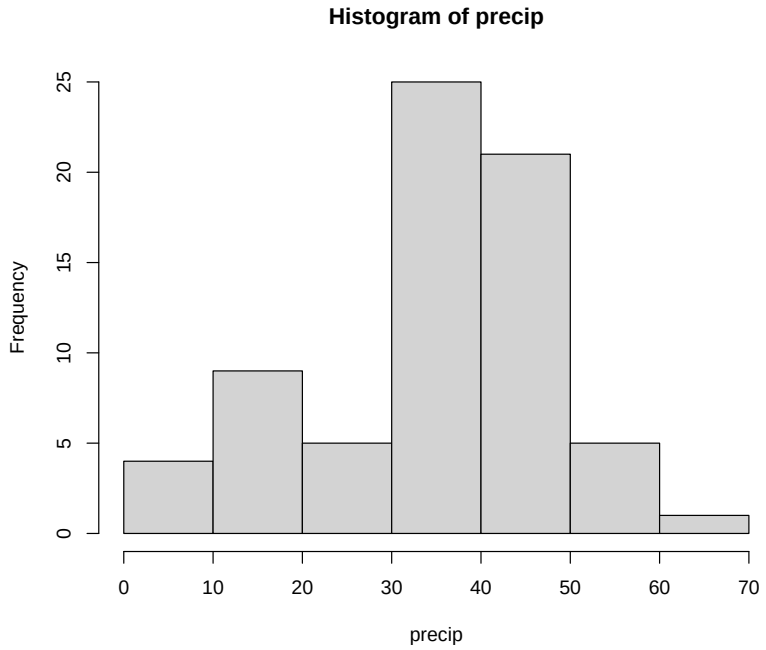
Veamos ahora un ejemplo de un análisis exploratorio muy básico (de una variable numérica). En el siguiente código:

- Se carga el objeto `precip` (uno de los conjuntos de datos de ejemplo disponibles en el paquete base de `R`) que contiene el promedio de precipitación, en pulgadas de lluvia, de 70 ciudades de Estados Unidos.
- Se hace un resumen estadístico de los datos.
- Se hace el correspondiente histograma y gráfico de cajas.

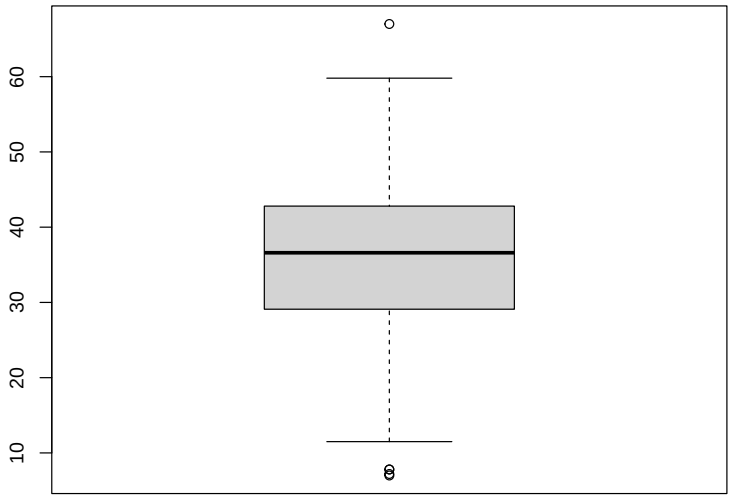
```
data(precip)    # Datos de lluvia
# ?precip       # Mostrar ayuda?
# precip        # Imprimir?
summary(precip) # Resumen estadístico
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    7.00   29.38   36.60   34.89   42.77   67.00
```

```
hist(precip)    # Histograma
```




```
boxplot(precip) # Gráfico de cajas
```



1.6 Funciones y librerías (paquetes)

Al iniciar el programa `R` se cargan por defecto una serie de librerías básicas con las que se pueden realizar una gran cantidad de operaciones empleando las funciones que implementan. Estas librerías conforman el llamado **paquete base**.

En otras ocasiones es necesario cargar librerías adicionales, empleando los denominados paquetes (packages). Normalmente se emplean los disponibles en el repositorio CRAN oficial http://cran.r-project.org/web/packages/.

1.6.1 Funciones internas

Las llamadas a una función son de la forma `nombre_función(argumento1, argumento2, ...)` y típicamente al evaluarlas devuelven un objeto con los resultados (o generan un gráfico). Los argumentos pueden tener nombres (se asignan por posición ó nombre) y valores por defecto (solo es necesario especificarlos para asignarles un valor distinto). Un nombre seguido de paréntesis hace siempre referencia a una función (realmente es un tipo de objeto y si por ejemplo se introduce solo el nombre en la línea de comandos simplemente se imprime el código).

```
x <- sin(pi/2)
# La función `sin()` y el objeto `pi` están en el paquete base
cat("El objeto x contiene:", x, "\n")
```

```
## El objeto x contiene: 1
```

El parámetro `...` aglutina los argumentos no definidos explícitamente (cuando la función puede operar sobre múltiples argumentos, e.g. `cat(...)`, o para poder incluir parámetros de otra función a la que se llama internamente).

Algunas funciones se comportan de manera diferente dependiendo del tipo de objeto (la clase) de sus argumentos, son lo que se denominan *funciones genéricas*. Entre ellas `summary()`, `print()`, `plot()` (por ejemplo, al ejecutar `methods(plot)` se muestran los métodos asociados esta función; el método por defecto es `plot.default()`).

1.6.2 Paquetes

La instalación de un paquete se puede hacer de varias formas:

- Desde la interfaz de comandos utilizando la instrucción
  ```
  install.packages("nombre del paquete")
  ```
- Desde el correspondiente menú de la intefaz gráfica (*Paquetes* > *Instalar paquete(s)…* en la consola de R y *Tools* > *Install packages…* o la pestaña *Packages* en RStudio).

Este proceso sólo es necesario realizarlo la primera vez que se utilice el paquete.

Para utilizar un paquete ya instalado será necesario cargarlo, lo cual se puede hacer de varias formas:

- Desde el menú *Paquetes* > *Cargar paquete(s)…*
- Por consola, utilizando `library(paquete)` (también `require(paquete)`)

Esta operación será necesario realizarla cada vez que se inicie una sesión de R.

Finalmente, la ayuda de un paquete se puede obtener con la sentencia

```
library(help = "nombre del paquete")
```

1.7 Objetos básicos

`R` es un lenguaje **orientado a objetos** lo que significa que las variables, datos, funciones, resultados, etc., se guardan en la memoria del ordenador en forma de *objetos* con un nombre específico.

Los principales tipos de valores básicos de `R` son:

- numéricos,
- cadenas de caracteres, y
- lógicos

1.7.1 Objetos numéricos

Los valores numéricos adoptan la notación habitual en informática: punto decimal, notacion científica, ...

```
pi
```

```
## [1] 3.141593
```

```
1e3
```

```
## [1] 1000
```

Con este tipo de objetos se pueden hacer operaciones aritméticas utilizando el operador correspondiente.

```
a <- 3.4
b <- 4.5
a * b
```

```
## [1] 15.3
```

```
a / b
```

```
## [1] 0.7555556
```

```
a + b
```

```
## [1] 7.9
```

```
min(a, b)
```

```
## [1] 3.4
```

1.7.2 Objetos tipo carácter

Las cadenas de caracteres se introducen delimitadas por comillas ("nombre") o por apóstrofos ('nombre').

```
a <- "casa grande"
a
```

```
## [1] "casa grande"
```

```
a <- 'casa grande'
a
```

```
## [1] "casa grande"
```

```
a <- 'casa "grande"'
a
```

```
## [1] "casa \"grande\""
```

1.7.3 Objetos lógicos

Los objetos lógicos sólo pueden tomar dos valores `TRUE` (numéricamente toma el valor 1) y `FALSE` (valor 0).

```
A <- TRUE
B <- FALSE
A
```

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

```
B
```

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

```
# valores numéricos
as.numeric(A)
```

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

```
as.numeric(B)
```

```
## [1] 0
```

1.7.4 Operadores lógicos

Existen varios operadores en R que devuelven un valor de tipo lógico. Veamos algún ejemplo

```
a <- 2
b <- 3
a == b  # compara a y b
```

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

```
a == a  # compara a y a
```

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

```
a < b
```

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

```
b < a
```

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

```
! (b < a) # ! niega la condición
```

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

```
2**2 == 2^2
```

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

```
3*2 == 3^2
```

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

Nótese la diferencia entre = (asignación) y == (operador lógico)

```
2 == 3
```

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

```
# 2 = 3 # produce un error:
# Error en 2 = 3 : lado izquierdo de la asignación inválida (do_set)
```

Se pueden encadenar varias condiciones lógicas utiilizando los operadores & (y lógico) y | (o lógico).

```
TRUE & TRUE
```

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

```
TRUE | TRUE
```

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

```
TRUE & FALSE
```

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

```
TRUE | FALSE
```

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

```
2 < 3 & 3 < 1
```

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

```
2 < 3 | 3 < 1
```

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

1.8 Área de trabajo

Como ya se ha comentado con anterioridad es posible guardar los comandos que se han utilizado en una sesión en ficheros llamados **script**. En ocasiones interesará además guardar todos los objetos que han sido generados a lo largo de una sesión de trabajo.

El **Workspace** o **Área de Trabajo** es el entorno en el que se almacenan todos los objetos creados en una sesión. Se puede guardar este entorno en el disco de forma que la próxima vez que se inicie el programa, al cargar dicho entorno, se pueda acceder a lo objetos almacenados en él.

En primer lugar, para saber los objetos que tenemos en memoria se utiliza la función `ls()`. Por ejemplo, supongamos que acabamos de iniciar una sesión de `R` y hemos escrito

```
a <- 1:10
b <- log(50)
```

Entonces al utilizar `ls()` se obtendrá la siguiente lista de objetos en memoria

```
ls()
```

```
## [1] "a" "b"
```

Los objetos se pueden eliminar empleando la función `rm()`.

```
rm(b)
ls()
```

```
## [1] "a"
```

Para borrar todos los objetos en memoria se puede utilizar `rm(list=ls())`.

```
rm(list = ls())
```

```
## character(0)
```

`character(0)` (cadena de texto vacía) significa que no hay objetos en memoria.

1.8.1 Guardar y cargar objetos

Para guardar el área de trabajo (Workspace) con todos los objetos de memoria (es decir, los que figuran al utilizar `ls()`) se utiliza la función `save.image(nombre archivo)`.

```
rm(list = ls()) # borramos todos los objetos en memoria
x <- 20
y <- 34
z <- "casa"
save.image(file = "prueba.RData") # guarda área de trabajo en prueba.RData
```

La función `save()` permite guardar los objetos especificados.

```
save(x, y, file = "prueba2.RData") # guarda los objetos x e y
```

Para cargar los objetos almacenados en un archivo se utiliza la función `load()`.

```
load("prueba2.RData") # carga los objetos x e y
```

1.8.2 Directorio de trabajo

Por defecto `R` utiliza una carpeta de trabajo donde guardará toda la información. Dicha carpeta se puede obtener con la función

```
getwd()
```

```
## [1] "d:/"
```

El directorio de trabajo se puede cambiar utilizando `setwd(directorio)`. Por ejemplo, para cambiar el directorio de trabajo a `c:\datos`, se utiliza el comando

```
setwd("c:/datos")
# Importante podemos emplear '/' o '\\' como separador en la ruta
# NO funciona setwd("c:\datos")
```

Capítulo 2

Estructuras de datos

En los ejemplos que hemos visto hasta ahora los objetos de `R` almacenaban un único valor cada uno. Sin embargo, las estructuras de datos que proporciona `R` permiten almacenar en un mismo objeto varios valores. Las principales estructuras son:

- Vectores
- Matrices y Arrays
- Data Frames
- Listas

2.1 Vectores

Un vector es un conjunto de valores básicos del mismo tipo. La forma más sencilla de crear vectores es a través de la función `c()` que se usa para combinar (concatenar) valores.

```
x <- c(3, 5, 7)
x
```

```
## [1] 3 5 7
```

```
y <- c(8, 9)
y
```

```
## [1] 8 9
```

```
c(x, y)
```

```
## [1] 3 5 7 8 9
```

```
z <- c("Hola", "Adios")
z
```

```
## [1] "Hola"  "Adios"
```

2.1.1 Generación de secuencias

Existen varias funciones que pemiten obtener secuencias de números

```
x <- 1:5
x
```

```
## [1] 1 2 3 4 5
```

```
seq(1, 5, 0.5)
```

```
## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

```
seq(from=1, to=5, length=9)
```

```
## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

```
rep(1, 5)
```

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

2.1.2 Generación secuencias aleatorias

A continuación se obtiene una simulación de 10 lanzamientos de un dado

```
sample(1:6, size=10, replace = T) # lanzamiento de un dado
```

```
##  [1] 6 3 1 1 5 5 5 3 6 6
```

Para simular el lanzamiento de una moneda podemos escribir

```
resultado <- c(cara = 1, cruz = 0) # se asignan nombres a los componentes
print(resultado)
```

```
## cara cruz
##    1    0
```

```
class(resultado)
```

```
## [1] "numeric"
```

```
attributes(resultado)
```

```
## $names
## [1] "cara" "cruz"
```

```
names(resultado)
```

```
## [1] "cara" "cruz"
```

```
lanz <- sample(resultado, size=10, replace = T)
lanz
```

```
## cara cruz cara cruz cara cara cara cara cara cruz
##    1    0    1    0    1    1    1    1    1    0
```

```
table(lanz)
```

```
## lanz
## 0 1
## 3 7
```

Otros ejemplos

```
rnorm(10)  # rnorm(10, mean = 0, sd = 1)
```

```
##  [1]  1.06057423 -1.13112660 -0.55005465 -0.06400286  0.37460541  0.90174865
##  [7]  0.60904681 -0.06503237 -0.72623274  0.99699218
```

```
runif(15, min = 2, max = 10)
```

```
##  [1] 4.133106 6.093477 7.363590 5.900827 7.166668 4.672351 8.618037 3.510272
##  [9] 9.658221 5.722486 5.455435 2.315432 3.362482 8.963233 7.090975
```

Como ya se comentó, se puede utilizar `help(funcion)` (o `?funcion`) para mostrar la ayuda de las funciones anteriores.

2.1.3 Selección de elementos de un vector

Para acceder a los elementos de un vector se indica entre corchetes el correspondiente vector de subíndices (enteros positivos).

```
x <- seq(-3, 3, 1)
x
```

```
## [1] -3 -2 -1  0  1  2  3
```

```
x[1]  # primer elemento
```

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

```
ii <- c(1, 5, 7)
x[ii] # posiciones 1, 5 y 7
```

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

```
ii <- x > 0
ii
```

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

```
x[ii]  # valores positivos
```

```
## [1] 1 2 3
```

```
ii <- 1:3
x[-ii]  # elementos de x salvo los 3 primeros
```

```
## [1] 0 1 2 3
```

2.1.4 Ordenación de vectores

```
x <- c(65, 18, 59, 18, 6, 94, 26)
sort(x)
```

```
## [1]  6 18 18 26 59 65 94
```

```
sort(x, decreasing = T)
```

```
## [1] 94 65 59 26 18 18  6
```

Otra posibilidad es utilizar un índice de ordenación.

```
ii <- order(x)
ii  # índice de ordenación
```

```
## [1] 5 2 4 7 3 1 6
```

```
x[ii]  # valores ordenados
```

```
## [1]  6 18 18 26 59 65 94
```

La función `rev()` devuelve los valores del vector en orden inverso.

```
rev(x)
```

```
## [1] 26 94  6 18 59 18 65
```

2.1.5 Datos faltantes

Los datos faltantes (también denominados valores perdidos) aparecen normalmente cuando algún dato no ha sido registrado. Este tipo de valores se registran como `NA` (abreviatura de *Not Available*).

Por ejemplo, supongamos que tenemos registrado las alturas de 5 personas pero desconocemos la altura de la cuarta persona. El vector sería registrado como sigue:

```
altura <- c(165, 178, 184, NA, 175)
altura
```

```
## [1] 165 178 184  NA 175
```

Es importante notar que cualquier operación aritmética sobre un vector que contiene algún `NA` dará como resultado otro `NA`.

```
mean(altura)
```

```
## [1] NA
```

En muchas funciones para forzar a `R` a que ignore los valores perdidos se utiliza la opción `na.rm = TRUE`.

```
mean(altura, na.rm = TRUE)
```

```
## [1] 175.5
```

`R` permite gestionar otros tipos de valores especiales:

- `NaN` (*Not a Number*): es resultado de una indeterminación.
- `Inf`: `R` representa valores no finitos $\pm\infty$ como `Inf` y `-Inf`.

```
5/0  # Infinito
```

```
## [1] Inf
```

```
log(0)  # -Infinito
```

```
## [1] -Inf
```

```
0/0  # Not a Number
```

```
## [1] NaN
```

2.1.6 Vectores no numéricos

Los vectores pueden ser no numéricos, aunque todas las componentes deben ser del mismo tipo:

```
a <- c("A Coruña", "Lugo", "Ourense", "Pontevedra")
a
```

```
## [1] "A Coruña"   "Lugo"       "Ourense"    "Pontevedra"
```

```
letters[1:10]  # primeras 10 letas del abecedario
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

```
LETTERS[1:10]  # lo mismo en mayúscula
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J"
```

```
month.name[1:6]  # primeros 6 meses del año en inglés
```

```
## [1] "January"  "February" "March"    "April"    "May"      "June"
```

2.1.7 Factores

Los factores se utilizan para representar datos categóricos. Se puede pensar en ellos como vectores de enteros en los que cada entero tiene asociada una etiqueta (*label*). Los factores son muy importantes en la modelización estadística ya que `R`los trata de forma especial.

Utilizar factores con etiquetas es preferible a utilizar enteros porque las etiquetas son auto-descriptivas.

Veamos un ejemplo. Supongamos que el vector `sexo` indica el sexo de un persona codificado como 0 si hombre y 1 si mujer

```
sexo <- c(0, 1, 1, 1, 0, 0, 1, 0, 1)
sexo
```

```
## [1] 0 1 1 1 0 0 1 0 1
```

```
table(sexo)
```

```
## sexo
## 0 1
## 4 5
```

El problema de introducir así los valores es que no queda reflejada la codificación de los mismos. Para ello guardaremos los datos en una estructura tipo factor:

```
sexo2 <- factor(sexo, labels = c("hombre", "mujer")); sexo2
```

```
## [1] hombre mujer  mujer  mujer  hombre hombre mujer  hombre mujer
## Levels: hombre mujer
```

```
levels(sexo2)  # devuelve los niveles de un factor
```

```
## [1] "hombre" "mujer"
```

```
unclass(sexo2)  # representación subyacente del factor
```

```
## [1] 1 2 2 2 1 1 2 1 2
## attr(,"levels")
## [1] "hombre" "mujer"
```

```
table(sexo2)
```

```
## sexo2
## hombre  mujer
##      4      5
```

Veamos otro ejemplo, en el que inicialmente tenemos datos categóricos. Los niveles se toman automáticamente por orden alfabético

```
respuestas <- factor(c('si', 'si', 'no', 'si', 'si', 'no', 'no'))
respuestas
```

```
## [1] si si no si si no no
## Levels: no si
```

Si deseásemos otro orden (lo cual puede ser importante en algunos casos, por ejemplo para representaciones gráficas), habría que indicarlo expresamente

```
respuestas <- factor(c('si', 'si', 'no', 'si', 'si', 'no', 'no'), levels = c('si', 'no'))
respuestas
```

```
## [1] si si no si si no no
## Levels: si no
```

2.2 Matrices y arrays

2.2.1 Matrices

Las *matrices* son la extensión natural de los vectores a dos dimensiones. Su generalización a más dimensiones se llama *array*.

Las matrices se pueden crear concatenando vectores con las funciones `cbind()` o `rbind()`:

```
x <- c(3, 7, 1, 8, 4)
y <- c(7, 5, 2, 1, 0)
cbind(x, y)  # por columnas
```

```
##      x y
## [1,] 3 7
## [2,] 7 5
## [3,] 1 2
## [4,] 8 1
## [5,] 4 0
```

```
rbind(x, y)  # por filas
```

```
##   [,1] [,2] [,3] [,4] [,5]
## x    3    7    1    8    4
## y    7    5    2    1    0
```

Una matriz se puede crear con la función `matrix()` donde el parámetro `nrow` indica el número de filas y `ncol` el número de columnas. Por defecto, los valores se colocan por columnas.

```
matrix(1:8, nrow = 2, ncol = 4)  # equivalente a matrix(1:8, nrow=2)
```

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

Los nombres de los parámetros se pueden acortar siempre y cuando no haya ambigüedad, por lo que podríamos escribir

```
matrix(1:8, nr = 2, nc = 4)
```

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

Si queremos indicar que los valores se almacenen por filas

```
matrix(1:8, nr = 2, byrow = TRUE)
```

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

2.2.2 Nombres en matrices

Se pueden dar nombres a las filas y columnas de una matriz.

```
x <- matrix(c(1, 2, 3, 11, 12, 13), nrow = 2, byrow = TRUE)
x
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]   11   12   13
```

```
rownames(x) <- c("fila 1", "fila 2")
colnames(x) <- c("col 1", "col 2", "col 3")
x
```

```
##        col 1 col 2 col 3
## fila 1     1     2     3
## fila 2    11    12    13
```

Obtenemos el mismo resultado si escribimos

```
colnames(x) <- paste("col", 1:ncol(x), sep=" ")
```

Internamente, las matrices son vectores con un atributo especial: la *dimensión*.

```
dim(x)
```

```
## [1] 2 3
```

```
attributes(x)
```

```
## $dim
## [1] 2 3
##
## $dimnames
## $dimnames[[1]]
## [1] "fila 1" "fila 2"
##
## $dimnames[[2]]
## [1] "col 1" "col 2" "col 3"
```

2.2.3 Acceso a los elementos de una matriz

El acceso a los elementos de una matriz se realiza de forma análoga al acceso ya comentado para los vectores.

```
x <- matrix(1:6, 2, 3); x
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```
x[1, 1]
```

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

```
x[2, 2]
```

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

```
x[2, ]  # segunda fila
```

```
## [1] 2 4 6
```

```
x[ ,2]  # segunda columna
```

```
## [1] 3 4
```

```
x[1, 1:2]  # primera fila, columnas 1ª y 2ª
```

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

2.2.4 Ordenación por filas y columnas

En ocasiones, interesará ordenar los elementos de una matriz por los valores de una determinada columna o fila.

Por ejemplo, supongamos la matriz

```
x <- c(79, 100, 116, 121, 52, 134, 123, 109, 80, 107, 66, 118)
x <- matrix(x, ncol=4, byrow=T); x
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   79  100  116  121
```

```
## [2,]   52  134  123  109
## [3,]   80  107   66  118
```

La matriz ordenada por los valores de la primera columna viene dada por

```
ii <- order(x[ ,1])
x[ii, ]  # ordenación columna 1
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   52  134  123  109
## [2,]   79  100  116  121
## [3,]   80  107   66  118
```

De igual modo, si queremos ordenar por los valores de la cuarta columna:

```
ii <- order(x[ ,4]); x[ii, ]  # ordenación columna 4
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   52  134  123  109
## [2,]   80  107   66  118
## [3,]   79  100  116  121
```

2.2.5 Operaciones con Matrices y Arrays

A continuación se muestran algunas funciones que se pueden emplear con matrices

Función	Descripción
`dim(), nrow(), ncol()`	número de filas y/o columnas
`diag()`	diagonal de una matrix
`*`	multiplicación elemento a elemento
`%*%`	multiplicación matricial de matrices
`cbind(), rbind()`	encadenamiento de columnas o filas
`t()`	transpuesta
`solve(A)`	inversa de la matriz A
`solve(A,b)`	solución del sistema de ecuaciones $Ax = b$
`qr()`	descomposición de Cholesky
`eigen()`	autovalores y autovectores
`svd()`	descomposición singular

2.2.6 Ejemplos

```
x <- matrix(1:6, ncol = 3)
x
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```
t(x)  # matriz transpuesta
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6
```

```
dim(x)  # dimensiones de la matriz
```

```
## [1] 2 3
```

2.2.7 Inversión de una matriz

```
A <- matrix(c(2, 4, 0, 2), nrow = 2); A
```

```
##      [,1] [,2]
## [1,]    2    0
## [2,]    4    2
```

```
B <- solve(A)
B  # inversa
```

```
##      [,1] [,2]
## [1,]  0.5  0.0
## [2,] -1.0  0.5
```

```
A %*% B  # comprobamos que está bien
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

2.3 Data frames

Los `data.frames` (*marcos de datos*) son el objeto más habitual para el almacenamiento de conjuntos de datos. En este tipo de objetos cada individuo de la muestra se corresponde con una fila y cada una de las variables con una columna. Para la creación de estas estructuras se utiliza la función `data.frame()`.

Este tipo de estructuras son en apariencia muy similares a las matrices, con la ventaja de que permiten que los valores de las distintas columnas sean de tipos diferentes. Por ejemplo, supongamos que tenemos registrados los siguientes valores

```
Producto <- c("Zumo", "Queso", "Yogourt")
Seccion <- c("Bebidas", "Lácteos", "Lácteos")
Unidades <- c(2, 1, 10)
```

Los valores anteriores se podrían guardar en una única matriz

```
x <- cbind(Producto, Seccion, Unidades)
class(x)
```

```
## [1] "matrix" "array"
```

```
x
```

```
##      Producto  Seccion   Unidades
## [1,] "Zumo"    "Bebidas" "2"
## [2,] "Queso"   "Lácteos" "1"
## [3,] "Yogourt" "Lácteos" "10"
```

Sin embargo, el resultado anterior no es satisfactorio ya que todos los valores se han transformado en caracteres. Una solución mejor es utilizar un `data.frame`, con lo cual se mantiene el tipo original de las variables.

```
lista.compra <- data.frame(Producto, Seccion, Unidades)
class(lista.compra)
```

```
## [1] "data.frame"
```

```
lista.compra
```

```
##   Producto Seccion Unidades
## 1     Zumo Bebidas        2
```

```
## 2    Queso Lácteos        1
## 3  Yogourt Lácteos       10
```

A continuación se muestran ejemplos que ilustran la manera de acceder a los valores de un data.frame.

```
lista.compra$Unidades
```

```
## [1]  2  1 10
```

```
lista.compra[ ,3]  # de manera equivalente
```

```
## [1]  2  1 10
```

```
lista.compra$Seccion
```

```
## [1] "Bebidas" "Lácteos" "Lácteos"
```

```
lista.compra$Unidades[1:2]  # primeros dos valores de Unidades
```

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

```
lista.compra[2,]  # segunda fila
```

```
##   Producto Seccion Unidades
## 2    Queso Lácteos        1
```

La función `summary()` permite hacer un resumen estadístico de las variables (columnas) del data.frame.

```
summary(lista.compra)
```

```
##    Producto            Seccion            Unidades
##  Length:3           Length:3           Min.   : 1.000
##  Class :character   Class :character   1st Qu.: 1.500
##  Mode  :character   Mode  :character   Median : 2.000
##                                        Mean   : 4.333
##                                        3rd Qu.: 6.000
##                                        Max.   :10.000
```

2.4 Listas

Las listas son colecciones ordenadas de cualquier tipo de objetos (en `R` las listas son un tipo especial de vectores). Así, mientras que los elementos de los vectores, matrices y arrays deben ser del mismo tipo, en el caso de las listas se pueden tener elementos de tipos distintos.

```
x <- c(1, 2, 3, 4)
y <- c("Hombre", "Mujer")
z <- matrix(1:12, ncol = 4)
datos <- list(A = x, B = y, C = z)
datos
```

```
## $A
## [1] 1 2 3 4
##
## $B
## [1] "Hombre" "Mujer"
##
## $C
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

Capítulo 3

Gráficos

En el paquete base de `R` se dispone de múltiples funciones que permiten la generación de gráficos (los denominados gráficos estándar). Se dividen en dos grandes grupos:

- Gráficos de **alto nivel**: Crean un gráfico nuevo.
  - `plot`, `hist`, `boxplot`, …
- Gráficos de **bajo nivel**: Permiten añadir elementos (líneas, puntos, …) a un gráfico ya existente
  - `points`, `lines`, `legend`, `text`, …

El parámetro `add = TRUE` convierte una función de nivel alto a bajo.

Dentro de las funciones gráficas de alto nivel destaca la función `plot()` que tiene muchas variantes y dependiendo del tipo de datos que se le pasen como argumento actuará de modo distinto (es una *función genérica*, `methods(plot)` devuelve los métodos disponibles).

3.1 La función plot

Si ejecutamos `plot(x, y)` siendo `x` e `y` vectores, entonces `R` generará el denominado **gráfico de dispersión** que representa en un sistema coordenado los pares de valores (x, y).

Por ejemplo, utilizando el siguiente código

```
data(cars)
plot(cars$speed, cars$dist)    # otra posibilidad plot(cars)
```

[Figura 3.1]

El comando `plot` incluye por defecto una elección automáticas de títulos, ejes, escalas, etiquetas, etc., que pueden ser modificados añadiendo parámetros gráficos al comando:

Parámetro	Descripción
`type`	tipo de gráfico: `p`: puntos, `l`: líneas, `b`: puntos y líneas, `n`: gráfico en blanco…
`xlim`, `ylim`	límites de los ejes (e.g. `xlim=c(1, 10)` o `xlim=range(x)`)
`xlab`, `ylab`	títulos de los ejes
`main, sub`	título principal y subtítulo
`col`	color de los símbolos (véase `colors()`). También `col.axis`, `col.lab`, `col.main`, `col.sub`
`lty`	tipo de línea
`lwd`	anchura de línea
`pch`	tipo de símbolo
`cex`	tamaño de los símbolos

Parámetro	Descripción
`bg`	color de relleno (para `pch = 21:25`)

Para obtener ayuda sobre estos parámetros ejecutar `help(par)`.

Veamos algún ejemplo:

```
plot(cars, xlab = "velocidad", ylab = "distancia", main = "Título")
```

[Figura 3.2]

```
plot(cars, pch = 16, col = 'blue', main = 'pch=16')
```

[Figura 3.3]

3.2 Funciones gráficas de bajo nivel

Las principales funciones gráficas de bajo nivel son:

Función	Descripción
`points`, `lines`	agregan puntos y líneas
`text`	agrega un texto
`mtext`	agrega texto en los márgenes
`segments`	dibuja trozos de líneas desde puntos iniciales a finales
`abline`	dibuja líneas
`rect`	dibuja rectángulos
`polygon`	dibuja polígonos
`legend`	agrega una leyenda
`axis`	agrega ejes
`locator`	devuelve coordenadas de puntos
`identify`	similar a `locator`

3.3 Ejemplos

```
plot(cars)
abline(h = c(20, 40), lty = 2) # líneas horizontales discontinuas (lty=2)
# selecciona puntos y los dibuja en azul sólido
points(subset(cars, dist > 20 & dist < 40), pch = 16, col = 'blue')
```

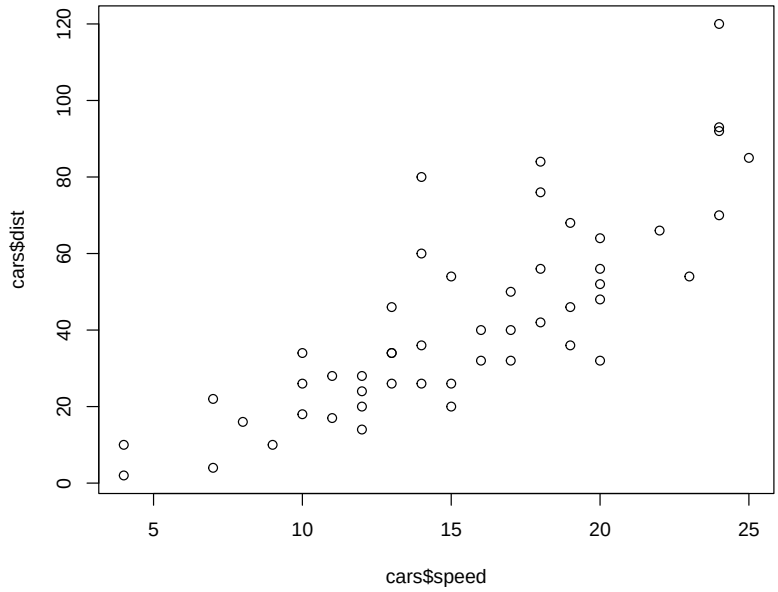


Figura 3.1: Gráfico de dispersión de distancia frente a velocidad

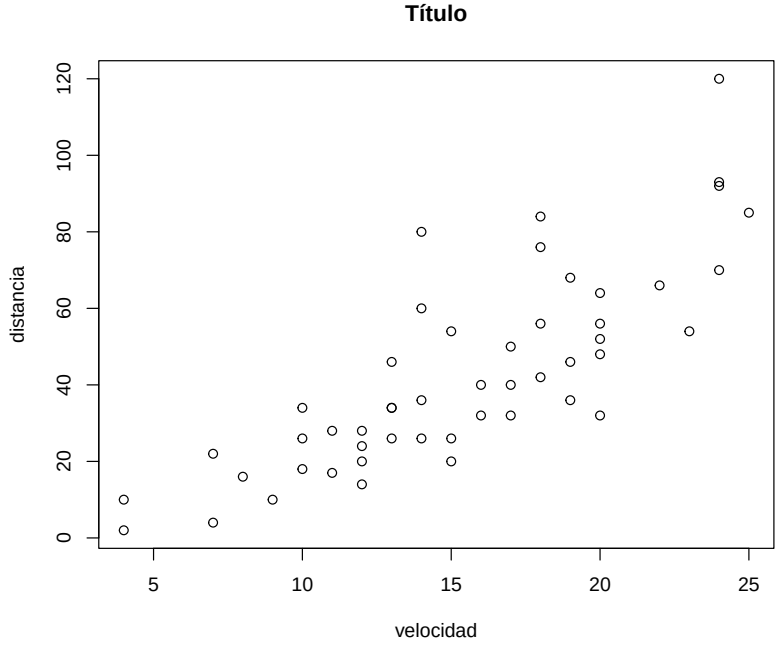
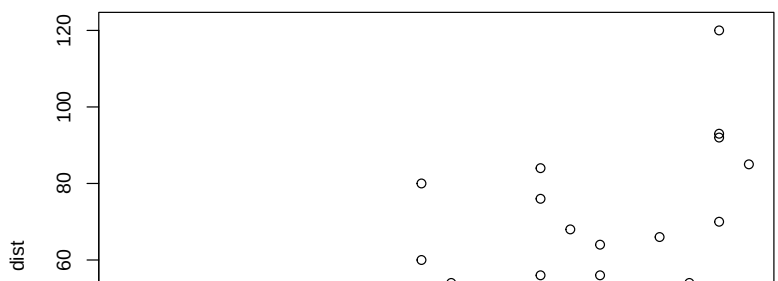


Figura 3.2: Gráfico de dispersión de distancia frente a velocidad, especificando título y etiquetas de los ejes



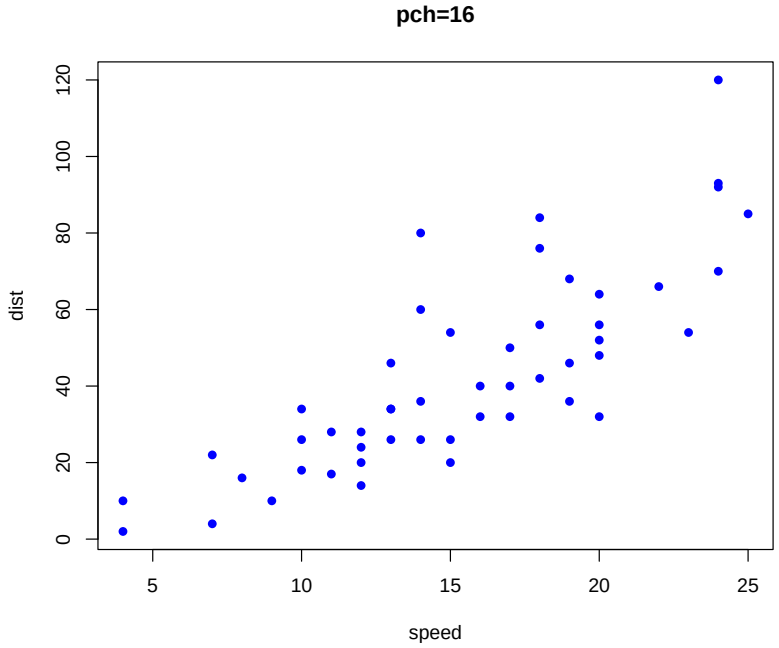
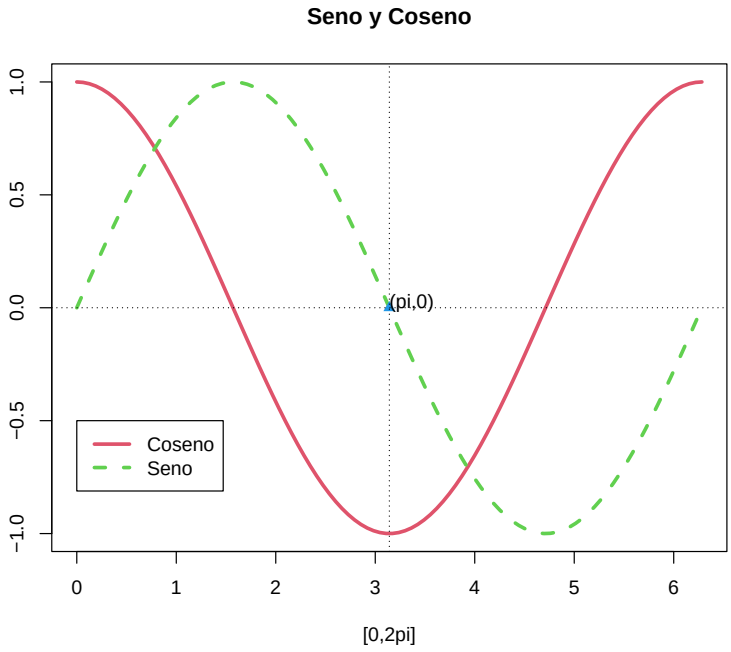


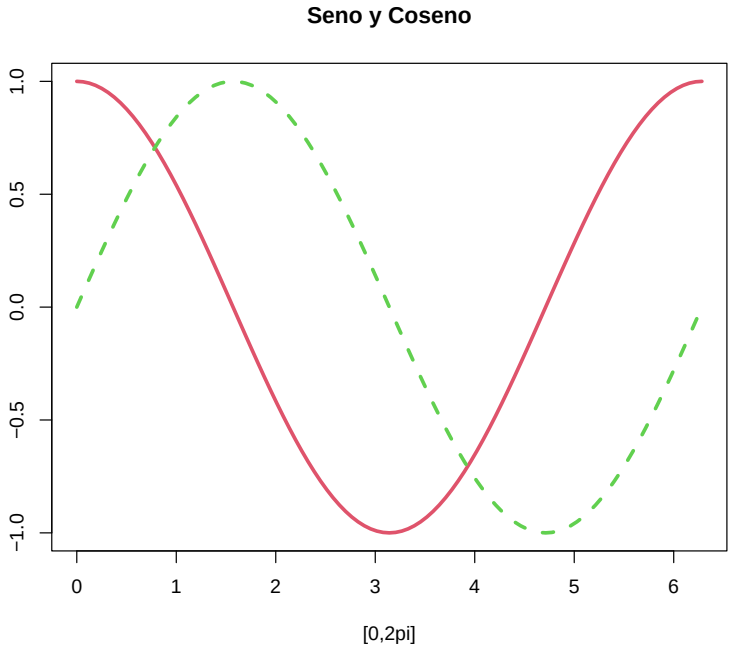
Figura 3.3: Gráfico de dispersión de distancia frente a velocidad, cambiando el color y el tipo de símbolo

```
x <- seq(0, 2 * pi, length = 100)
y1 <- cos(x)
y2 <- sin(x)
plot( x, y1, type = "l", col = 2, lwd = 3, xlab = "[0,2pi]", ylab = "", main = "Seno y Coseno")
lines(x, y2, col = 3, lwd = 3, lty = 2)
points(pi, 0, pch = 17, col = 4)
legend(0, -0.5, c("Coseno", "Seno"), col = 2:3, lty = 1:2, lwd = 3)

abline(v = pi, lty = 3)
abline(h = 0, lty = 3)
text(pi, 0, "(pi,0)", adj = c(0, 0))
```



Alternativamente se podría usar `curve()`:

```
curve(cos, 0, 2*pi, col = 2, lwd = 3,
      xlab = "[0,2pi]", ylab = "", main = "Seno y Coseno")
curve(sin, col = 3, lwd = 3, lty = 2, add = TRUE)
```



3.4 Parámetros gráficos

Como ya hemos visto, muchas funciones gráficas permiten establecer (temporalmente) opciones gráficas mediante estos parámetros. Con la función `par()` se pueden obtener y establecer (de forma permanente) todas las opciones gráficas. Algunas más de estas opciones son:

Parámetro	Descripción
`adj`	justificación del texto
`axes`	si es `FALSE` no dibuja los ejes ni la caja
`bg`	color del fondo
`bty`	tipo de caja alrededor del gráfico
`font`	estilo del texto (1: normal, 2: cursiva, 3:negrita, 4: negrita cursiva)
`las`	orientación de los caracteres en los ejes
`mar`	márgenes
`mfcol`	divide la pantalla gráfica por columnas
`mfrow`	lo mismo que `mfcol` pero por filas

Ejecutar `help(par)` para obtener la lista completa.

3.5 Múltiples gráficos por ventana

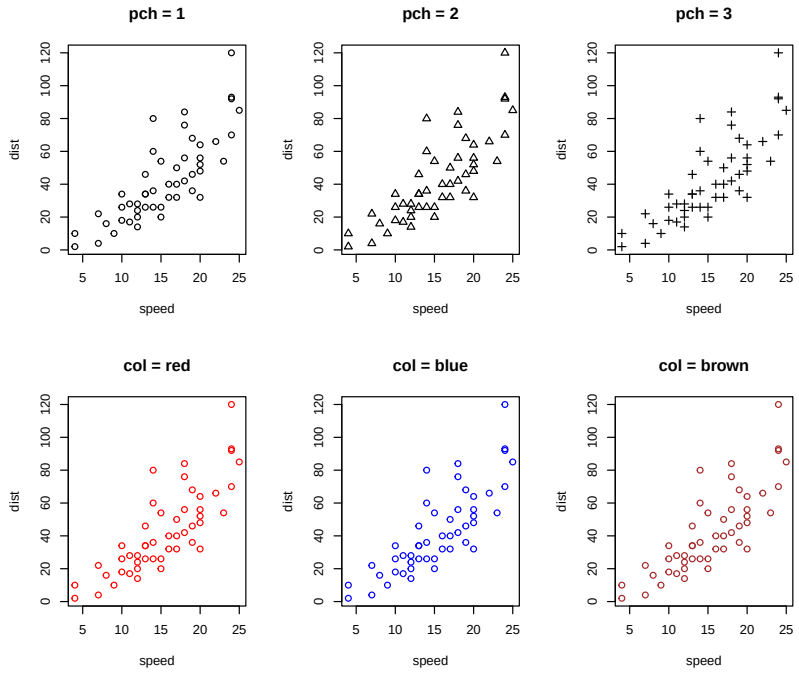
En `R` se pueden hacer varios gráficos por ventana. Para ello, antes de ejecutar la función `plot()`, se puede ejecutar:

```
par(mfrow = c(filas, columnas))
```

Los gráficos se irán mostrando en pantalla por filas. En caso de que se quieran mostrar por columnas en la función anterior se sustituye `mfrow` por `mfcol`.

Por ejemplo:

```
old.par <- par(mfrow = c(2, 3))
plot(cars, pch = 1, main = "pch = 1")
plot(cars, pch = 2, main = "pch = 2")
plot(cars, pch = 3, main = "pch = 3")

plot(cars, col = "red", main = "col = red")
plot(cars, col = "blue", main = "col = blue")
plot(cars, col = "brown", main = "col = brown")
```



```
par(old.par)
```

La función `par()` devuelve la anterior configuración de parámetros, lo que permite volverlos a establecer.

Para estructuras gráficas más complicadas véase `help(layout)`.

3.6 Exportar gráficos

Para guardar gráficos, en Windows, se puede usar el menú `Archivo -> Guardar como` de la ventana gráfica (seleccionando el formato deseado: bitmap, postscript,...) y también mediante código ejecutando `savePlot(filename, type)`. Alternativamente, se pueden emplear ficheros como dispositivos gráficos. Por ejemplo, a continuación guardamos un gráfico en el fichero *car.pdf*:

```
pdf("cars.pdf")    # abrimos el dispositivo gráfico
plot(cars)
dev.off()          # cerramos el dispositivo
```

Con el siguiente código guardaremos el gráfico en formato jpeg:

```
jgeg("cars.jpg")   # abrimos el dispositivo gráfico
plot(cars)
dev.off()          # cerramos el dispositivo
```

Otros formatos disponibles son `bmp`, `png` y `tiff`. Para más detalles ejecutar `help(Devices)`.

Sin embargo para exportar resultados, incluyendo gráficos, suele ser preferible emplear informes en RMarkdown cono se muestra en el Capítulo 12.

3.7 Otras librerías gráficas

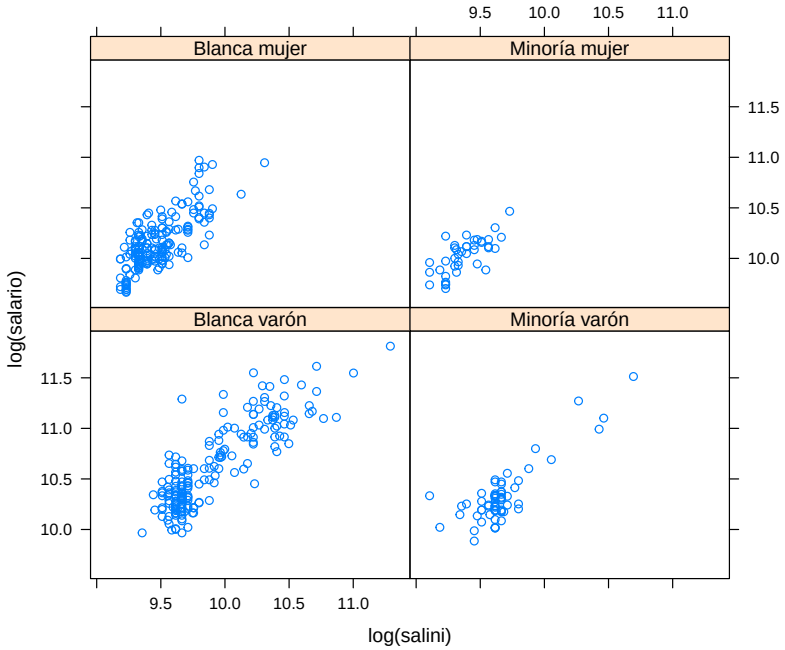
Además de los gráficos estándar, en `R` están disponibles muchas librerías gráficas adicionales:

- Gráficos Lattice (Trellis)
  - Especialmente adecuados para gráficas condicionales múltiples.
  - No se pueden combinar con las funciones estándar.
  - Generalmente el argumento principal es una formula:
    - `y ~ x | a` gráficas de `y` sobre `x` condicionadas por `a`
    - `y ~ x | a*b` gráficas condicionadas por `a` y `b`
  - Devuelven un objeto con el que se puede interactuar.
- ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics.
- rgl: 3D visualization device system for R using OpenGL.

Para más detalles ver CRAN Task View: Graphics

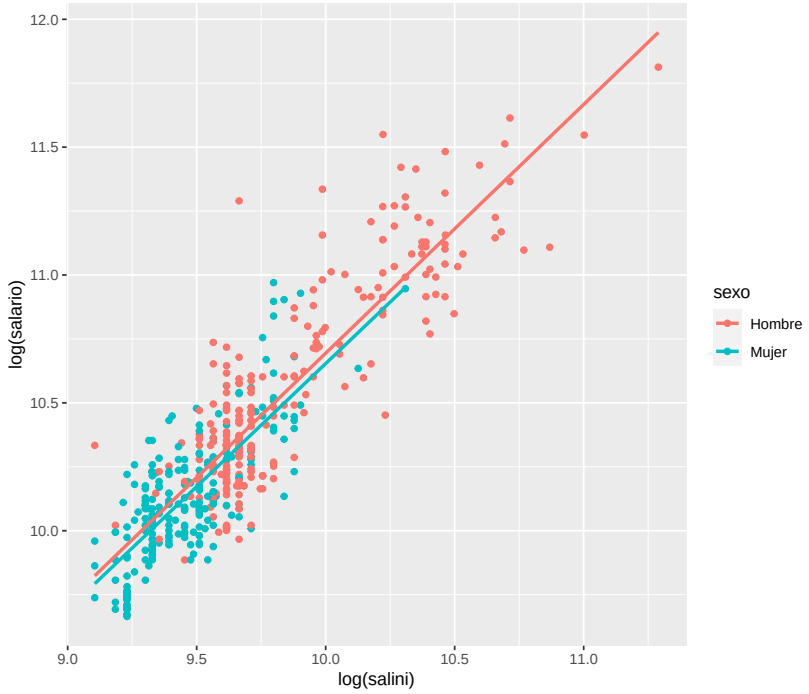
3.7.1 Ejemplos

```
load("datos/empleados.RData")
library(lattice)
xyplot(log(salario) ~ log(salini) | sexoraza, data = empleados)
```

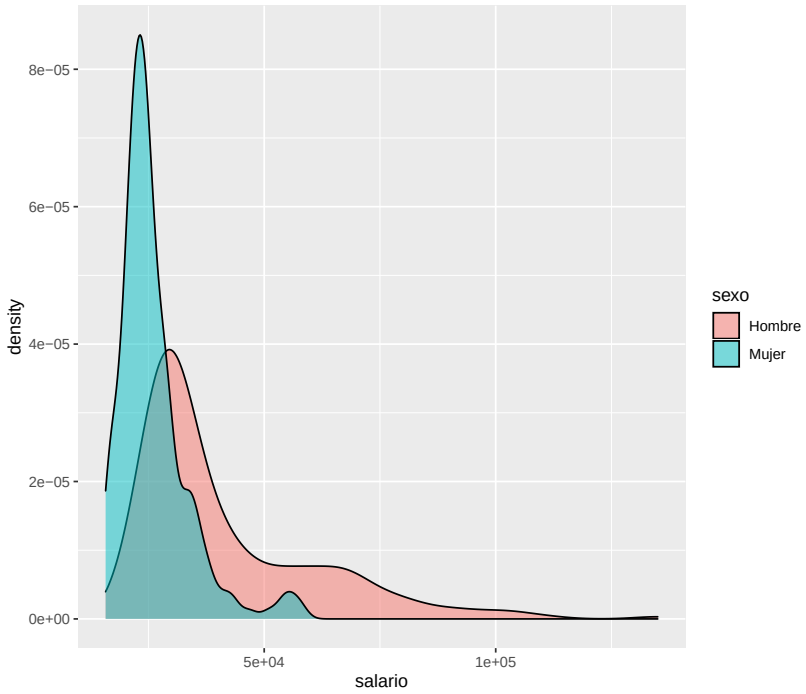



```
# Equivalente a xyplot(log(salario) ~ log(salini) | sexo*minoria, data = empleados)
```

```
library(ggplot2)
ggplot(empleados, aes(log(salini), log(salario), col = sexo)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)
```




```
ggplot(empleados, aes(salario, fill = sexo)) +
  geom_density(alpha=.5)
```


8e−05
6e−05
4e−05
2e−05
0e+00
density
5e+04
1e+05
salario
sexo
Hombre
Mujer

Capítulo 4

Manipulación de datos con R

La mayoría de los estudios estadísticos requieren disponer de un conjunto de datos.

4.1 Lectura, importación y exportación de datos

Además de la introducción directa, R es capaz de importar datos externos en múltiples formatos:

- bases de datos disponibles en librerías de R
- archivos de texto en formato ASCII
- archivos en otros formatos: Excel, SPSS, ...
- bases de datos relacionales: MySQL, Oracle, ...
- formatos web: HTML, XML, JSON, ...
- ....

4.1.1 Formato de datos de R

El formato de archivo en el que habitualmente se almacena objetos (datos) R es binario y está comprimido (en formato `"gzip"` por defecto). Para cargar un fichero de datos se emplea normalmente `load()`:

```
res <- load("datos/empleados.RData")
res
```

```
## [1] "empleados"
```

```
ls()
```

```
##  [1] "cite_fsimres" "cite_simres"  "citefig"      "citefig2"     "citepkg"
##  [6] "empleados"    "fig.path"     "inline"       "inline2"      "is_html"
## [11] "is_latex"     "latexfig"     "res"
```

y para guardar `save()`:

```
# Guardar
save(empleados, file = "datos/empleados_new.RData")
```

El objeto empleado normalmente en R para almacenar datos en memoria es el `data.frame`.

4.1.2 Acceso a datos en paquetes

R dispone de múltiples conjuntos de datos en distintos paquetes, especialmente en el paquete `datasets` que se carga por defecto al abrir R. Con el comando `data()` podemos obtener un listado de las bases

de datos disponibles.

Para cargar una base de datos concreta se utiliza el comando `data(nombre)` (aunque en algunos casos se cargan automáticamente al emplearlos). Por ejemplo, `data(cars)` carga la base de datos llamada `cars` en el entorno de trabajo (`".GlobalEnv"`) y `?cars` muestra la ayuda correspondiente con la descripición de la base de datos.

4.1.3 Lectura de archivos de texto

En R para leer archivos de texto se suele utilizar la función `read.table()`. Supóngase, por ejemplo, que en el directorio actual está el fichero *empleados.txt*. La lectura de este fichero vendría dada por el código:

```
# Session > Set Working Directory > To Source...?
datos <- read.table(file = "datos/empleados.txt", header = TRUE)
# head(datos)
str(datos)
```

```
## 'data.frame':    474 obs. of  10 variables:
##  $ id      : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ sexo    : chr  "Hombre" "Hombre" "Mujer" "Mujer" ...
##  $ fechnac : chr  "2/3/1952" "5/23/1958" "7/26/1929" "4/15/1947" ...
##  $ educ    : int  15 16 12 8 15 15 15 12 15 12 ...
##  $ catlab  : chr  "Directivo" "Administrativo" "Administrativo" "Administrativo" ...
##  $ salario : num  57000 40200 21450 21900 45000 ...
##  $ salini  : int  27000 18750 12000 13200 21000 13500 18750 9750 12750 13500 ...
##  $ tiempemp: int  98 98 98 98 98 98 98 98 98 98 ...
##  $ expprev : int  144 36 381 190 138 67 114 0 115 244 ...
##  $ minoria : chr  "No" "No" "No" "No" ...
```

Si el fichero estuviese en el directorio *c:\datos* bastaría con especificar `file = "c:/datos/empleados.txt"`. Nótese también que para la lectura del fichero anterior se ha establecido el argumento `header=TRUE` para indicar que la primera línea del fichero contiene los nombres de las variables.

Los argumentos utilizados habitualmente para esta función son:

- `header`: indica si el fichero tiene cabecera (`header=TRUE`) o no (`header=FALSE`). Por defecto toma el valor `header=FALSE`.
- `sep`: carácter separador de columnas que por defecto es un espacio en blanco (`sep=""`). Otras opciones serían: `sep=","` si el separador es un ";", `sep="*"` si el separador es un "*", etc.
- `dec`: carácter utilizado en el fichero para los números decimales. Por defecto se establece `dec = "."`. Si los decimales vienen dados por "," se utiliza `dec = ","`

Resumiendo, los (principales) argumentos por defecto de la función `read.table` son los que se muestran en la siguiente línea:

```
read.table(file, header = FALSE, sep = "", dec = ".")
```

Para más detalles sobre esta función véase `help(read.table)`.

Estan disponibles otras funciones con valores por defecto de los parámetros adecuados para otras situaciones. Por ejemplo, para ficheros separados por tabuladores se puede utilizar `read.delim()` o `read.delim2()`:

```
read.delim(file, header = TRUE, sep = "\t", dec = ".")
read.delim2(file, header = TRUE, sep = "\t", dec = ",")
```

4.1.4 Alternativa `tidyverse`

Para leer archivos de texto en distintos formatos también se puede emplear el paquete `readr` (colección `tidyverse`), para lo que se recomienda consultar el Capítulo 11 del libro R for Data Science.

4.1.5 Importación desde SPSS

El programa R permite lectura de ficheros de datos en formato SPSS (extensión *.sav*) sin necesidad de tener instalado dicho programa en el ordenador. Para ello se necesita:

- cargar la librería `foreign`
- utilizar la función `read.spss`

Por ejemplo:

```
library(foreign)
datos <- read.spss(file = "datos/Employee data.sav", to.data.frame = TRUE)
# head(datos)
str(datos)
```

```
## 'data.frame':    474 obs. of  10 variables:
##  $ id      : num  1 2 3 4 5 6 7 8 9 10 ...
##  $ sexo    : Factor w/ 2 levels "Hombre","Mujer": 1 1 2 2 1 1 1 2 2 2 ...
##  $ fechnac : num  1.17e+10 1.19e+10 1.09e+10 1.15e+10 1.17e+10 ...
##  $ educ    : Factor w/ 10 levels "8","12","14",..: 4 5 2 1 4 4 4 2 4 2 ...
##  $ catlab  : Factor w/ 3 levels "Administrativo",..: 3 1 1 1 1 1 1 1 1 1 ...
##  $ salario : Factor w/ 221 levels "15750","15900",..: 179 137 28 31 150 101 121 31 71 45 ...
##  $ salini  : Factor w/ 90 levels "9000","9750",..: 60 42 13 21 48 23 42 2 18 23 ...
##  $ tiempemp: Factor w/ 36 levels "63","64","65",..: 36 36 36 36 36 36 36 36 36 36 ...
##  $ expprev : Factor w/ 208 levels "Ausente","10",..: 38 131 139 64 34 181 13 1 14 91 ...
##  $ minoria : Factor w/ 2 levels "No","Sí": 1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, "variable.labels")= Named chr [1:10] "Código de empleado" "Sexo" "Fecha de nacimient
##   ..- attr(*, "names")= chr [1:10] "id" "sexo" "fechnac" "educ" ...
##  - attr(*, "codepage")= int 1252
```

Nota: Si hay fechas, puede ser recomendable emplear la función `spss.get()` del paquete `Hmisc`.

4.1.6 Importación desde Excel

Se pueden leer fichero de Excel (con extensión *.xlsx*) utilizando por ejemplo los paquetes `openxlsx`, `readxl` (colección `tidyverse`), `XLConnect` o `RODBC` (este paquete se empleará más adelante para acceder a bases de datos), entre otros.

Sin embargo, un procedimiento sencillo consiste en exportar los datos desde Excel a un archivo de texto separado por tabuladores (extensión *.csv*). Por ejemplo, supongamos que queremos leer el fichero *coches.xls*:

- Desde Excel se selecciona el menú `Archivo -> Guardar como -> Guardar como` y en `Tipo` se escoge la opción de archivo CSV. De esta forma se guardarán los datos en el archivo *coches.csv*.
- El fichero *coches.csv* es un fichero de texto plano (se puede editar con Notepad), con cabecera, las columnas separadas por ";", y siendo "," el carácter decimal.
- Por lo tanto, la lectura de este fichero se puede hacer con:

  ```
  datos <- read.table("coches.csv", header = TRUE, sep = ";", dec = ",")
  ```

Otra posibilidad es utilizar la función `read.csv2`, que es una adaptación de la función general `read.table` con las siguientes opciones:

```
read.csv2(file, header = TRUE, sep = ";", dec = ",")
```

Por lo tanto, la lectura del fichero *coches.csv* se puede hacer de modo más directo con:

```
datos <- read.csv2("coches.csv")
```

4.1.7 Exportación de datos

Puede ser de interés la exportación de datos para que puedan leídos con otros programas. Para ello, se puede emplear la función `write.table()`. Esta función es similar, pero funcionando en sentido inverso, a `read.table()` (Sección 4.1.3).

Veamos un ejemplo:

```
tipo <- c("A", "B", "C")
longitud <- c(120.34, 99.45, 115.67)
datos <- data.frame(tipo, longitud)
datos
```

```
##   tipo longitud
## 1    A   120.34
## 2    B    99.45
## 3    C   115.67
```

Para guardar el data.frame `datos` en un fichero de texto se puede utilizar:

```
write.table(datos, file = "datos.txt")
```

Otra posibilidad es utilizar la función:

```
write.csv2(datos, file = "datos.csv")
```

que dará lugar al fichero *datos.csv* importable directamente desde Excel.

4.2 Manipulación de datos

Una vez cargada una (o varias) bases de datos hay una series de operaciones que serán de interés para el tratamiento de datos:

- Operaciones con variables:
  - crear
  - recodificar (e.g. categorizar)
  - ...
- Operaciones con casos:
  - ordenar
  - filtrar
  - ...

A continuación se tratan algunas operaciones *básicas*.

4.2.1 Operaciones con variables

4.2.1.1 Creación (y eliminación) de variables

Consideremos de nuevo la base de datos `cars` incluida en el paquete `datasets`:

```
data(cars)
# str(cars)
head(cars)
```

```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
## 4     7   22
## 5     8   16
## 6     9   10
```

Utilizando el comando `help(cars)` se obtiene que `cars` es un data.frame con 50 observaciones y dos variables:

- `speed`: Velocidad (millas por hora)
- `dist`: tiempo hasta detenerse (pies)

Recordemos que, para acceder a la variable `speed` se puede hacer directamente con su nombre o bien utilizando notación "matricial".

```
cars$speed
```

```
##  [1]  4  4  7  7  8  9 10 10 10 11 11 12 12 12 12 13 13 13 13 14 14 14 14 15 15
## [26] 15 16 16 17 17 17 18 18 18 18 19 19 19 20 20 20 20 20 22 23 24 24 24 24 25
```

```
cars[, 1]  # Equivalente
```

```
##  [1]  4  4  7  7  8  9 10 10 10 11 11 12 12 12 12 13 13 13 13 14 14 14 14 15 15
## [26] 15 16 16 17 17 17 18 18 18 18 19 19 19 20 20 20 20 20 22 23 24 24 24 24 25
```

Supongamos ahora que queremos transformar la variable original `speed` (millas por hora) en una nueva variable `velocidad` (kilómetros por hora) y añadir esta nueva variable al data.frame `cars`. La transformación que permite pasar millas a kilómetros es `kilómetros=millas/0.62137` que en R se hace directamente con:

```
cars$speed/0.62137
```

Finalmente, incluimos la nueva variable que llamaremos `velocidad` en `cars`:

```
cars$velocidad <- cars$speed / 0.62137
head(cars)
```

```
##   speed dist velocidad
## 1     4    2  6.437388
## 2     4   10  6.437388
## 3     7    4 11.265430
## 4     7   22 11.265430
## 5     8   16 12.874777
## 6     9   10 14.484124
```

También transformaremos la variable `dist` (en pies) en una nueva variable `distancia` (en metros). Ahora la transformación deseada es `metros=pies/3.2808`:

```
cars$distancia <- cars$dis / 3.2808
head(cars)
```

```
##   speed dist velocidad distancia
## 1     4    2  6.437388 0.6096074
## 2     4   10  6.437388 3.0480371
## 3     7    4 11.265430 1.2192148
## 4     7   22 11.265430 6.7056815
## 5     8   16 12.874777 4.8768593
## 6     9   10 14.484124 3.0480371
```

Ahora, eliminaremos las variables originales `speed` y `dist`, y guardaremos el data.frame resultante con el nombre `coches`. En primer lugar, veamos varias formas de acceder a las variables de interés:

```
cars[, c(3, 4)]
cars[, c("velocidad", "distancia")]
cars[, -c(1, 2)]
```

Utilizando alguna de las opciones anteriores se obtiene el `data.frame` deseado:

```
coches <- cars[, c("velocidad", "distancia")]
# head(coches)
str(coches)
```

```
## 'data.frame':    50 obs. of  2 variables:
##  $ velocidad: num  6.44 6.44 11.27 11.27 12.87 ...
##  $ distancia: num  0.61 3.05 1.22 6.71 4.88 ...
```

Finalmente los datos anteriores podrían ser guardados en un fichero exportable a Excel con el siguiente comando:

```
write.csv2(coches, file = "coches.csv")
```

4.2.2 Operaciones con casos

4.2.2.1 Ordenación

Continuemos con el data.frame `cars`. Se puede comprobar que los datos disponibles están ordenados por los valores de `speed`. A continuación haremos la ordenación utilizando los valores de `dist`. Para ello utilizaremos el conocido como vector de índices de ordenación. Este vector establece el orden en que tienen que ser elegidos los elementos para obtener la ordenación deseada. Veamos un ejemplo sencillo:

```
x <- c(2.5, 4.3, 1.2, 3.1, 5.0) # valores originales
ii <- order(x)
ii    # vector de ordenación
```

```
## [1] 3 1 4 2 5
```

```
x[ii] # valores ordenados
```

```
## [1] 1.2 2.5 3.1 4.3 5.0
```

En el caso de vectores, el procedimiento anterior se podría hacer directamente con:

```
sort(x)
```

Sin embargo, para ordenar data.frames será necesario la utilización del vector de índices de ordenación. A continuación, los datos de `cars` ordenados por `dist`:

```
ii <- order(cars$dist) # Vector de índices de ordenación
cars2 <- cars[ii, ]    # Datos ordenados por dist
head(cars2)
```

```
##    speed dist velocidad distancia
## 1      4    2  6.437388 0.6096074
## 3      7    4 11.265430 1.2192148
## 2      4   10  6.437388 3.0480371
## 6      9   10 14.484124 3.0480371
## 12    12   14 19.312165 4.2672519
## 5      8   16 12.874777 4.8768593
```

4.2.2.2 Filtrado

El filtrado de datos consiste en elegir una submuestra que cumpla determinadas condiciones. Para ello se puede utilizar la función `subset()` (que además permite seleccionar variables).

A continuación se muestran un par de ejemplos:

```
subset(cars, dist > 85) # datos con dis>85
```

```
##    speed dist velocidad distancia
## 47    24   92  38.62433  28.04194
```

```
## 48    24   93  38.62433  28.34674
## 49    24  120  38.62433  36.57644
```

```
subset(cars, speed > 10 & speed < 15 & dist > 45) # speed en (10,15) y dist>45
```

```
##    speed dist velocidad distancia
## 19    13   46  20.92151  14.02097
## 22    14   60  22.53086  18.28822
## 23    14   80  22.53086  24.38430
```

También se pueden hacer el filtrado empleando directamente los correspondientes vectores de índices:

```
ii <- cars$dist > 85
cars[ii, ]   # dis>85
```

```
##    speed dist velocidad distancia
## 47    24   92  38.62433  28.04194
## 48    24   93  38.62433  28.34674
## 49    24  120  38.62433  36.57644
```

```
ii <- cars$speed > 10 & cars$speed < 15 & cars$dist > 45
cars[ii, ]  # speed en (10,15) y dist>45
```

```
##    speed dist velocidad distancia
## 19    13   46  20.92151  14.02097
## 22    14   60  22.53086  18.28822
## 23    14   80  22.53086  24.38430
```

En este caso puede ser de utilidad la función `which()`:

```
it <- which(ii)
str(it)
```

```
##  int [1:3] 19 22 23
```

```
cars[it, 1:2]
```

```
##    speed dist
## 19    13   46
## 22    14   60
## 23    14   80
```

```
# rownames(cars[it, 1:2])

id <- which(!ii)
str(cars[id, 1:2])
```

```
## 'data.frame':    47 obs. of  2 variables:
##  $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
##  $ dist : num  2 10 4 22 16 10 18 26 34 17 ...
```

```
# Se podría p.e. emplear cars[id, ] para predecir cars[it, ]$speed
# ?which.min
```

Capítulo 5

Análisis exploratorio de datos

El objetivo del *análisis exploratorio de datos* es presentar una descripción de los mismos que faciliten su análisis mediante procedimientos que permitan:

- Organizar los datos
- Resumirlos
- Representarlos gráficamente
- Análizar la información

5.1 Medidas resumen

5.1.1 Datos de ejemplo

El fichero *empleados.RData* contiene datos de empleados de un banco que utilizaremos, entre otros, a modo de ejemplo.

```
load("datos/empleados.RData")
data.frame(Etiquetas = attr(empleados, "variable.labels"))  # Listamos las etiquetas
```

```
##                              Etiquetas
## id                    Código de empleado
## sexo                                Sexo
## fechnac             Fecha de nacimiento
## educ              Nivel educativo (años)
## catlab                 Categoría Laboral
## salario                   Salario actual
## salini                   Salario inicial
## tiempemp       Meses desde el contrato
## expprev     Experiencia previa (meses)
## minoria            Clasificación étnica
## sexoraza Clasificación por sexo y raza
```

Para hacer referencia directamente a las variables de *empleados*

```
attach(empleados)
```

5.1.2 Tablas de frecuencias

```
table(sexo)
```

```
## sexo
## Hombre  Mujer
##    258    216
```

```
prop.table(table(sexo))
```

```
## sexo
##    Hombre     Mujer
## 0.5443038 0.4556962
```

```
table(sexo,catlab)
```

```
##         catlab
## sexo     Administrativo Seguridad Directivo
##   Hombre            157        27        74
##   Mujer             206         0        10
```

```
prop.table(table(sexo,catlab))
```

```
##         catlab
## sexo     Administrativo  Seguridad  Directivo
##   Hombre     0.33122363 0.05696203 0.15611814
##   Mujer      0.43459916 0.00000000 0.02109705
```

```
prop.table(table(sexo,catlab), 1)
```

```
##         catlab
## sexo     Administrativo Seguridad Directivo
##   Hombre      0.6085271 0.1046512 0.2868217
##   Mujer       0.9537037 0.0000000 0.0462963
```

```
prop.table(table(sexo,catlab), 2)
```

```
##         catlab
## sexo     Administrativo Seguridad Directivo
##   Hombre      0.4325069 1.0000000 0.8809524
##   Mujer       0.5674931 0.0000000 0.1190476
```

```
table(catlab,educ,sexo)
```

```
## , , sexo = Hombre
##
##                 educ
## catlab             8  12  14  15  16  17  18  19  20  21
##   Administrativo  10  48   6  78  10   2   2   1   0   0
##   Seguridad       13  13   0   1   0   0   0   0   0   0
##   Directivo        0   1   0   4  25   8   7  26   2   1
##
## , , sexo = Mujer
##
##                 educ
## catlab             8  12  14  15  16  17  18  19  20  21
##   Administrativo  30 128   0  33  14   1   0   0   0   0
##   Seguridad        0   0   0   0   0   0   0   0   0   0
##   Directivo        0   0   0   0  10   0   0   0   0   0
```

```
round(prop.table(table(catlab,educ,sexo)),2)
```

```
## , , sexo = Hombre
##
##                 educ
## catlab              8   12   14   15   16   17   18   19   20   21
##   Administrativo 0.02 0.10 0.01 0.16 0.02 0.00 0.00 0.00 0.00 0.00
##   Seguridad      0.03 0.03 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
##   Directivo      0.00 0.00 0.00 0.01 0.05 0.02 0.01 0.05 0.00 0.00
```

```
##
## , , sexo = Mujer
##
##                 educ
## catlab              8   12   14   15   16   17   18   19   20   21
##   Administrativo 0.06 0.27 0.00 0.07 0.03 0.00 0.00 0.00 0.00 0.00
##   Seguridad      0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
##   Directivo      0.00 0.00 0.00 0.00 0.02 0.00 0.00 0.00 0.00 0.00
```

Si la variable es ordinal, entonces también son de interés las frecuencias acumuladas

```
table(educ)
```

```
## educ
##   8  12  14  15  16  17  18  19  20  21
##  53 190   6 116  59  11   9  27   2   1
```

```
prop.table(table(educ))
```

```
## educ
##           8          12          14          15          16          17
## 0.111814346 0.400843882 0.012658228 0.244725738 0.124472574 0.023206751
##          18          19          20          21
## 0.018987342 0.056962025 0.004219409 0.002109705
```

```
cumsum(table(educ))
```

```
##   8  12  14  15  16  17  18  19  20  21
##  53 243 249 365 424 435 444 471 473 474
```

```
cumsum(prop.table(table(educ)))
```

```
##         8        12        14        15        16        17        18        19
## 0.1118143 0.5126582 0.5253165 0.7700422 0.8945148 0.9177215 0.9367089 0.9936709
##        20        21
## 0.9978903 1.0000000
```

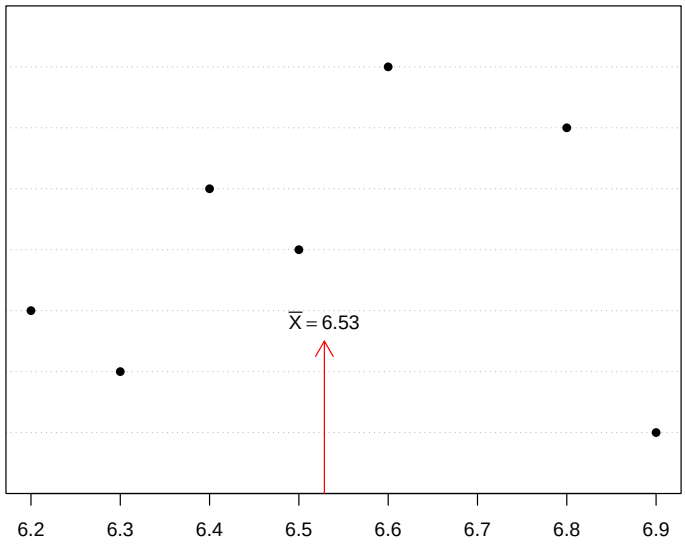
5.1.3 Media y varianza

La media es la medida de centralización por excelencia. Para su cálculo se utiliza la instrucción *mean*

```
consumo<-c(6.9, 6.3, 6.2, 6.5 ,6.4, 6.8, 6.6)
mean(consumo)
```

```
## [1] 6.528571
```

```
dotchart(consumo,pch=16)
text(mean(consumo),2.5, pos=3,expression(bar(X)==6.53))
arrows(mean(consumo),0,mean(consumo),2.5,length = 0.15,col='red')
```




```
mean(salario)
```

```
## [1] 34419.57
```

```
mean(subset(empleados,catlab=='Directivo')$salario)
```

```
## [1] 63977.8
```

También se puede utilizar la función *tapply*, que se estudiará con detalle más adelante

```
tapply(salario, catlab, mean)
```

```
## Administrativo      Seguridad      Directivo
##       27838.54       30938.89       63977.80
```

La principal medida de dispersión es la varianza. En la práctica, cuando se trabaja con datos muestrales, se sustituye por la *cuasi*-varianza (también llamada varianza muestral corregida), que se calcula mediante el comando *var*

```
var(consumo)
```

```
## [1] 0.06571429
```

```
var(salario)
```

```
## [1] 291578214
```

La *cuasi*-desviación típica se calcula

```
sd(consumo)
```

```
## [1] 0.256348
```

```
sd(salario)
```

```
## [1] 17075.66
```

o, equivalentemente,

```
sqrt(var(consumo))
```

```
## [1] 0.256348
```

```
sqrt(var(salario))
```

```
## [1] 17075.66
```

La media de dispersión adimensional (relativa) más utilizada es el *coeficiente de variación* (de Pearson)

```
sd(consumo)/abs(mean(consumo))
```

```
## [1] 0.03926555
```

que también podemos expresar en tanto por cien

```
100*sd(consumo)/abs(mean(consumo))
```

```
## [1] 3.926555
```

El coeficiente de variación nos permite, entre otras cosas, comparar dispersiones de variables medidas en diferentes unidades

```
100*sd(salini)/abs(mean(salini))
```

```
## [1] 46.2541
```

```
100*sd(salario)/abs(mean(salario))
```

```
## [1] 49.61033
```

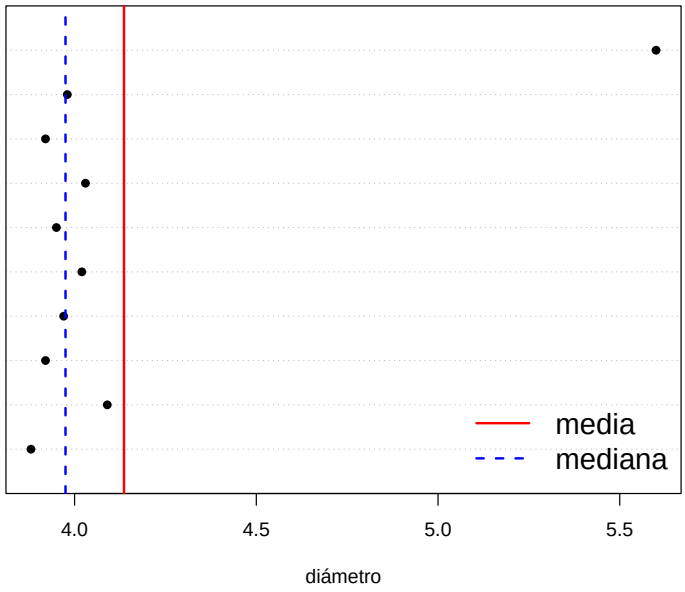
```
100*sd(expprev)/abs(mean(expprev))
```

```
## [1] 109.1022
```

5.1.4 Mediana y cuantiles

La mediana es una medida de centralización robusta. Se calcula mediante *median*

```
diametro <- c(3.88,4.09,3.92,3.97,4.02,3.95, 4.03,3.92,3.98,5.60)
dotchart(diametro,pch=16,xlab="diámetro")
abline(v=mean(diametro),col='red',lwd=2)
abline(v=median(diametro),col='blue',lty=2,lwd=2)
legend("bottomright",c("media","mediana"),
       col=c("red","blue"),lty=c(1,2),lwd=c(2,2),box.lty=0,cex=1.5)
```



Podemos comprobar que la variable *salario* presenta una asimetría derecha

```
mean(salario); median(salario)
```

```
## [1] 34419.57
```

```
## [1] 28875
```

Calculemos cuántos empleados tienen un salario inferior al salario medio

```
mean(salario < mean(salario))
```

```
## [1] 0.6940928
```

```
paste('El ', round(100*mean(salario < mean(salario)),0), '%',
      ' de los empleados tienen un salario inferior al salario medio', sep='')
```

```
## [1] "El 69% de los empleados tienen un salario inferior al salario medio"
```

Como sabemos, la mitad de los empleados tienen un salario inferior a la mediana

```
mean(salario < median(salario))
```

```
## [1] 0.5
```

Los cuantiles son una generalización de la mediana, que se corresponde con el cuantil de orden 0.5. *R* contempla distintas formas de calcular los cuantiles

```
median(c(1,2,3,4))
```

```
## [1] 2.5
```

```
quantile(c(1,2,3,4),0.5)
```

```
## 50%
## 2.5
```

```
quantile(c(1,2,3,4),0.5,type=1)
```

```
## 50%
##   2
```

Calculemos los *cuartiles* y los *deciles* de la variable *salario*

```
quantile(salario)
```

```
##       0%      25%      50%      75%     100%
##  15750.0  24000.0  28875.0  36937.5 135000.0
```

```
quantile(salario, probs=c(0.25,0.5,0.75))
```

```
##     25%     50%     75%
## 24000.0 28875.0 36937.5
```

```
quantile(salario, probs=seq(0.1, 0.9, 0.1))
```

```
##     10%     20%     30%     40%     50%     60%     70%     80%     90%
## 21045.0 22950.0 24885.0 26700.0 28875.0 30750.0 34500.0 40920.0 59392.5
```

El *rango* y el *rango intercuartílico*

```
data.frame(Rango=max(salario)-min(salario),
           RI=as.numeric(quantile(salario, 0.75) - quantile(salario, 0.25)))
```

```
##    Rango      RI
## 1 119250 12937.5
```

5.1.5 *Summary*

```
summary(empleados)
```

```
##        id            sexo        fechnac                educ
##  Min.   :  1.0   Hombre:258   Min.   :1929-02-10   Min.   : 8.00
##  1st Qu.:119.2   Mujer :216   1st Qu.:1948-01-03   1st Qu.:12.00
##  Median :237.5                Median :1962-01-23   Median :12.00
##  Mean   :237.5                Mean   :1956-10-08   Mean   :13.49
##  3rd Qu.:355.8                3rd Qu.:1965-07-06   3rd Qu.:15.00
##  Max.   :474.0                Max.   :1971-02-10   Max.   :21.00
##                               NA's   :1
##             catlab       salario           salini         tiempemp
##  Administrativo:363   Min.   : 15750   Min.   : 9000   Min.   :63.00
##  Seguridad     : 27   1st Qu.: 24000   1st Qu.:12488   1st Qu.:72.00
##  Directivo     : 84   Median : 28875   Median :15000   Median :81.00
##                       Mean   : 34420   Mean   :17016   Mean   :81.11
##                       3rd Qu.: 36938   3rd Qu.:17490   3rd Qu.:90.00
##                       Max.   :135000   Max.   :79980   Max.   :98.00
##
##     expprev       minoria           sexoraza
##  Min.   :  0.00   No:370   Blanca varón :194
##  1st Qu.: 19.25   Sí:104   Minoría varón: 64
##  Median : 55.00            Blanca mujer :176
##  Mean   : 95.86            Minoría mujer: 40
##  3rd Qu.:138.75
##  Max.   :476.00
##
```

```
summary(subset(empleados,catlab=='Directivo'))
```

```
##        id            sexo       fechnac                educ
##  Min.   :  1.0   Hombre:74   Min.   :1937-07-12   Min.   :12.00
##  1st Qu.:102.5   Mujer :10   1st Qu.:1954-08-09   1st Qu.:16.00
##  Median :233.5               Median :1961-05-29   Median :17.00
##  Mean   :234.1               Mean   :1958-11-26   Mean   :17.25
```

```
##  3rd Qu.:344.2                3rd Qu.:1963-10-03   3rd Qu.:19.00
##  Max.   :468.0                Max.   :1966-04-05   Max.   :21.00
##             catlab      salario           salini         tiempemp
##  Administrativo: 0   Min.   : 34410   Min.   :15750   Min.   :64.00
##  Seguridad     : 0   1st Qu.: 51956   1st Qu.:23063   1st Qu.:73.00
##  Directivo     :84   Median : 60500   Median :28740   Median :81.00
##                      Mean   : 63978   Mean   :30258   Mean   :81.15
##                      3rd Qu.: 71281   3rd Qu.:34058   3rd Qu.:91.00
##                      Max.   :135000   Max.   :79980   Max.   :98.00
##     expprev       minoria          sexoraza
##  Min.   :  3.00   No:80   Blanca varón :70
##  1st Qu.: 19.75   Sí: 4   Minoría varón: 4
##  Median : 52.00           Blanca mujer :10
##  Mean   : 77.62           Minoría mujer: 0
##  3rd Qu.:125.25
##  Max.   :285.00
```

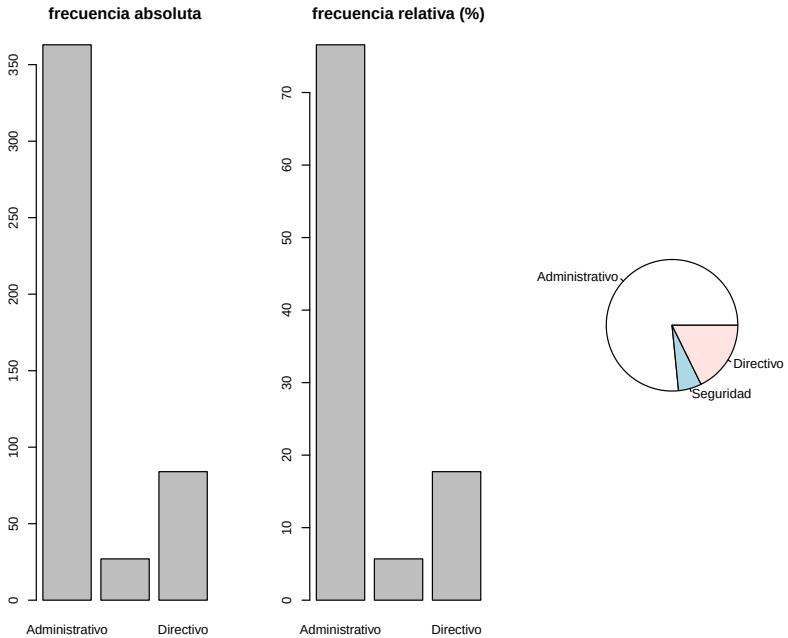
5.2 Gráficos

5.2.1 Diagrama de barras y gráfico de sectores

```
table(catlab)
```

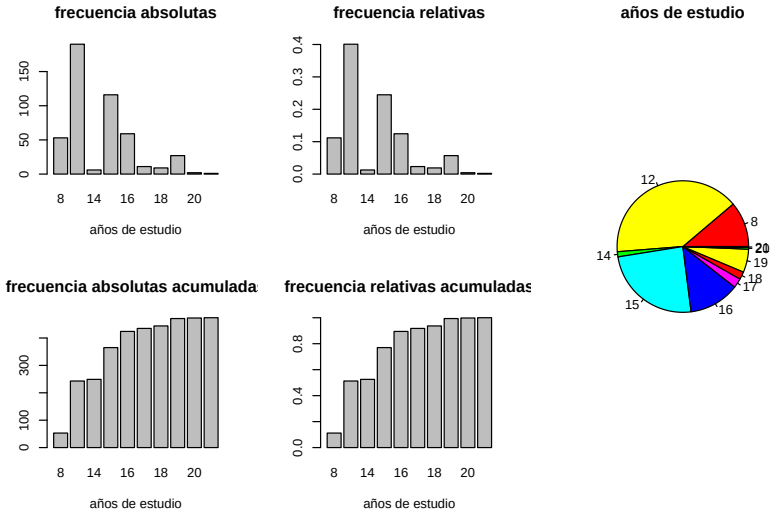
```
## catlab
## Administrativo      Seguridad      Directivo
##            363             27             84
```

```
par(mfrow = c(1, 3))
barplot(table(catlab),main="frecuencia absoluta")
barplot(100*prop.table(table(catlab)),main="frecuencia relativa (%)")
pie(table(catlab))
```




```
nj <- table(educ)
fj <- prop.table(nj)
```

```
Nj <- cumsum(nj)
Fj <- cumsum(fj)
layout(matrix(c(1,2,5,3,4,5), 2, 3, byrow=TRUE), respect=TRUE)
barplot(nj,main="frecuencia absolutas",xlab='años de estudio')
barplot(fj,main="frecuencia relativas",xlab='años de estudio')
barplot(Nj,main="frecuencia absolutas acumuladas",xlab='años de estudio')
barplot(Fj,main="frecuencia relativas acumuladas",xlab='años de estudio')
pie(nj,col=rainbow(6),main='años de estudio')
```



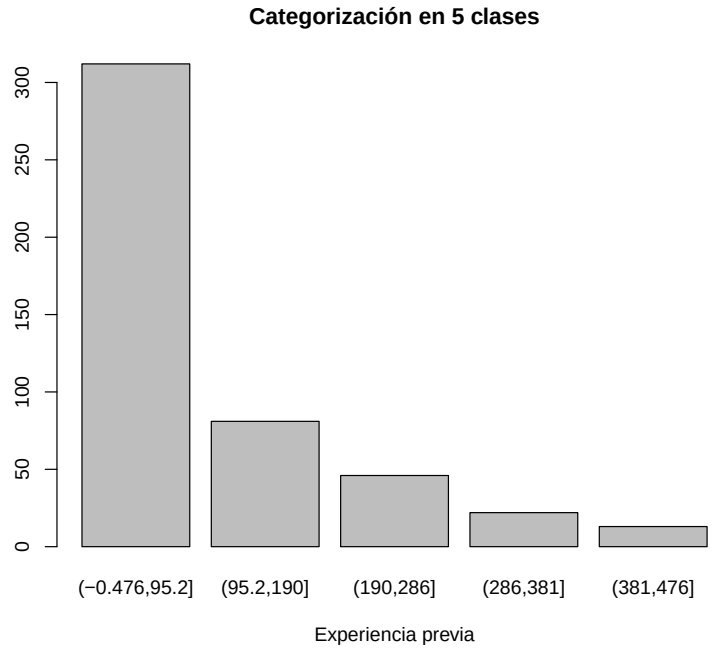

```
par(mfrow = c(1, 1))
```

Con datos continuos, podemos hacer uso de la función *cut* (más adelante veremos como se representa el histograma)

```
table(cut(expprev, breaks=5))
```

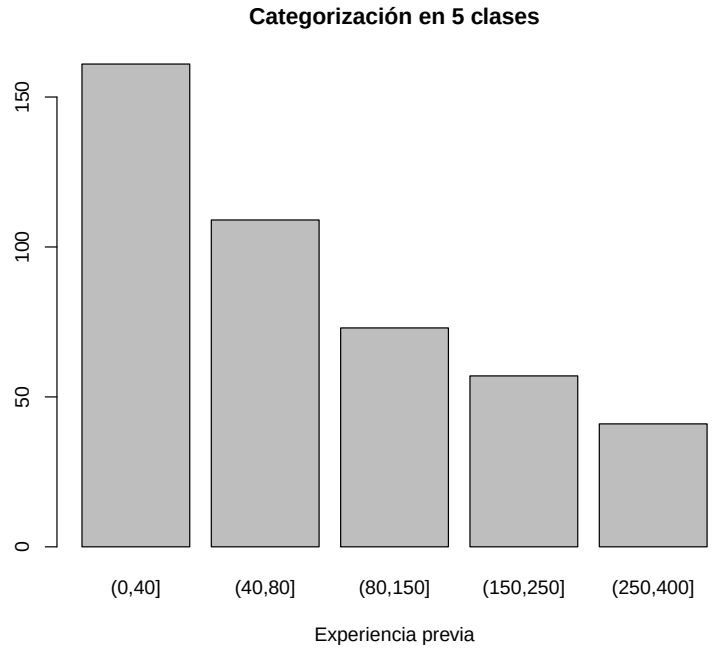
```
##
## (-0.476,95.2]    (95.2,190]     (190,286]     (286,381]     (381,476]
##           312            81            46            22            13
```

```
barplot(table(cut(expprev,breaks=5)),xlab="Experiencia previa",
        main="Categorización en 5 clases")
```



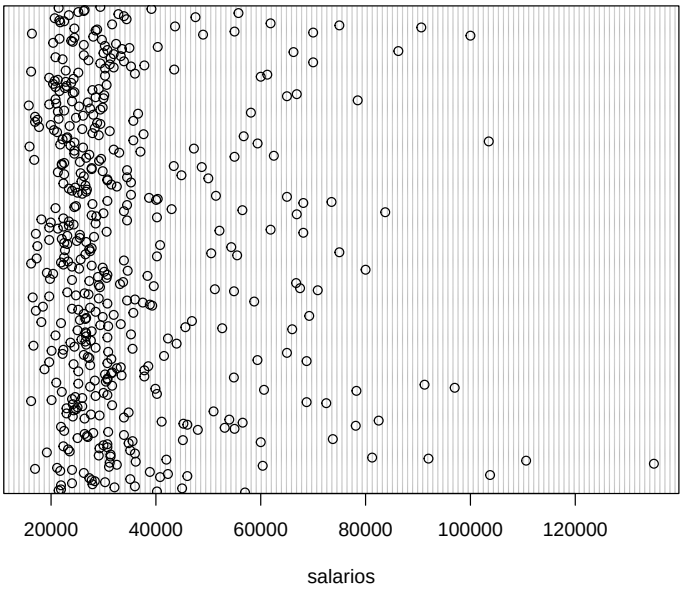
Debemos ser muy cuidadosos a la hora de valorar gráficas como la siguiente

```
tt <- table(cut(expprev, breaks=c(0,40,80,150,250,400)))
barplot(tt,xlab="Experiencia previa", main="Categorización en 5 clases")
```

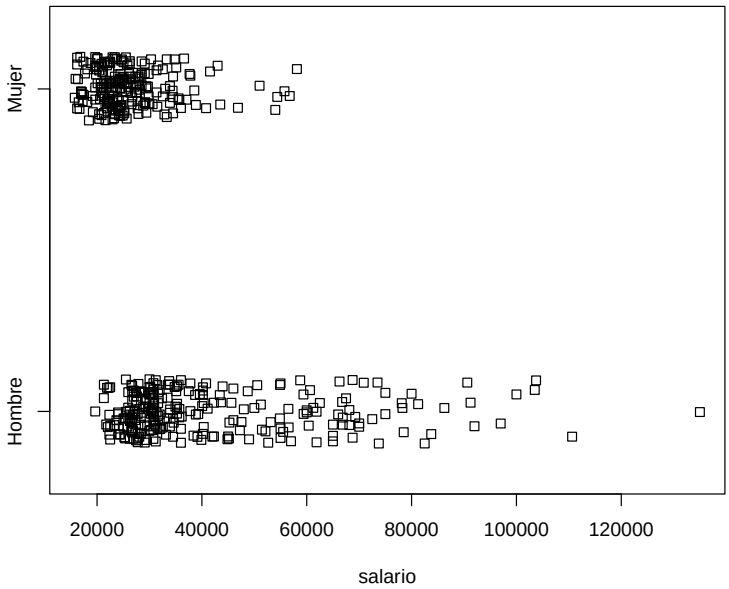


5.2.2 Gráfico de puntos

```
dotchart(salario, xlab='salarios')
```




```
stripchart(salario~sexo, method='jitter')
```



5.2.3 Árbol de tallo y hojas

Esta representación puede ser útil cuando se dispone de pocos datos.

```
stem(salario)
```

```
##
##   The decimal point is 4 digit(s) to the right of the |
##
##    1 | 666666777777777778888999
```

```
##    2 | 00000000000001111111111111111111222222222222222222222233333333333+148
##    3 | 00000000000000000011111111111111111111111112222222222222233333333333+36
##    4 | 000000000111222233444555666778899
##    5 | 01111233445555566777778999
##    6 | 0001122355566777888999
##    7 | 00134455889
##    8 | 01346
##    9 | 1127
##   10 | 044
##   11 | 1
##   12 |
##   13 | 5
```

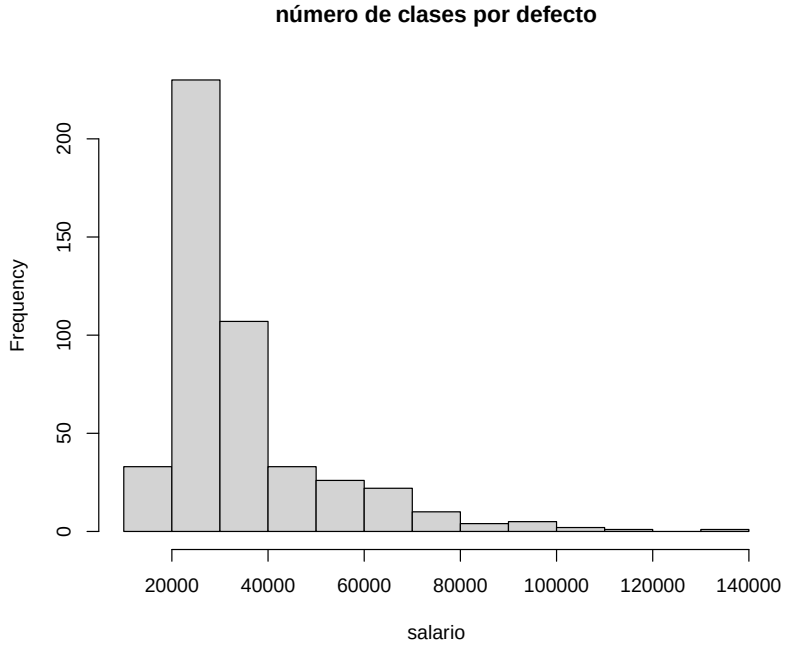
```
stem(tiempemp)
```

```
##
##   The decimal point is at the |
##
##   62 | 000
##   64 | 00000000000000000000
##   66 | 000000000000000000000000000000000
##   68 | 00000000000000000000000000000000
##   70 | 0000000000000000
##   72 | 000000000000000000000000
##   74 | 000000000000000
##   76 | 00000000000000000000
##   78 | 0000000000000000000000000000000000000
##   80 | 00000000000000000000000000000000000000
##   82 | 0000000000000000000000000000000000
##   84 | 00000000000000000000
##   86 | 00000000000000000000
##   88 | 0000000000000000000
##   90 | 00000000000000000000000000000
##   92 | 000000000000000000000000000000000000000
##   94 | 00000000000000000000
##   96 | 00000000000000000000000000
##   98 | 00000000000000
```

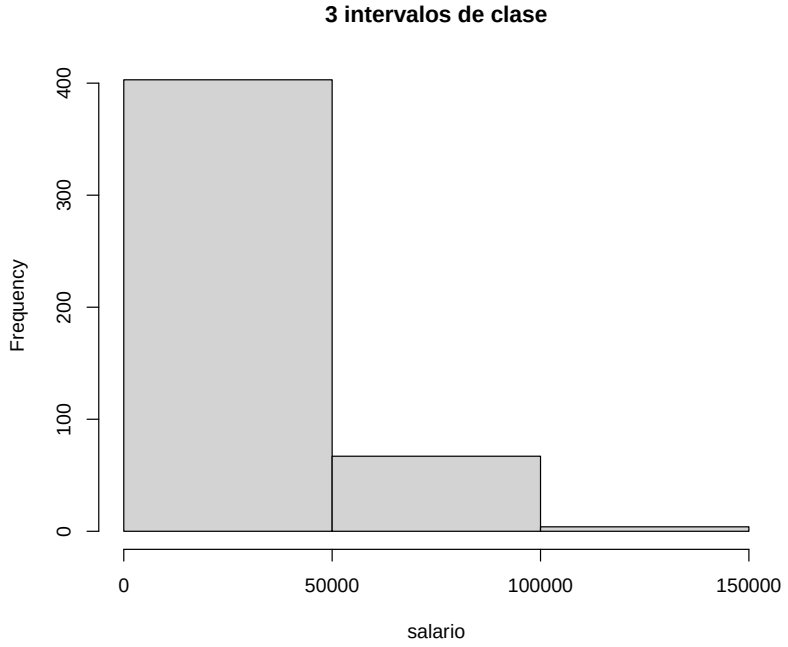
5.2.4 Histograma

Este gráfico es uno de los más habituales para representar datos continuos

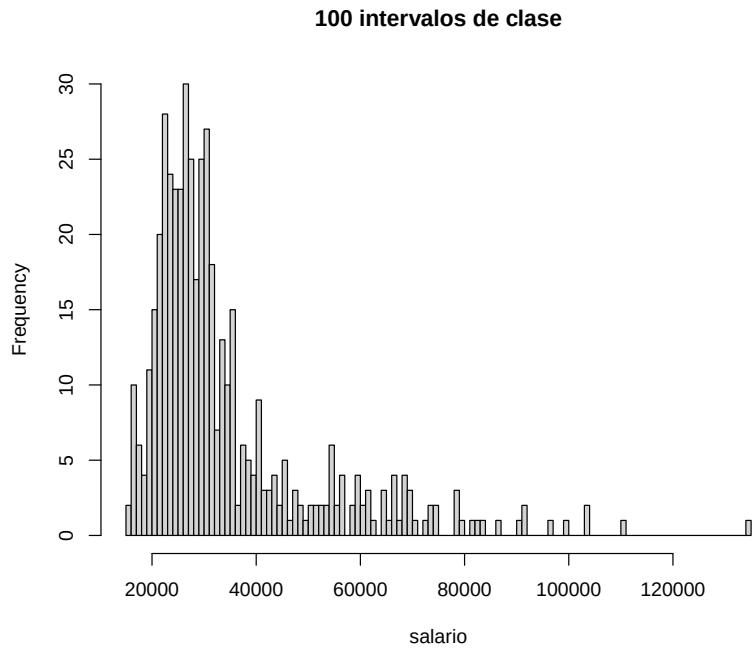
```
hist(salario, main='número de clases por defecto')
```



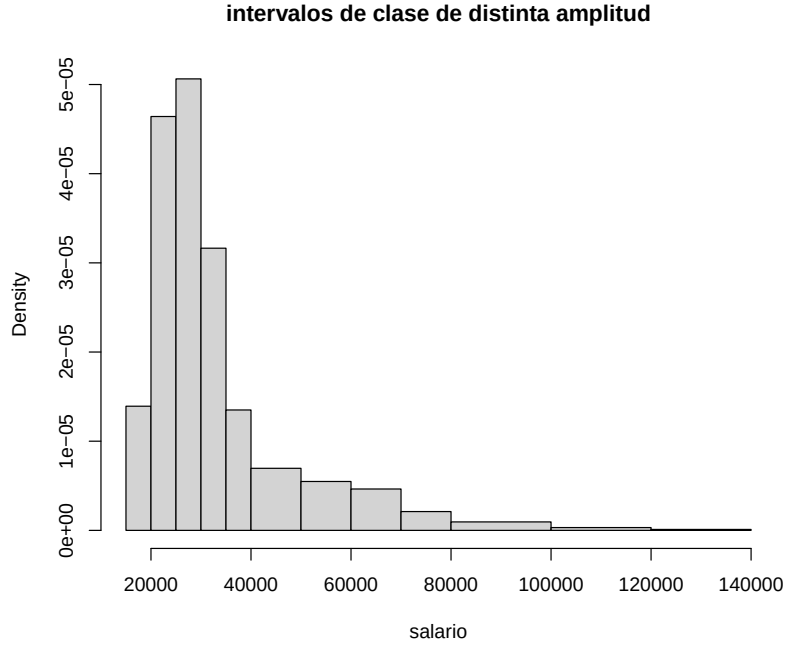

```
hist(salario, breaks=3, main='3 intervalos de clase')
```




```
hist(salario, breaks=100, main='100 intervalos de clase')
```



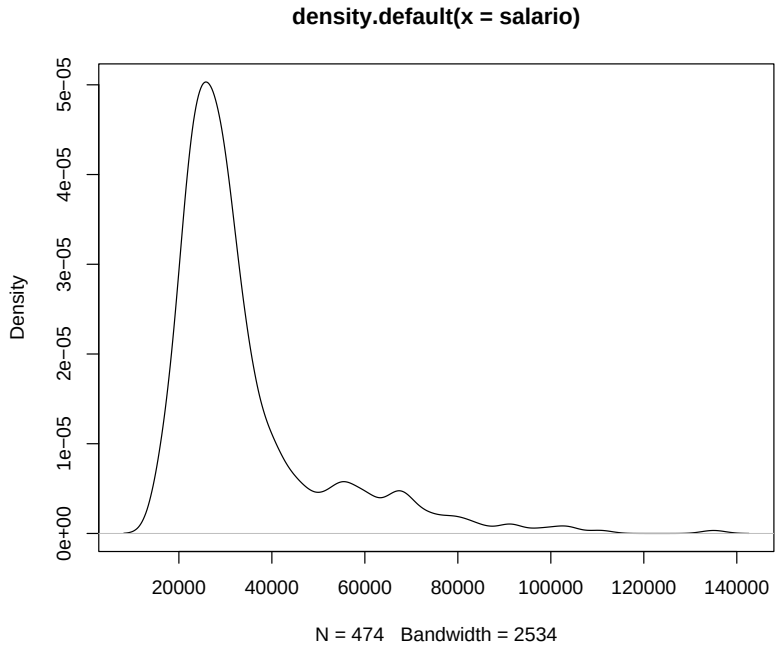

```
cl1 <- seq(15000,40000,5000)
cl2 <- seq(50000,80000,10000)
cl3 <- seq(100000,140000,20000)
hist(salario, breaks=c(cl1,cl2,cl3),main='intervalos de clase de distinta amplitud')
```



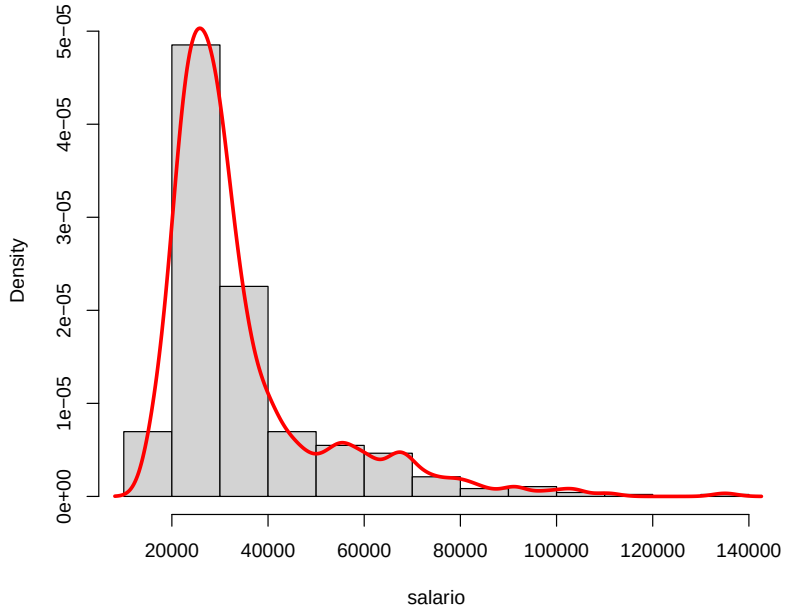
5.2.5 Gráfico de densidad

Es una versión suavizada del histograma.

```
plot(density(salario))
```

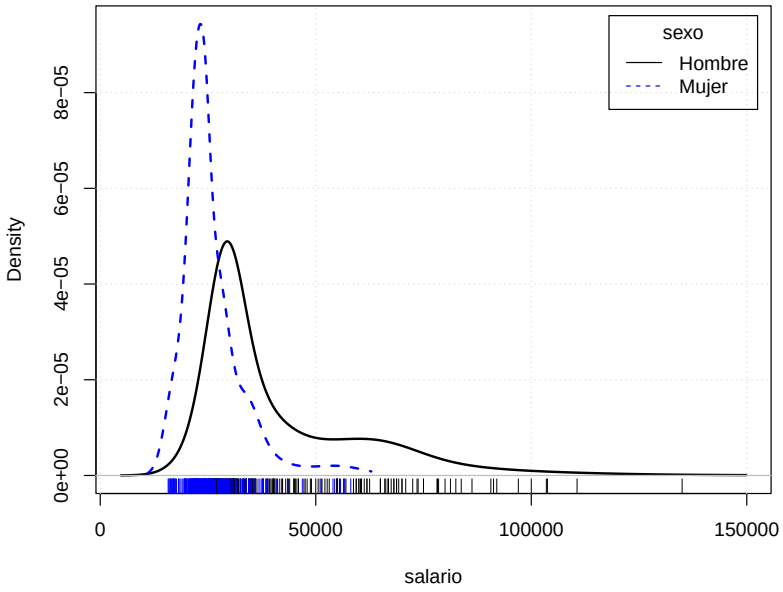



```
hist(salario, freq=F, main='')
lines(density(salario), lwd=3, col='red')
```



El paquete *car* nos da acceso a la instrucción *densityPlot*:

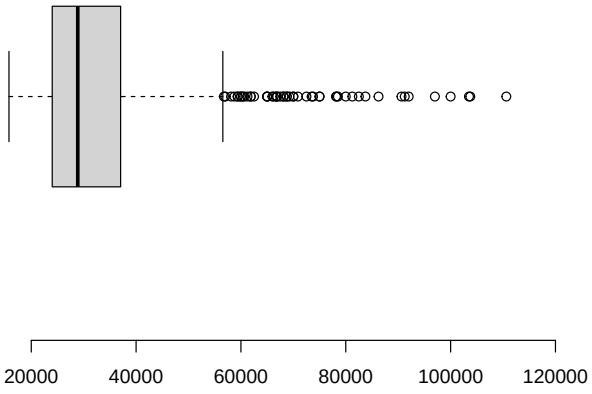
```
library(car)  # help(car)
densityPlot(salario~sexo)
```



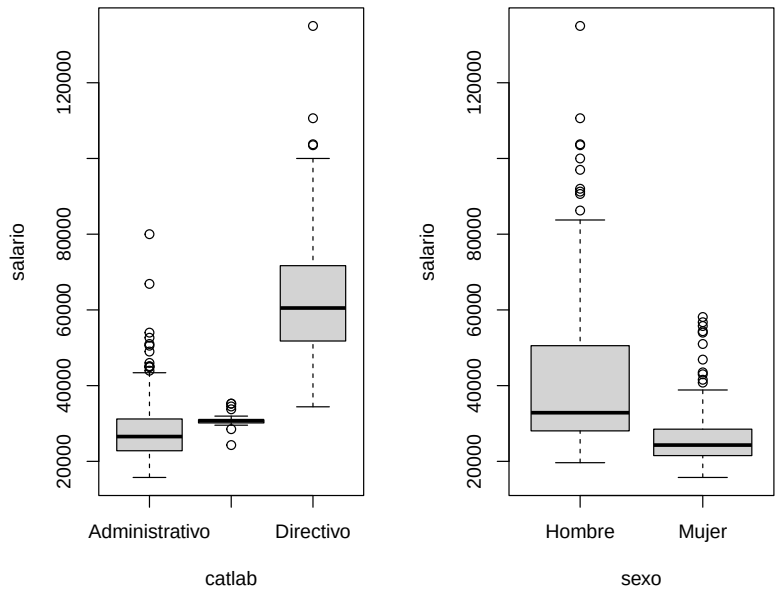
5.2.6 Diagrama de cajas

Se trata de un gráfico muy polivalente

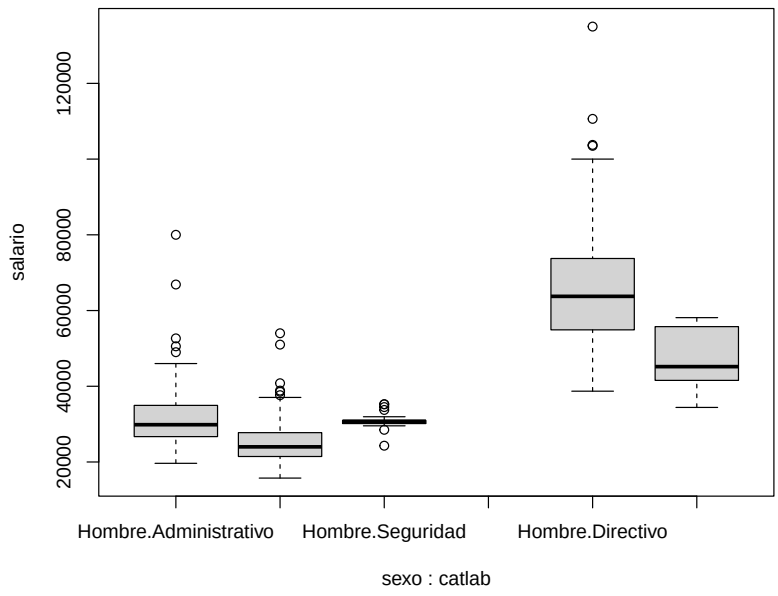
```
boxplot(salario, horizontal=T, axes=F)
axis(1)
```



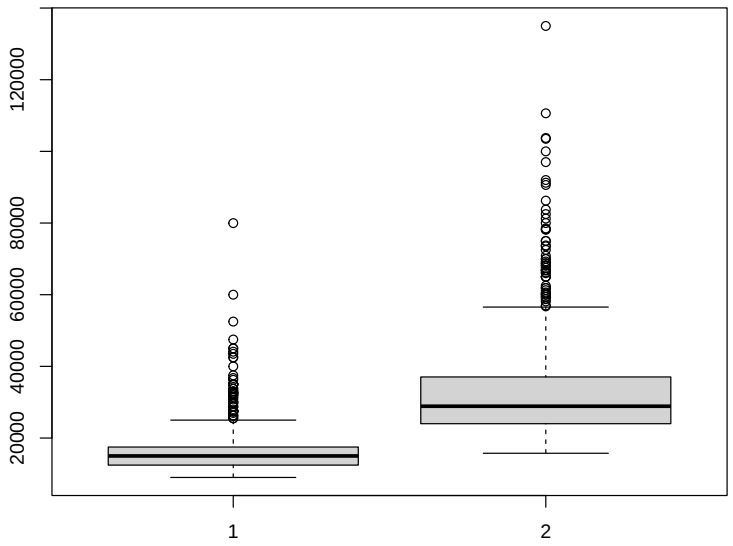

```
par(mfrow=c(1,2))
boxplot(salario~catlab)
boxplot(salario~sexo)
```



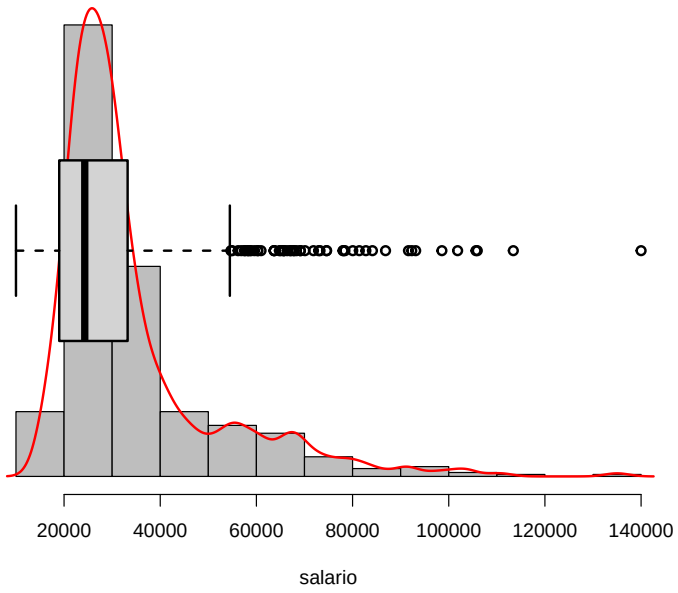

```
par(mfrow=c(1,1))
boxplot(salario~sexo*catlab)
```




```
boxplot(salini, salario)
```



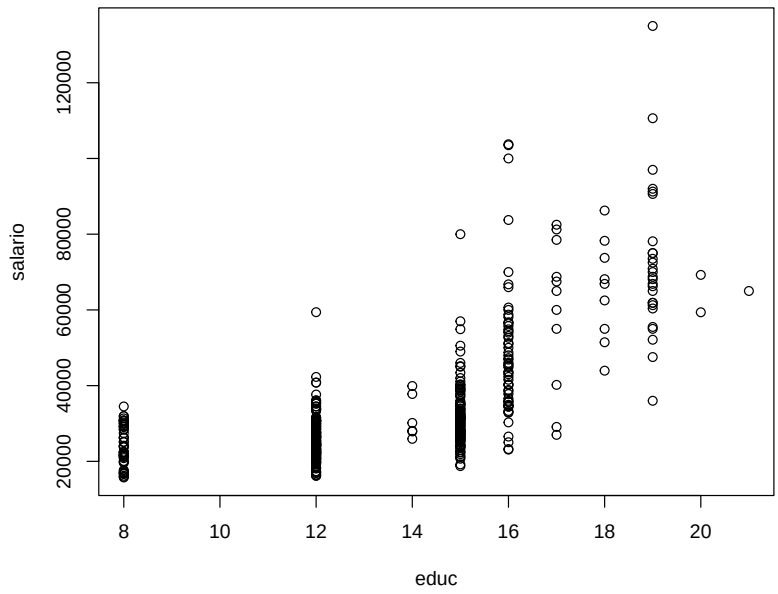

```
hist(salario,probability=T,ylab="",col='grey',axes=F,main=""); axis(1)
lines(density(salario),col='red',lwd=2)
par(new=T)
boxplot(salario,horizontal=T,axes=F,lwd=2)
```



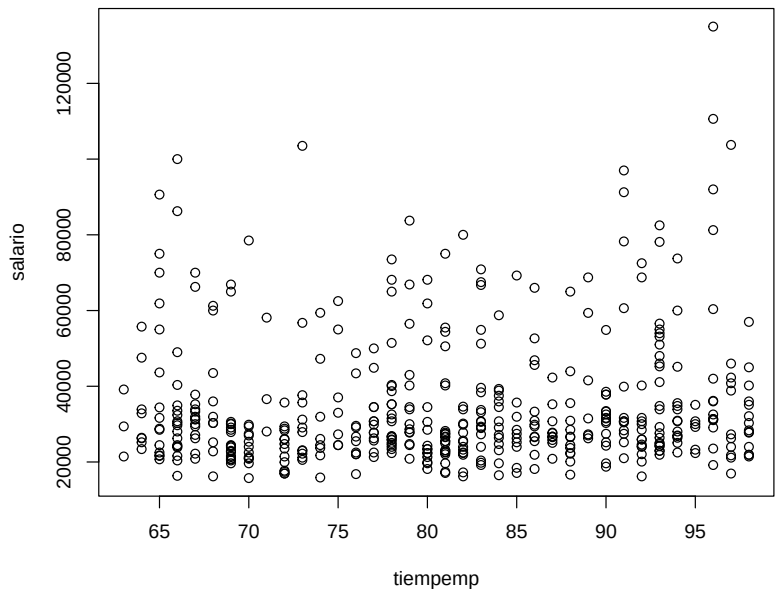
5.2.7 Gráfica de dispersión

Permite ver la relación entre dos variables:

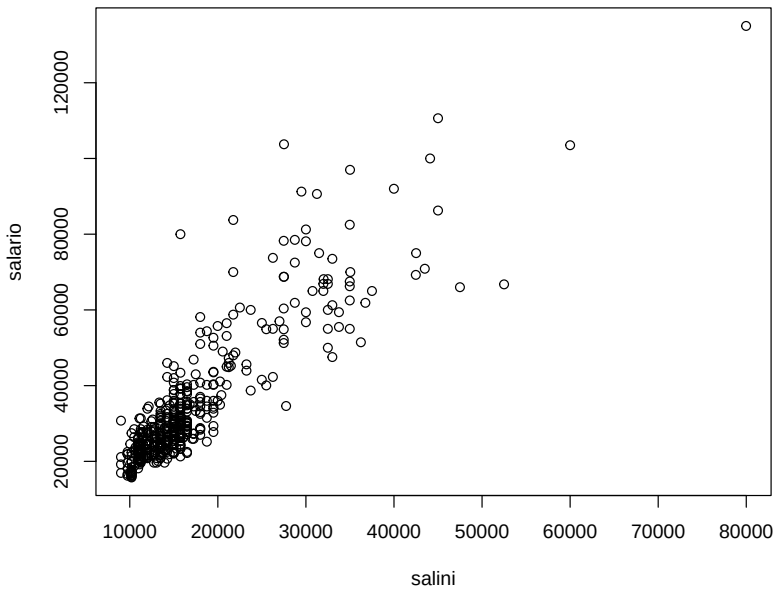
```
plot(educ,salario)
```




```
plot(tiempemp,salario)
```




```
plot(salini,salario)
```

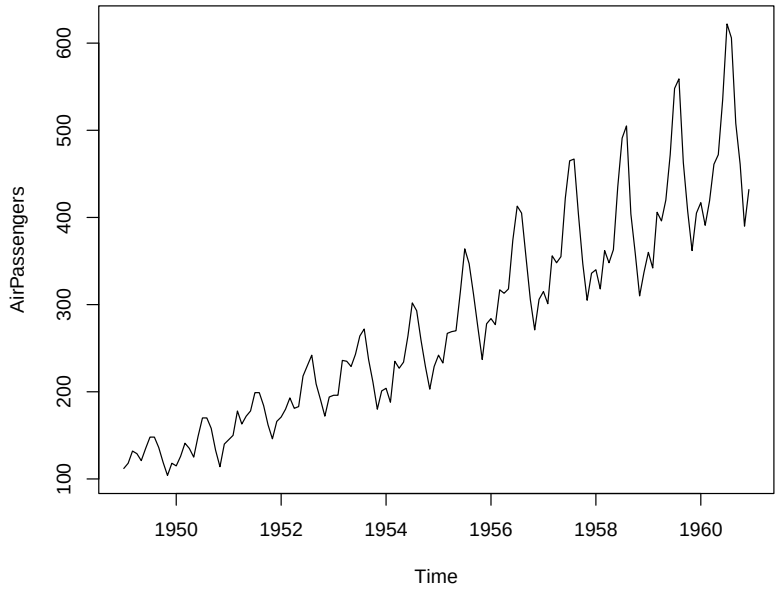


En el caso de una serie temporal

```
AirPassengers
```

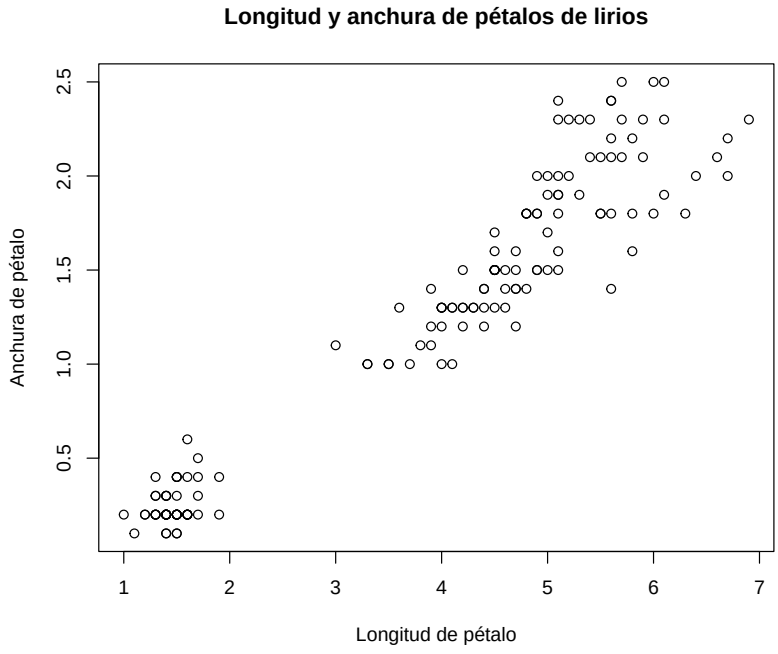
```
##      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 1949 112 118 132 129 121 135 148 148 136 119 104 118
## 1950 115 126 141 135 125 149 170 170 158 133 114 140
## 1951 145 150 178 163 172 178 199 199 184 162 146 166
## 1952 171 180 193 181 183 218 230 242 209 191 172 194
## 1953 196 196 236 235 229 243 264 272 237 211 180 201
## 1954 204 188 235 227 234 264 302 293 259 229 203 229
## 1955 242 233 267 269 270 315 364 347 312 274 237 278
## 1956 284 277 317 313 318 374 413 405 355 306 271 306
## 1957 315 301 356 348 355 422 465 467 404 347 305 336
## 1958 340 318 362 348 363 435 491 505 404 359 310 337
## 1959 360 342 406 396 420 472 548 559 463 407 362 405
## 1960 417 391 419 461 472 535 622 606 508 461 390 432
```

```
plot(AirPassengers)
```

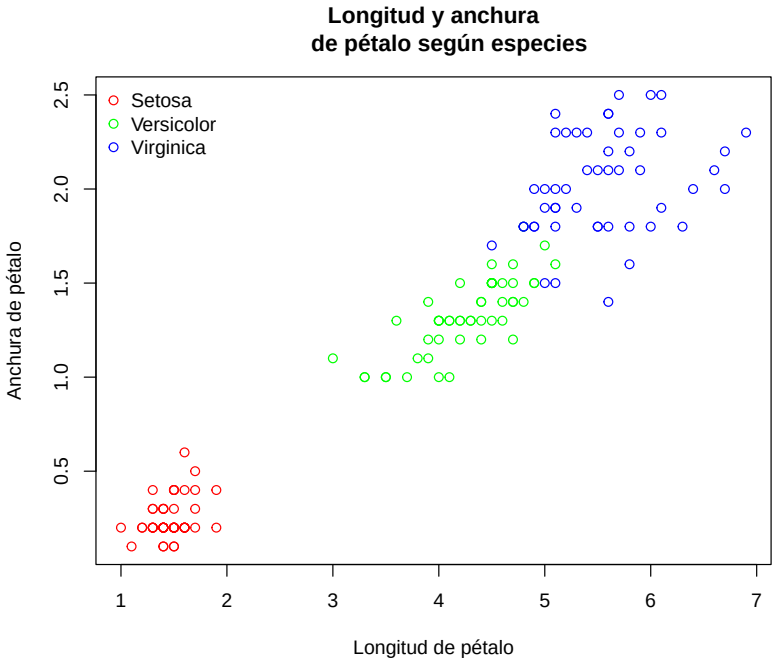


Y un último ejemplo utilizando los datos *iris* de Fisher:

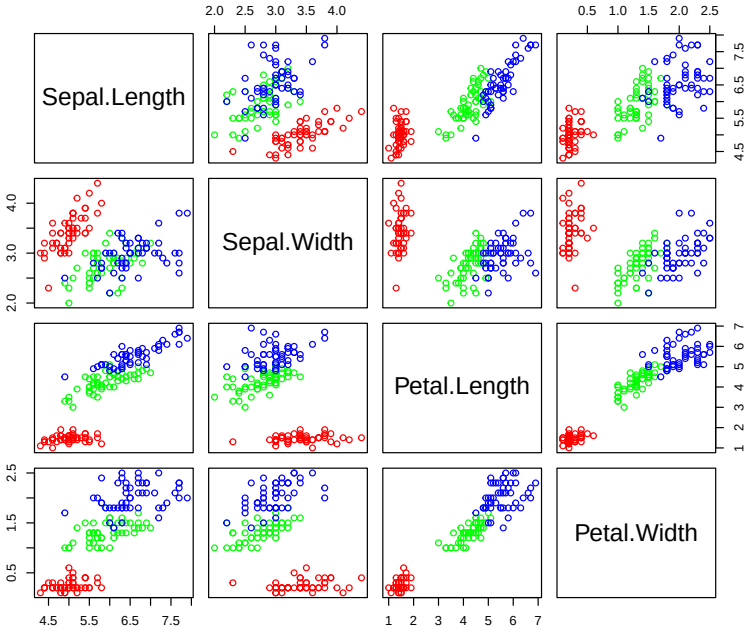
```
plot(iris[,3],iris[,4],main="Longitud y anchura de pétalos de lirios",
     xlab="Longitud de pétalo",ylab="Anchura de pétalo")
```




```
iris.color<-c("red","green","blue")[iris$Species]
plot(iris[,3],iris[,4],col=iris.color,main="Longitud y anchura
     de pétalo según especies",xlab="Longitud de pétalo",
     ylab="Anchura de pétalo")
legend("topleft",c("Setosa","Versicolor","Virginica"),pch=1,
       col=c("red","green","blue"),box.lty=0)
```




```
pairs(iris[,1:4],col=iris.color)
```



Capítulo 6

Inferencia estadística

El objetivo de este capítulo es ofrecer un primer acercamiento a la inferencia estadística, cubriendo de forma somera los siguientes apartados:

- contrastes de normalidad
- contrastes paramétricos y no paramétricos, con una y dos muestras
- regresión y correlación
- análisis de la varianza con un factor

En este capítulo utilizaremos como ejemplo los datos de clientes de una compañía de distribución industrial (HATCO) contenidos en el fichero *hatco.RData*.

```
load('datos/hatco.RData')
```

Listado de etiquetas

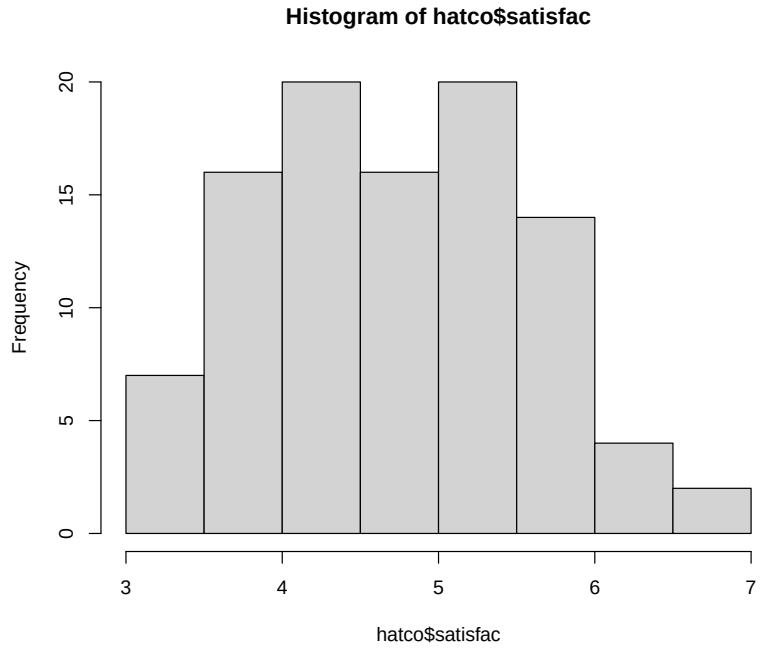
```
as.data.frame(attr(hatco, "variable.labels"))
```

```
##          attr(hatco, "variable.labels")
## empresa                         Empresa
## tamano              Tamaño de la empresa
## adquisic      Estructura de adquisición
## tindustr               Tipo de industria
## tsitcomp    Tipo de situación de compra
## velocida            Velocidad de entrega
## precio                 Nivel de precios
## flexprec        Flexibilidad de precios
## imgfabri          Imagen del fabricante
## servconj              Servicio conjunto
## imgfvent     Imagen de fuerza de ventas
## calidadp            Calidad de producto
## fidelida   Porcentaje de compra a HATCO
## satisfac            Satisfacción global
## nfidelid        Nivel de compra a HATCO
## nsatisfa          Nivel de satisfacción
```

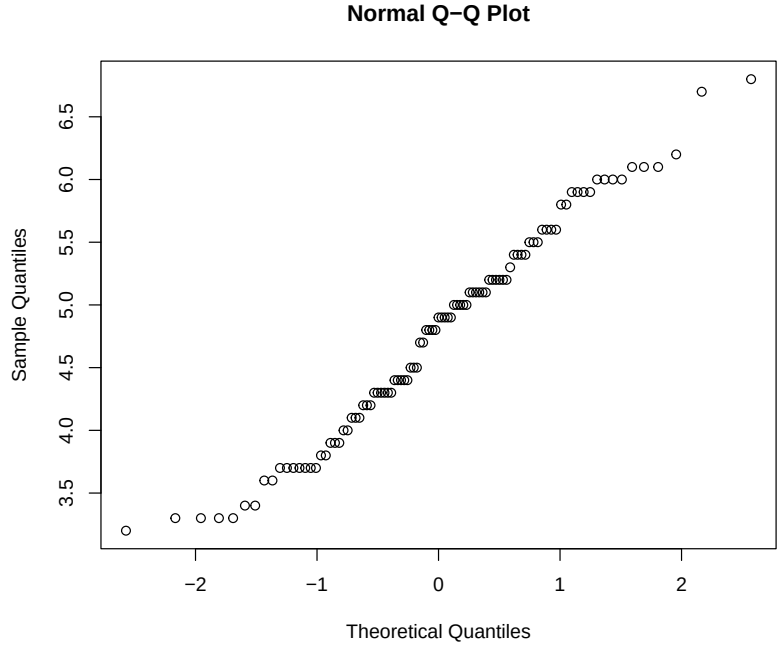
6.1 Normalidad

Queremos hacer un estudio inferencial de la variable *satisfac* (satisfacción global). Lo primero que vamos a hacer es comprobar si, visualmente, los datos parecen razonablemente simétricos y si se pueden ajustar por una distribución normal

```
hist(hatco$satisfac)
```




```
qqnorm(hatco$satisfac)
```




```
shapiro.test(hatco$satisfac)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  hatco$satisfac
## W = 0.97608, p-value = 0.06813
```

6.2 Contrastes

6.2.1 Una muestra

Obtenemos un intervalo de confianza de *satisfac*

```
t.test(hatco$satisfac)  # with(hatco, t.test(satisfac))
```

```
##
##  One Sample t-test
##
## data:  hatco$satisfac
## t = 55.301, df = 98, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  4.603406 4.946089
## sample estimates:
## mean of x
##  4.774747
```

Contrastamos si es razonable suponer que la media es 5

```
t.test(hatco$satisfac, mu=5)
```

```
##
##  One Sample t-test
##
## data:  hatco$satisfac
## t = -2.6089, df = 98, p-value = 0.01051
## alternative hypothesis: true mean is not equal to 5
## 95 percent confidence interval:
##  4.603406 4.946089
## sample estimates:
## mean of x
##  4.774747
```

Utilizando una confianza del 99%

```
t.test(hatco$satisfac, mu=5, conf.level=0.99)
```

```
##
##  One Sample t-test
##
## data:  hatco$satisfac
## t = -2.6089, df = 98, p-value = 0.01051
## alternative hypothesis: true mean is not equal to 5
## 99 percent confidence interval:
##  4.547935 5.001560
## sample estimates:
## mean of x
##  4.774747
```

Veamos si podemos afirmar que la media es menor que 5

```
t.test(hatco$satisfac, mu=5, alternative = 'less')
```

```
##
##  One Sample t-test
##
## data:  hatco$satisfac
## t = -2.6089, df = 98, p-value = 0.005253
## alternative hypothesis: true mean is less than 5
```

```
## 95 percent confidence interval:
##      -Inf 4.918122
## sample estimates:
## mean of x
##  4.774747
```

¿Y mayor que 4.65?

```
t.test(hatco$satisfac, mu=4.65, alternative = 'greater')
```

```
##
##  One Sample t-test
##
## data:  hatco$satisfac
## t = 1.4448, df = 98, p-value = 0.07585
## alternative hypothesis: true mean is greater than 4.65
## 95 percent confidence interval:
##  4.631373      Inf
## sample estimates:
## mean of x
##  4.774747
```

El test de los rangos con signo de Wilcoxon es un contraste no paramétrico (exige que la distribución sea simétrica) que se puede utilizar como alternativa al contraste t de Student

```
with(hatco, wilcox.test(satisfac, mu=5))
```

```
##
##  Wilcoxon signed rank test with continuity correction
##
## data:  satisfac
## V = 1574, p-value = 0.01303
## alternative hypothesis: true location is not equal to 5
```

6.2.2 Dos muestras

Disponemos de dos muestras independientes, el porcentaje de compra en las empresas con nivel de satisfacción bajo y alto, y asumimos que las varianzas son iguales

```
t.test(fidelida ~ nsatisfa, data = hatco, var.equal=TRUE)
```

```
##
##  Two Sample t-test
##
## data:  fidelida by nsatisfa
## t = -6.5833, df = 97, p-value = 2.363e-09
## alternative hypothesis: true difference in means between group bajo and group alto is not equal to
## 95 percent confidence interval:
##  -12.915013  -6.931653
## sample estimates:
## mean in group bajo mean in group alto
##           41.72778           51.65111
```

Si no se asume igualdad de varianzas, se calcula la variante Welch del test t

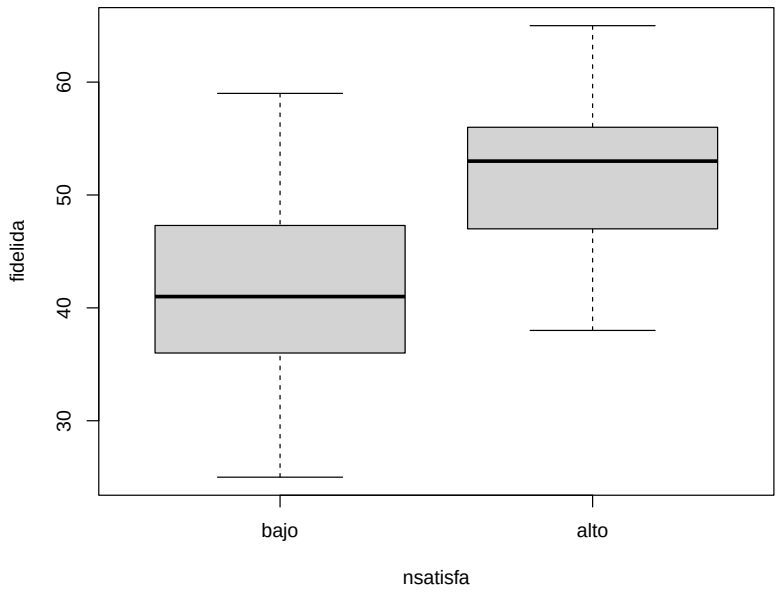
```
t.test(fidelida ~ nsatisfa, data = hatco)
```

```
##
##  Welch Two Sample t-test
##
## data:  fidelida by nsatisfa
```

```
## t = -6.6901, df = 96.995, p-value = 1.437e-09
## alternative hypothesis: true difference in means between group bajo and group alto is not equal
## 95 percent confidence interval:
##  -12.86727  -6.97940
## sample estimates:
## mean in group bajo mean in group alto
##           41.72778           51.65111
```

Comparemos visualmente las varianzas

```
boxplot(fidelida ~ nsatisfa, data = hatco)
```



La comparación de las varianzas puede hacerse con el test F

```
var.test(fidelida ~ nsatisfa, data = hatco)
```

```
##
##  F test to compare two variances
##
## data:  fidelida by nsatisfa
## F = 1.4248, num df = 53, denom df = 44, p-value = 0.2292
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  0.797925 2.505462
## sample estimates:
## ratio of variances
##           1.424804
```

Una alternativa no paramétrica

```
bartlett.test(fidelida ~ nsatisfa, data = hatco)
```

```
##
##  Bartlett test of homogeneity of variances
##
## data:  fidelida by nsatisfa
## Bartlett's K-squared = 1.4675, df = 1, p-value = 0.2257
```

También puede utilizarse el test de Wilcoxon como alternativa al test t

```
wilcox.test(fidelida ~ nsatisfa, data = hatco)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  fidelida by nsatisfa
## W = 430.5, p-value = 3.504e-08
## alternative hypothesis: true location shift is not equal to 0
```

Si disponemos de datos apareados, por ejemplo nivel de precios e imagen de fuerza de ventas

```
with(hatco, t.test(precio, imgfvent, paired = TRUE))
```

```
##
##  Paired t-test
##
## data:  precio and imgfvent
## t = -2.2347, df = 98, p-value = 0.02771
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.55114759 -0.03269079
## sample estimates:
## mean of the differences
##              -0.2919192
```

Y la correspondiente alternativa no paramétrica

```
with(hatco, wilcox.test(precio, imgfvent, paired = TRUE))
```

```
##
##  Wilcoxon signed rank test with continuity correction
##
## data:  precio and imgfvent
## V = 1789.5, p-value = 0.02431
## alternative hypothesis: true location shift is not equal to 0
```

6.3 Regresión y correlación

6.3.1 Regresión lineal simple

Utilizando la función *lm* (modelo lineal) se puede llevar a cabo, entre otras muchas cosas, una regresión lineal simple

```
lm(satisfac ~ fidelida, data = hatco)
```

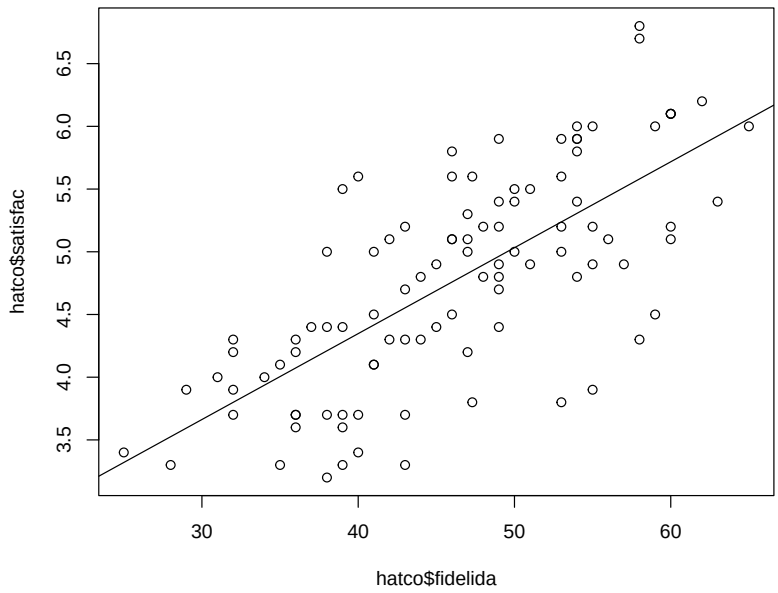
```
##
## Call:
## lm(formula = satisfac ~ fidelida, data = hatco)
##
## Coefficients:
## (Intercept)     fidelida
##      1.6074       0.0685
```

```
modelo <- lm(satisfac ~ fidelida, data = hatco, na.action=na.exclude)
summary(modelo)
```

```
##
## Call:
## lm(formula = satisfac ~ fidelida, data = hatco, na.action = na.exclude)
```

```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.47492 -0.37341  0.09358  0.38258  1.25258
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.607399   0.322436   4.985 2.71e-06 ***
## fidelida    0.068500   0.006848  10.003  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6058 on 97 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:  0.5078, Adjusted R-squared:  0.5027
## F-statistic: 100.1 on 1 and 97 DF,  p-value: < 2.2e-16
```

```
plot(hatco$fidelida, hatco$satisfac)      # Cuidado con el orden de las variables
# with(hatco, plot(fidelida, satisfac))   # Alternativa empleando with
# plot(satisfac ~ fidelida, data = hatco) # Alternativa empleando fórmulas
abline(modelo)
```



Valores ajustados

```
fitted(modelo)
```

```
##        1        2        3        4        5        6        7        8
## 3.799412 4.552917 4.895419 3.799412 5.580423 4.689918 4.758418 4.621417
##        9       10       11       12       13       14       15       16
## 5.922925 5.306421 3.799412 4.826919 4.278915 4.210415 5.306421 4.963919
##       17       18       19       20       21       22       23       24
## 4.210415 4.347416 5.306421 5.374922 4.415916 4.004913 5.374922 4.073414
##       25       26       27       28       29       30       31       32
## 4.963919 4.963919 4.073414 5.306421 4.963919 4.758418 4.552917 5.237921
##       33       34       35       36       37       38       39       40
## 5.717424 4.847469 4.004913 4.278915 4.621417 4.758418 3.593911 3.525410
```

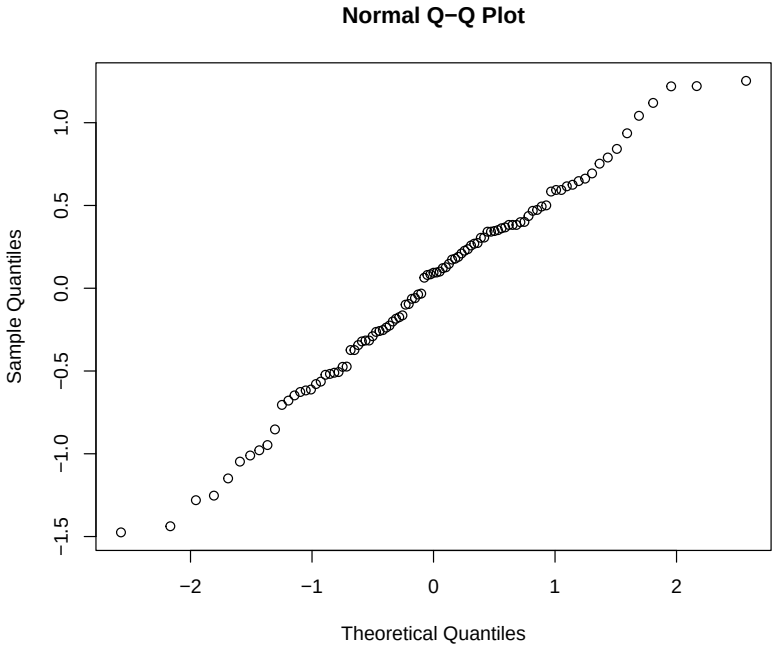
```
##       41       42       43       44       45       46       47       48
## 4.347416 5.580423 5.237921 4.895419 4.210415 5.306421 5.374922 4.552917
##       49       50       51       52       53       54       55       56
## 5.511923 5.237921 4.415916 5.237921 5.032420 3.799412 4.278915 4.826919
##       57       58       59       60       61       62       63       64
## 5.854425 6.059926 4.758418 5.032420 5.306421 5.717424 4.826919 4.073414
##       65       66       67       68       69       70       71       72
## 4.347416 4.689918 5.648924 4.758418 5.580423 4.963919 5.032420 5.374922
##       73       74       75       76       77       78       79       80
## 5.100920 5.717424 4.415916 4.963919 4.484416 4.826919 4.278915 5.443422
##       81       82       83       84       85       86       87       88
## 5.648924 4.847469 4.415916 4.141914 5.237921 4.552917 5.100920 4.073414
##       89       90       91       92       93       94       95       96
## 3.936413 5.717424 4.963919 4.278915 4.552917 4.073414 3.730912 3.319909
##       97       98       99      100
## 5.717424 4.210415 4.484416       NA
```

Residuos

```
head(resid(modelo))
```

```
##          1          2          3          4          5          6
##  0.4005878 -0.2529168  0.3045811  0.1005878  1.2195769 -0.2899177
```

```
qqnorm(resid(modelo))
```

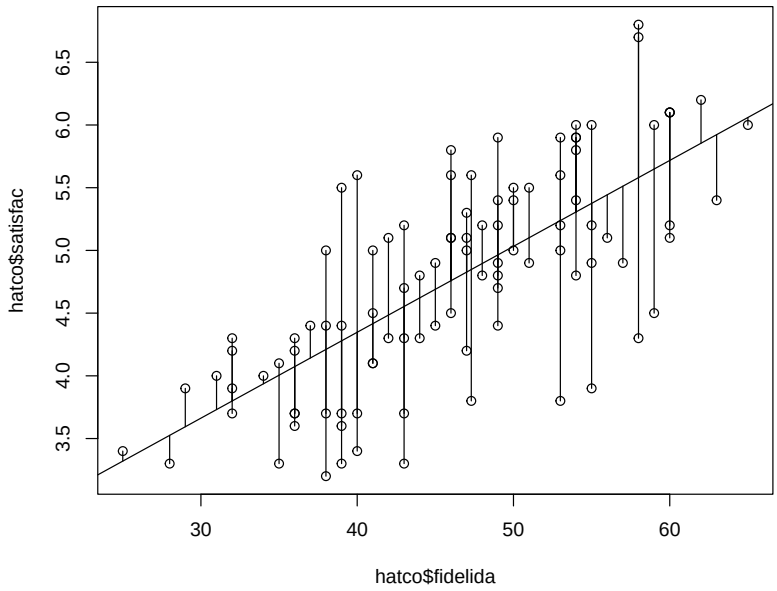



```
shapiro.test(resid(modelo))
```

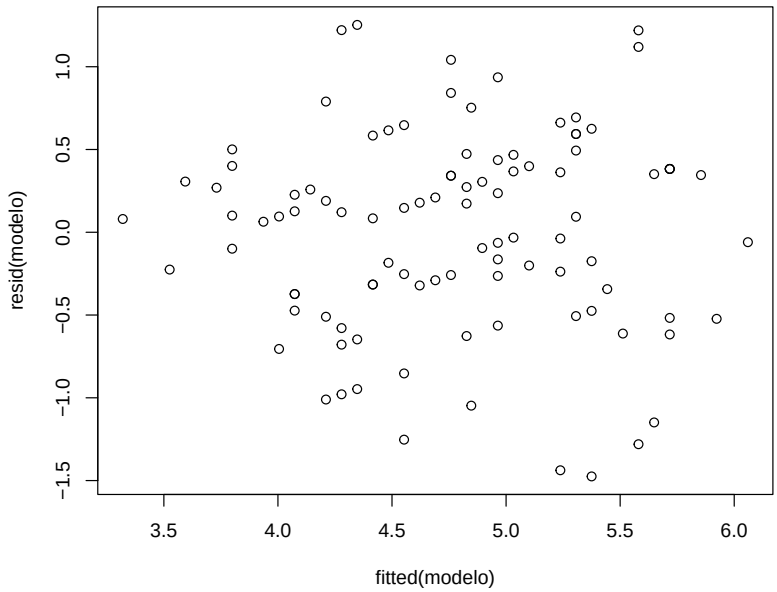
```
##
##  Shapiro-Wilk normality test
##
## data:  resid(modelo)
## W = 0.98515, p-value = 0.3325
```

```
plot(hatco$fidelida, hatco$satisfac)

abline(modelo)
```

```
# segments(hatco$fidelida, fitted(modelo), hatco$fidelida, hatco$satisfac)
with(hatco, segments(fidelida, fitted(modelo), fidelida, satisfac))
```




```
plot(fitted(modelo), resid(modelo))
```



Banda de confianza

```
predict(modelo, interval='confidence')
```

```
##          fit      lwr      upr
## 1   3.799412 3.571263 4.027561
## 2   4.552917 4.424306 4.681528
```

```
## 3   4.895419 4.772225 5.018613
## 4   3.799412 3.571263 4.027561
## 5   5.580423 5.380031 5.780815
## 6   4.689918 4.567906 4.811929
## 7   4.758418 4.637529 4.879307
## 8   4.621417 4.496801 4.746033
## 9   5.922925 5.665048 6.180803
## 10  5.306421 5.146011 5.466832
## 11  3.799412 3.571263 4.027561
## 12  4.826919 4.705631 4.948206
## 13  4.278915 4.123089 4.434741
## 14  4.210415 4.045670 4.375159
## 15  5.306421 5.146011 5.466832
## 16  4.963919 4.837379 5.090459
## 17  4.210415 4.045670 4.375159
## 18  4.347416 4.199793 4.495038
## 19  5.306421 5.146011 5.466832
## 20  5.374922 5.205264 5.544580
## 21  4.415916 4.275658 4.556174
## 22  4.004913 3.810147 4.199680
## 23  5.374922 5.205264 5.544580
## 24  4.073414 3.889113 4.257714
## 25  4.963919 4.837379 5.090459
## 26  4.963919 4.837379 5.090459
## 27  4.073414 3.889113 4.257714
## 28  5.306421 5.146011 5.466832
## 29  4.963919 4.837379 5.090459
## 30  4.758418 4.637529 4.879307
## 31  4.552917 4.424306 4.681528
## 32  5.237921 5.086103 5.389740
## 33  5.717424 5.494745 5.940103
## 34  4.847469 4.725765 4.969172
## 35  4.004913 3.810147 4.199680
## 36  4.278915 4.123089 4.434741
## 37  4.621417 4.496801 4.746033
## 38  4.758418 4.637529 4.879307
## 39  3.593911 3.330292 3.857530
## 40  3.525410 3.249642 3.801179
## 41  4.347416 4.199793 4.495038
## 42  5.580423 5.380031 5.780815
## 43  5.237921 5.086103 5.389740
## 44  4.895419 4.772225 5.018613
## 45  4.210415 4.045670 4.375159
## 46  5.306421 5.146011 5.466832
## 47  5.374922 5.205264 5.544580
## 48  4.552917 4.424306 4.681528
## 49  5.511923 5.322196 5.701650
## 50  5.237921 5.086103 5.389740
## 51  4.415916 4.275658 4.556174
## 52  5.237921 5.086103 5.389740
## 53  5.032420 4.901205 5.163635
## 54  3.799412 3.571263 4.027561
## 55  4.278915 4.123089 4.434741
## 56  4.826919 4.705631 4.948206
## 57  5.854425 5.608471 6.100378
## 58  6.059926 5.777748 6.342104
```

```
## 59  4.758418 4.637529 4.879307
## 60  5.032420 4.901205 5.163635
## 61  5.306421 5.146011 5.466832
## 62  5.717424 5.494745 5.940103
## 63  4.826919 4.705631 4.948206
## 64  4.073414 3.889113 4.257714
## 65  4.347416 4.199793 4.495038
## 66  4.689918 4.567906 4.811929
## 67  5.648924 5.437531 5.860316
## 68  4.758418 4.637529 4.879307
## 69  5.580423 5.380031 5.780815
## 70  4.963919 4.837379 5.090459
## 71  5.032420 4.901205 5.163635
## 72  5.374922 5.205264 5.544580
## 73  5.100920 4.963837 5.238003
## 74  5.717424 5.494745 5.940103
## 75  4.415916 4.275658 4.556174
## 76  4.963919 4.837379 5.090459
## 77  4.484416 4.350544 4.618289
## 78  4.826919 4.705631 4.948206
## 79  4.278915 4.123089 4.434741
## 80  5.443422 5.263964 5.622881
## 81  5.648924 5.437531 5.860316
## 82  4.847469 4.725765 4.969172
## 83  4.415916 4.275658 4.556174
## 84  4.141914 3.967647 4.316181
## 85  5.237921 5.086103 5.389740
## 86  4.552917 4.424306 4.681528
## 87  5.100920 4.963837 5.238003
## 88  4.073414 3.889113 4.257714
## 89  3.936413 3.730815 4.142011
## 90  5.717424 5.494745 5.940103
## 91  4.963919 4.837379 5.090459
## 92  4.278915 4.123089 4.434741
## 93  4.552917 4.424306 4.681528
## 94  4.073414 3.889113 4.257714
## 95  3.730912 3.491126 3.970697
## 96  3.319909 3.006980 3.632839
## 97  5.717424 5.494745 5.940103
## 98  4.210415 4.045670 4.375159
## 99  4.484416 4.350544 4.618289
## 100       NA       NA       NA
```

Banda de predicción

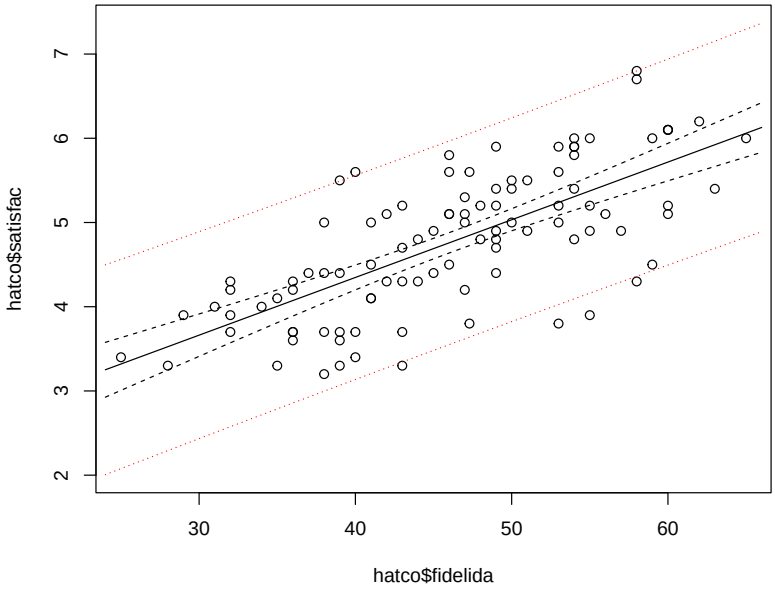
```
head(predict(modelo, interval='prediction'))
```

```
##        fit      lwr      upr
## 1 3.799412 2.575563 5.023261
## 2 4.552917 3.343663 5.762171
## 3 4.895419 3.686729 6.104109
## 4 3.799412 2.575563 5.023261
## 5 5.580423 4.361444 6.799403
## 6 4.689918 3.481348 5.898487
```

Representación gráfica de las bandas

```
bandas.frame <- data.frame(fidelida=24:66)
bc <- predict(modelo, interval = 'confidence', newdata = bandas.frame)
```

```
bp <- predict(modelo, interval = 'prediction', newdata = bandas.frame)
plot(hatco$fidelida, hatco$satisfac, ylim = range(hatco$satisfac, bp, na.rm = TRUE))
matlines(bandas.frame$fidelida, bc, lty=c(1,2,2), col='black')
matlines(bandas.frame$fidelida, bp, lty=c(0,3,3), col='red')
```



6.3.2 Correlación

Coeficiente de correlación de Pearson

```
cor(hatco$fidelida, hatco$satisfac, use='complete.obs')
```

```
## [1] 0.712581
```

```
cor(hatco[,6:14], use='complete.obs')
```

```
##             velocida      precio    flexprec    imgfabri    servconj
## velocida  1.00000000 -0.35439461  0.51879732  0.04885481  0.60908594
## precio   -0.35439461  1.00000000 -0.48550163  0.27150666  0.51134698
## flexprec  0.51879732 -0.48550163  1.00000000 -0.11472112  0.07496499
## imgfabri  0.04885481  0.27150666 -0.11472112  1.00000000  0.29800272
## servconj  0.60908594  0.51134698  0.07496499  0.29800272  1.00000000
## imgfvent  0.08084452  0.18873090 -0.03801323  0.79015164  0.24641510
## calidadp -0.48984768  0.46822563 -0.44542562  0.19904126 -0.06152068
## fidelida  0.67428681  0.07682487  0.57807750  0.22442574  0.69802972
## satisfac  0.64981476  0.02636286  0.53057615  0.47553688  0.63054720
##             imgfvent    calidadp    fidelida    satisfac
## velocida  0.08084452 -0.48984768  0.67428681  0.64981476
## precio    0.18873090  0.46822563  0.07682487  0.02636286
## flexprec -0.03801323 -0.44542562  0.57807750  0.53057615
## imgfabri  0.79015164  0.19904126  0.22442574  0.47553688
## servconj  0.24641510 -0.06152068  0.69802972  0.63054720
## imgfvent  1.00000000  0.18052945  0.26674626  0.34349253
## calidadp  0.18052945  1.00000000 -0.20401261 -0.28687427
## fidelida  0.26674626 -0.20401261  1.00000000  0.71258104
## satisfac  0.34349253 -0.28687427  0.71258104  1.00000000
```

```
cor.test(hatco$fidelida, hatco$satisfac)
```

```
##
##  Pearson's product-moment correlation
##
## data:  hatco$fidelida and hatco$satisfac
## t = 10.003, df = 97, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.5995024 0.7977691
## sample estimates:
##      cor
## 0.712581
```

El coeficiente de correlación de Spearman es una variante no paramétrica

```
cor.test(hatco$fidelida, hatco$satisfac, method='spearman')
```

```
##
##  Spearman's rank correlation rho
##
## data:  hatco$fidelida and hatco$satisfac
## S = 46601, p-value < 2.2e-16
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##       rho
## 0.7118039
```

6.4 Análisis de la varianza

6.4.1 ANOVA con un factor

Vamos a estudiar si hay diferencias en las medias de la variable *satisfac* (satisfacción global) entre los diferentes grupos definidos por *nfidelid* (nivel de compra), utilizando el procedimiento clásico de análisis de la varianza. Este procedimiento exige normalidad y homocedasticidad.

```
table(hatco$nfidelid)
```

```
##
##  bajo medio  alto
##     3    64    33
```

```
tapply(hatco$satisfac, hatco$nfidelid, mean, na.rm = TRUE)
```

```
##     bajo    medio     alto
## 3.533333 4.498437 5.443750
```

La variable explicativa tiene que ser obligatoriamente de tipo *factor*. Por coherencia con la función (general) *lm*, la variación entre grupos está etiquetada *nfidelid*, y la variación dentro de los grupos como *Residuals*

```
anova(lm(satisfac~nfidelid, data = hatco))
```

```
## Analysis of Variance Table
##
## Response: satisfac
##           Df Sum Sq Mean Sq F value    Pr(>F)
## nfidelid   2 23.832 11.9158  23.588 4.647e-09 ***
## Residuals 96 48.495  0.5052
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Como alternativa, se puede utilizar la función *aov*

```
aov(satisfac~nfidelid, data = hatco)
```

```
## Call:
##    aov(formula = satisfac ~ nfidelid, data = hatco)
##
## Terms:
##                 nfidelid Residuals
## Sum of Squares  23.83161  48.49526
## Deg. of Freedom        2        96
##
## Residual standard error: 0.7107454
## Estimated effects may be unbalanced
## 1 observation deleted due to missingness
```

```
summary(aov(satisfac~nfidelid, data = hatco))
```

```
##             Df Sum Sq Mean Sq F value   Pr(>F)
## nfidelid     2  23.83  11.916   23.59 4.65e-09 ***
## Residuals   96  48.50   0.505
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 1 observation deleted due to missingness
```

Comparaciones entre pares de variables

```
pairwise.t.test(hatco$satisfac, hatco$nfidelid)
```

```
##
##  Pairwise comparisons using t tests with pooled SD
##
## data:  hatco$satisfac and hatco$nfidelid
##
##       bajo    medio
## medio 0.024   -
## alto  4.6e-05 5.5e-08
##
## P value adjustment method: holm
```

Relajamos la hipótesis de varianzas iguales

```
oneway.test(satisfac~nfidelid, data = hatco)
```

```
##
##  One-way analysis of means (not assuming equal variances)
##
## data:  satisfac and nfidelid
## F = 35.013, num df = 2.0000, denom df = 6.7661, p-value = 0.0002697
```

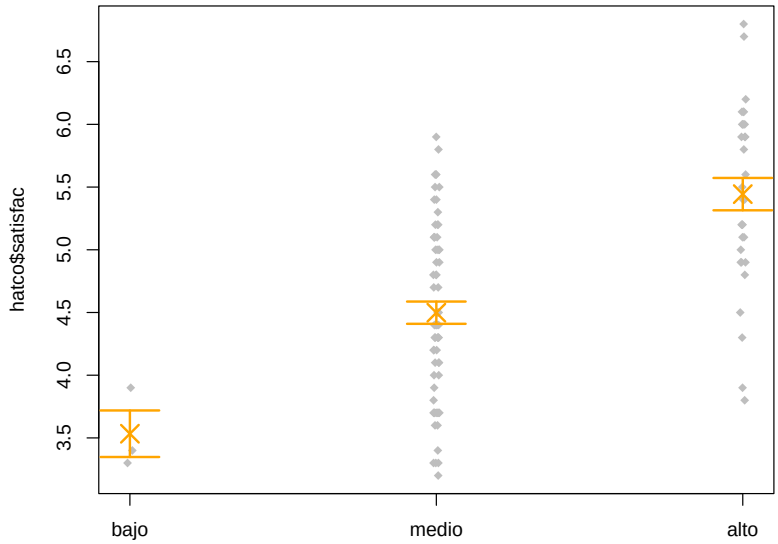
Podemos utilizar el test de Bartlett para contrastar la igualdad de varianzas

```
bartlett.test(satisfac~nfidelid, data = hatco)
```

```
##
##  Bartlett test of homogeneity of variances
##
## data:  satisfac by nfidelid
## Bartlett's K-squared = 1.4922, df = 2, p-value = 0.4742
```

Representación gráfica

```
medias <- tapply(hatco$satisfac, hatco$nfidelid, mean, na.rm = TRUE)
desviaciones <- tapply(hatco$satisfac, hatco$nfidelid, sd, na.rm = TRUE)
n <- tapply(hatco$satisfac[!is.na(hatco$satisfac)], hatco$nfidelid[!is.na(hatco$satisfac)], length
errores <- desviaciones/sqrt(n)
stripchart(hatco$satisfac~hatco$nfidelid, method='jitter', jit=0.01, pch=18, col='grey', vertical
arrows(1:3, medias+errores, 1:3, medias-errores, angle=90, code=3, lwd=2, col='orange')
points(1:3, medias, pch=4, lwd=2, cex=2, col='orange')
```



6.4.2 Test de Kruskal-Wallis

Alternativa no paramétrica al análisis de la varianza con un factor

```
kruskal.test(satisfac~nfidelid, data = hatco)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  satisfac by nfidelid
## Kruskal-Wallis chi-squared = 31.073, df = 2, p-value = 1.789e-07
```

Capítulo 7

Modelado de datos

La realidad puede ser muy compleja por lo que es habitual emplear un modelo para tratar de explicarla.

- Modelos estocásticos (con componente aleatoria).
  - Tienen en cuenta la incertidumbre debida a no disponer de la suficiente información sobre las variables que influyen en el fenómeno en estudio.
  - La inferencia estadística proporciona herramientas para ajustar y contrastar la validez del modelo a partir de los datos observados.

Sin embargo resultaría muy extraño que la realidad coincida exactamente con un modelo concreto.

- George Box afirmó en su famoso aforismo:

  > En esencia, todos los modelos son falsos, pero algunos son útiles.

- El objetivo de un modelo es disponer de una aproximación simple de la realidad que sea útil.

7.1 Modelos de regresión

Nos centraremos en los modelos de regresión:

$$Y = f(X_1, \cdots, X_p) + \varepsilon$$

donde:

- $Y \equiv$ **variable respuesta** (o dependiente).
- $(X_1, \cdots, X_p) \equiv$ **variables explicativas** (independientes, o covariables).
- $\varepsilon \equiv$ **error aleatorio.**

7.1.1 Herramientas disponibles en `R`

`R` dispone de múltiples herramientas para trabajar con modelos de este tipo. Algunas de las funciones y paquetes disponibles se muestran a continuación:

- Modelos paramétricos:
  - Modelos lineales:
    - Regresión lineal: `lm()` (`aov()`, `lme()`, `biglm`, …).
    - Regresión lineal robusta: `MASS::rlm()`.
    - Métodos de regularización (Ridge regression, Lasso): `glmnet`, …
  - Modelos lineales generalizados: `glm()` (`bigglm`, …).

  - Modelos paramétricos no lineales: `nls()` (`nlme`, …).

- Modelos no paramétricos:
  - Regresión local (métodos de suavizado): `loess()`, `KernSmooth`, `sm`, …
  - Modelos aditivos generalizados (GAM): `gam`, `mgcv`, …
  - Arboles de decisión (Random Forest, Boosting): `rpart`, `randomForest`, `xgboost`, …
  - Redes neuronales, …

Desde el punto de vista de la programación, con todos estos modelos se trabaja de una forma muy similar en `R`.

7.2 Fórmulas

En `R` para especificar un modelo estadístico (realmente una familia) se suelen emplear fórmulas (también para generar gráficos). Son de la forma:

```
respuesta ~ modelo
```

`modelo` especifica los "términos" mediante operadores (tienen un significado especial en este contexto):

Operador	Descripción
`a+b`	incluye `a` y `b` (efectos principales)
`-b`	excluye `b` del modelo
`a:b`	interacción de `a` y `b`
\	`b %in% a` efectos de `b` anidados en `a` (`a:b`)
\	`a/b = a + b %in% a = a + a:b`
`a*b = a+b+a:b`	efectos principales más interacciones
`^n`	interacciones hasta nivel `n` (`(a+b)^2 = a+b+a:b`)
`poly(a, n)`	polinomios de `a` hasta grado `n`
`1`	término constante
`.`	todas las variables disponibles o modelo actual en actualizaciones

Para realizar operaciones aritméticas (que incluyan `+`, `-`, `*`, `^`, `1`, …) es necesario "aislar" la operación dentro una función (e.g. `log(abs(x) + 1)`). Por ejemplo, para realizar un ajuste cuadrático se debería utilizar `y ~ x + I(x^2)`, ya que `y ~ x + x^2 = y ~ x` (la interacción `x:x = x`).

- `I()` función identidad.

7.3 Ejemplo: regresión lineal simple

Introducido en descriptiva y con referencias al tema siguiente

Capítulo 8

Modelos lineales

Suponen que la función de regresión es lineal:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \varepsilon$$

El efecto de las variables explicativas sobre la respuesta es simple (proporcional a su valor).

8.1 Ejemplo

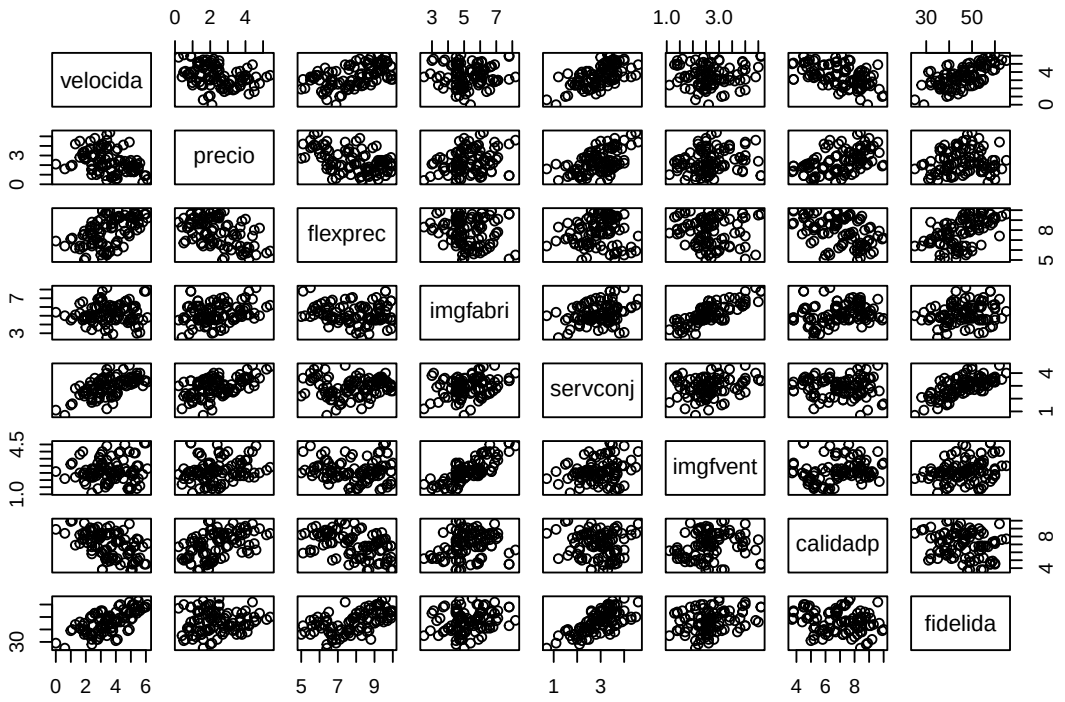
El fichero *hatco.RData* contiene observaciones de clientes de la compañía de distribución industrial (Compañía Hair, Anderson y Tatham). Las variables se pueden clasificar en tres grupos:

```
load('datos/hatco.RData')
as.data.frame(attr(hatco, "variable.labels"))
```

```
##          attr(hatco, "variable.labels")
## empresa                         Empresa
## tamano              Tamaño de la empresa
## adquisic       Estructura de adquisición
## tindustr                Tipo de industria
## tsitcomp     Tipo de situación de compra
## velocida             Velocidad de entrega
## precio                   Nivel de precios
## flexprec         Flexibilidad de precios
## imgfabri           Imagen del fabricante
## servconj               Servicio conjunto
## imgfvent      Imagen de fuerza de ventas
## calidadp              Calidad de producto
## fidelida   Porcentaje de compra a HATCO
## satisfac              Satisfacción global
## nfidelid         Nivel de compra a HATCO
## nsatisfa           Nivel de satisfacción
```

Consideraremos como respuesta la variable *fidelida* y como variables explicativas el resto de variables continuas menos *satisfac*.

```
datos <- hatco[, 6:13]  # Nota: realmente no copia el objeto...
plot(datos)
```




```
# cor(datos, use = "complete") # Por defecto 8 decimales...
print(cor(datos, use = "complete"), digits = 2)
```

```
##          velocida precio flexprec imgfabri servconj imgfvent calidadp fidelida
## velocida    1.000 -0.354    0.519    0.049    0.609    0.081   -0.490    0.674
## precio     -0.354  1.000   -0.486    0.272    0.511    0.189    0.468    0.077
## flexprec    0.519 -0.486    1.000   -0.115    0.075   -0.038   -0.445    0.578
## imgfabri    0.049  0.272   -0.115    1.000    0.298    0.790    0.199    0.224
## servconj    0.609  0.511    0.075    0.298    1.000    0.246   -0.062    0.698
## imgfvent    0.081  0.189   -0.038    0.790    0.246    1.000    0.181    0.267
## calidadp   -0.490  0.468   -0.445    0.199   -0.062    0.181    1.000   -0.204
## fidelida    0.674  0.077    0.578    0.224    0.698    0.267   -0.204    1.000
```

8.2 Ajuste: función `lm`

Para el ajuste (estimación de los parámetros) de un modelo lineal a un conjunto de datos (por mínimos cuadrados) se emplea la función `lm`:

```
ajuste <- lm(formula, datos, seleccion, pesos, na.action)
```

- `formula` fórmula que especifica el modelo.
- `datos` data.frame opcional con las variables de la formula.
- `seleccion` especificación opcional de un subconjunto de observaciones.
- `pesos` vector opcional de pesos (WLS).
- `na.action` opción para manejar los datos faltantes (`na.omit`).

```
modelo <- lm(fidelida ~ servconj, datos)
modelo
```

```
##
## Call:
## lm(formula = fidelida ~ servconj, data = datos)
```

```
##
## Coefficients:
## (Intercept)     servconj
##       21.98         8.30
```

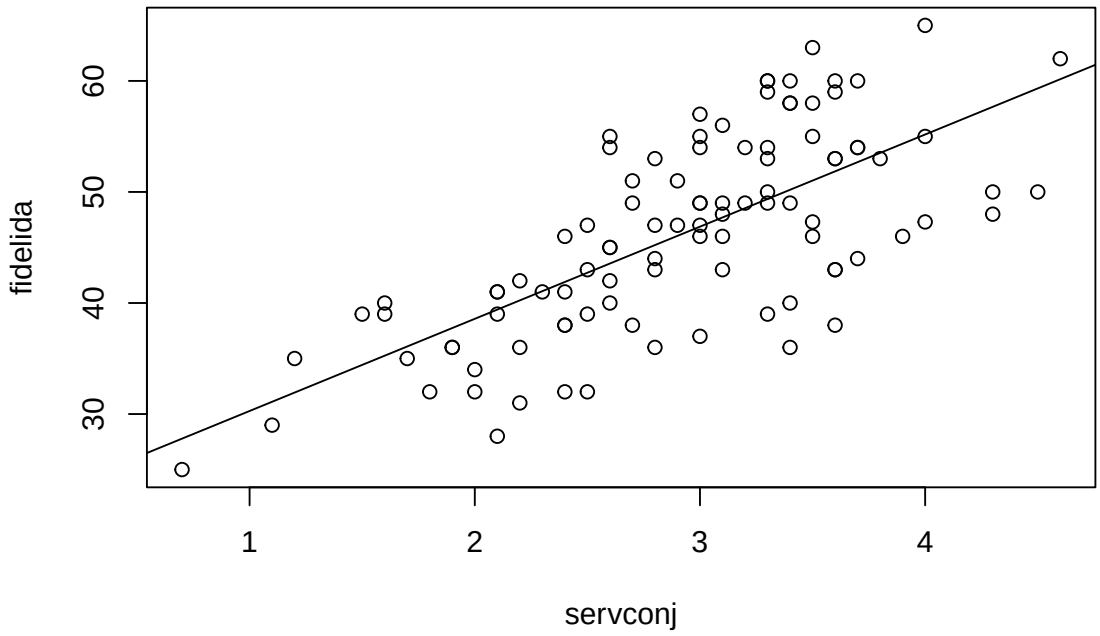
Al imprimir el ajuste resultante se muestra un pequeño resumen del ajuste (aunque el objeto que contiene los resultados es una lista).

Para obtener un resumen más completo se puede utilizar la función `summary()`.

```
summary(modelo)
```

```
##
## Call:
## lm(formula = fidelida ~ servconj, data = datos)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.1956  -4.0655   0.2944   4.5945  11.9744
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  21.9754     2.6086   8.424 3.34e-13 ***
## servconj      8.3000     0.8645   9.601 9.76e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.432 on 97 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:  0.4872, Adjusted R-squared:  0.482
## F-statistic: 92.17 on 1 and 97 DF,  p-value: 9.765e-16
```

```
plot(fidelida ~ servconj, datos)
abline(modelo)
```



8.2.1 Extracción de información

Para la extracción de información se pueden acceder a los componentes del modelo ajustado o emplear funciones (genéricas). Algunas de las más utilizadas son las siguientes:

Función	Descripción
`fitted`	valores ajustados
`coef`	coeficientes estimados (y errores estándar)
`confint`	intervalos de confianza para los coeficientes
`residuals`	residuos
`plot`	gráficos de diagnóstico
`termplot`	gráfico de efectos parciales
`anova`	calcula tablas de análisis de varianza (también permite comparar modelos)
`predict`	calcula predicciones para nuevos datos

Ejemplo:

```
modelo2 <- lm(fidelida ~ servconj + flexprec, data = hatco)
summary(modelo2)
```

```
##
## Call:
## lm(formula = fidelida ~ servconj + flexprec, data = hatco)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.2549  -2.2850   0.3411   3.3260   7.0853
##
## Coefficients:
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -3.4617     2.9734  -1.164    0.247
## servconj      7.8287     0.5897  13.276   <2e-16 ***
## flexprec      3.4017     0.3191  10.661   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.375 on 96 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:  0.7652, Adjusted R-squared:  0.7603
## F-statistic: 156.4 on 2 and 96 DF,  p-value: < 2.2e-16
```

```
confint(modelo2)
```

```
##                 2.5 %   97.5 %
## (Intercept) -9.363813 2.440344
## servconj     6.658219 8.999274
## flexprec     2.768333 4.035030
```

```
anova(modelo2)
```

```
## Analysis of Variance Table
##
## Response: fidelida
##           Df Sum Sq Mean Sq F value    Pr(>F)
## servconj   1 3813.6  3813.6  199.23 < 2.2e-16 ***
## flexprec   1 2175.6  2175.6  113.66 < 2.2e-16 ***
## Residuals 96 1837.6    19.1
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# anova(modelo2, modelo)
# termplot(modelo2, partial.resid = TRUE)
```

Muchas de estas funciones genéricas son válidas para otros tipos de modelos (glm, ...).

Algunas funciones como `summary()` devuelven información adicional:

```
res <- summary(modelo2)
names(res)
```

```
##  [1] "call"          "terms"         "residuals"     "coefficients"
##  [5] "aliased"       "sigma"         "df"            "r.squared"
##  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"  "na.action"
```

```
res$sigma
```

```
## [1] 4.375074
```

```
res$adj.r.squared
```

```
## [1] 0.7603292
```

8.3 Predicción

Para calcular predicciones (estimaciones de la media condicionada) se puede emplear la función `predict()` (ejecutar `help(predict.lm)` para ver todas las opciones disponibles). Por defecto obtiene las predicciones correspondientes a las observaciones (`modelo$fitted.values`). Para otros casos hay que emplear el argumento `newdata`:

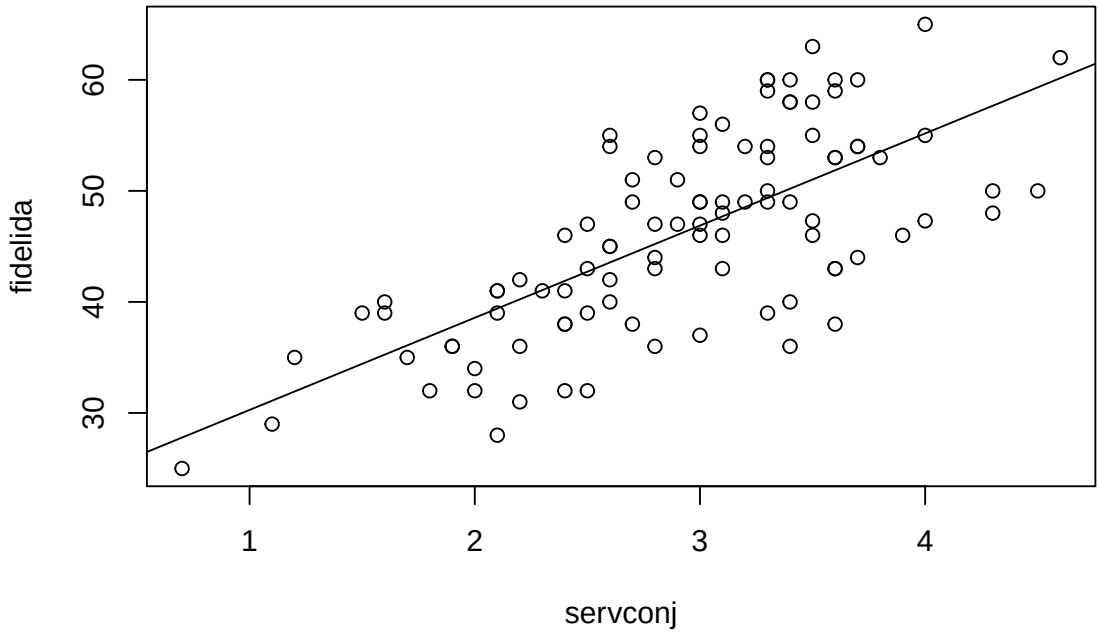
- data.frame con los valores de (todas) las covariables, sus nombres deben coincidir con los originales.

Ejemplo:

```
valores <- 0:5
pred <- predict(modelo, newdata = data.frame(servconj = valores))
pred
```

```
##        1        2        3        4        5        6
## 21.97544 30.27548 38.57552 46.87556 55.17560 63.47564
```

```
plot(fidelida ~ servconj, datos)
lines(valores, pred)
```

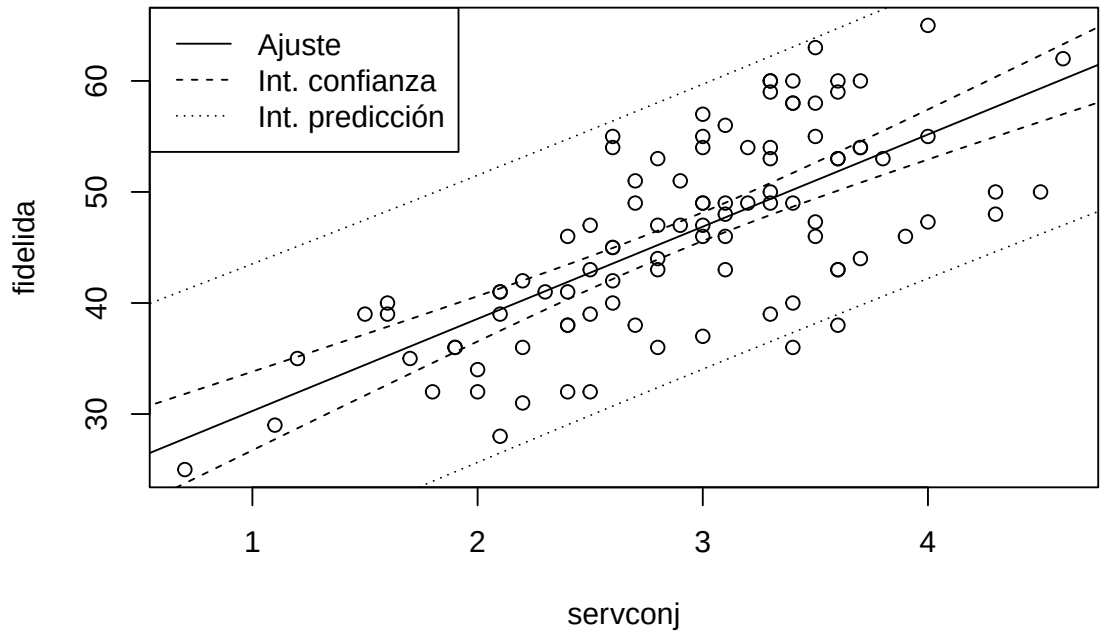


Esta función también permite obtener intervalos de confianza y de predicción:

```
valores <- seq(0, 5, len = 100)
newdata <- data.frame(servconj = valores)
pred <- predict(modelo, newdata = newdata, interval = c("confidence"))
head(pred)
```

```
##        fit      lwr      upr
## 1 21.97544 16.79816 27.15272
## 2 22.39463 17.30126 27.48800
## 3 22.81383 17.80427 27.82338
## 4 23.23302 18.30718 28.15886
## 5 23.65221 18.80999 28.49444
## 6 24.07141 19.31269 28.83013
```

```
plot(fidelida ~ servconj, datos)
matlines(valores, pred, lty = c(1, 2, 2), col = 1)
pred2 <- predict(modelo, newdata = newdata, interval = c("prediction"))
matlines(valores, pred2[, -1], lty = 3, col = 1)
legend("topleft", c("Ajuste", "Int. confianza", "Int. predicción"), lty = c(1, 2, 3))
```



8.4 Selección de variables explicativas

Cuando se dispone de un conjunto grande de posibles variables explicativas suele ser especialmente importante determinar cuales de estas deberían ser incluidas en el modelo de regresión. Si alguna de las variables no contiene información relevante sobre la respuesta no se debería incluir (se simplificaría la interpretación del modelo, aumentaría la precisión de la estimación y se evitarían problemas como la multicolinealidad). Se trataría entonces de conseguir un buen ajuste con el menor número de variables explicativas posible.

Para actualizar un modelo (p.e. eliminando o añadiendo variables) se puede emplear la función `update`:

```
modelo.completo <- lm(fidelida ~ . , data = datos)
summary(modelo.completo)
```

```
##
## Call:
## lm(formula = fidelida ~ ., data = datos)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.3351  -2.0733   0.5224   2.9218   6.7106
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -9.5935     4.8213  -1.990   0.0496 *
## velocida     -0.6023     1.9590  -0.307   0.7592
## precio       -1.0771     2.0283  -0.531   0.5967
## flexprec      3.4616     0.3997   8.660 1.62e-13 ***
## imgfabri     -0.1735     0.6472  -0.268   0.7892
## servconj      9.0919     3.8023   2.391   0.0189 *
## imgfvent      1.5596     0.9221   1.691   0.0942 .
```

```
## calidadp      0.4874     0.3451   1.412   0.1613
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.281 on 91 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:  0.7869, Adjusted R-squared:  0.7705
## F-statistic:    48 on 7 and 91 DF,  p-value: < 2.2e-16
```

```
modelo.reducido <- update(modelo.completo, . ~ . - imgfabri)
summary(modelo.reducido)
```

```
##
## Call:
## lm(formula = fidelida ~ velocida + precio + flexprec + servconj +
##     imgfvent + calidadp, data = datos)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.2195  -2.0022   0.4724   2.9514   6.8328
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -9.9900     4.5656  -2.188   0.0312 *
## velocida     -0.5207     1.9254  -0.270   0.7874
## precio       -1.0017     1.9986  -0.501   0.6174
## flexprec      3.4709     0.3962   8.761 9.23e-14 ***
## servconj      8.9111     3.7230   2.394   0.0187 *
## imgfvent      1.3699     0.5883   2.329   0.0221 *
## calidadp      0.4844     0.3432   1.411   0.1615
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.26 on 92 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:  0.7867, Adjusted R-squared:  0.7728
## F-statistic: 56.56 on 6 and 92 DF,  p-value: < 2.2e-16
```

Para obtener el modelo "óptimo" lo ideal sería evaluar todos los modelos posibles.

8.4.1 Búsqueda exhaustiva

La función `regsubsets` del paquete `leaps` permite seleccionar los mejores modelos fijando el número de variables explicativas. Por defecto, evalúa todos los modelos posibles con un determinado número de parámetros (variando desde 1 hasta un máximo de `nvmax=8`) y selecciona el mejor (`nbest=1`).

```
library(leaps)
res <- regsubsets(fidelida ~ . , data = datos)
summary(res)
```

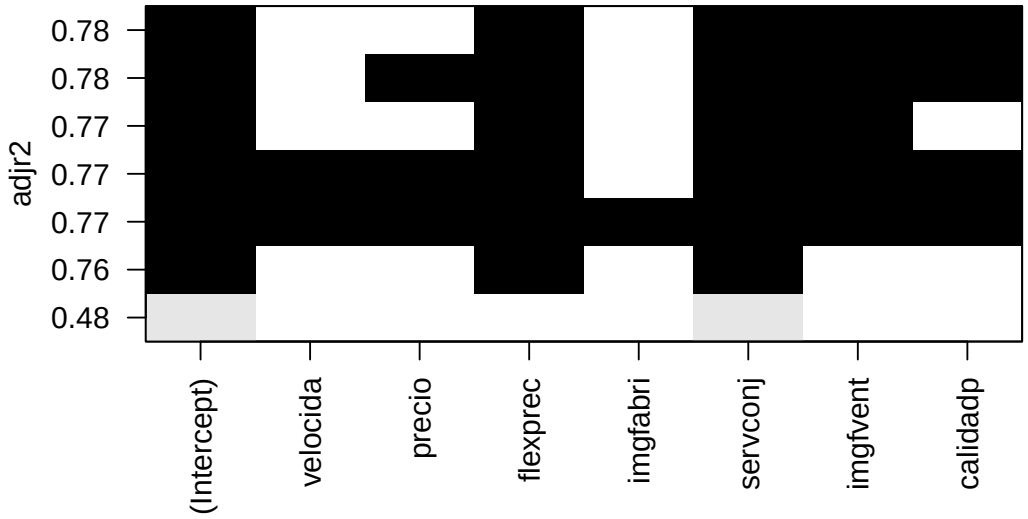
```
## Subset selection object
## Call: regsubsets.formula(fidelida ~ ., data = datos)
## 7 Variables  (and intercept)
##          Forced in Forced out
## velocida     FALSE      FALSE
## precio       FALSE      FALSE
## flexprec     FALSE      FALSE
## imgfabri     FALSE      FALSE
## servconj     FALSE      FALSE
```

```
## imgfvent     FALSE      FALSE
## calidadp     FALSE      FALSE
## 1 subsets of each size up to 7
## Selection Algorithm: exhaustive
##          velocida precio flexprec imgfabri servconj imgfvent calidadp
## 1  ( 1 ) " "      " "    " "      " "      "*"      " "      " "
## 2  ( 1 ) " "      " "    "*"      " "      "*"      " "      " "
## 3  ( 1 ) " "      " "    "*"      " "      "*"      "*"      " "
## 4  ( 1 ) " "      " "    "*"      " "      "*"      "*"      "*"
## 5  ( 1 ) " "      "*"    "*"      " "      "*"      "*"      "*"
## 6  ( 1 ) "*"      "*"    "*"      " "      "*"      "*"      "*"
## 7  ( 1 ) "*"      "*"    "*"      "*"      "*"      "*"      "*"
```

```
# names(summary(res))
```

Al representar el resultado se obtiene un gráfico con los mejores modelos ordenados según el criterio determinado por el argumento `scale = c("bic", "Cp", "adjr2", "r2")`. Por ejemplo, en este caso, empleando el coeficiente de determinación ajustado, obtendríamos:

```
plot(res, scale = "adjr2")
```



En este caso (considerando que una mejora del 2% no es significativa), el modelo resultante sería:

```
lm(fidelida ~ servconj + flexprec, data = hatco)
```

```
##
## Call:
## lm(formula = fidelida ~ servconj + flexprec, data = hatco)
##
## Coefficients:
## (Intercept)     servconj     flexprec
##      -3.462        7.829        3.402
```

Notas:

- Si se emplea alguno de los criterios habituales, el mejor modelo con un determinado número de variables no depende del criterio empleado. Pero estos criterios pueden diferir al comparar modelos con distinto número de variables explicativas.
- Si el número de variables explicativas es grande, en lugar de emplear una búsqueda exhaustiva se puede emplear un criterio por pasos, mediante el argumento `method = c("backward", "forward", "seqrep")`, pero puede ser recomendable emplear el paquete `MASS` para obtener directamente el modelo final.

8.4.2 Selección por pasos

Si el número de variables es grande (no sería práctico evaluar todas las posibilidades) se suele utilizar alguno (o varios) de los siguientes métodos:

- **Selección progresiva** (forward): Se parte de una situación en la que no hay ninguna variable y en cada paso se incluye una aplicando un **criterio de entrada** (hasta que ninguna de las restantes lo verifican).
- **Eliminación progresiva** (backward): Se parte del modelo con todas las variables y en cada paso se elimina una aplicando un **criterio de salida** (hasta que ninguna de las incluidas lo verifican).
- **Regresión paso a paso** (stepwise): El más utilizado, se combina un criterio de entrada y uno de salida. Normalmente se parte sin ninguna variable y **en cada paso puede haber una inclusión y una exclusión** (forward/backward).

La función `stepAIC` del paquete `MASS` permite seleccionar el modelo por pasos, hacia delante o hacia atrás según criterio AIC o BIC (también esta disponible una función `step` del paquete base `stats` con menos opciones). La función `stepwise` del paquete `RcmdrMisc` es una interfaz de `stepAIC` que facilita su uso:

```
library(MASS)
library(RcmdrMisc)
modelo <- stepwise(modelo.completo, direction = "forward/backward", criterion = "BIC")
```

```
##
## Direction:  forward/backward
## Criterion:  BIC
##
## Start:  AIC=437.24
## fidelida ~ 1
##
##            Df Sum of Sq    RSS    AIC
## + servconj  1    3813.6 4013.2 375.71
## + velocida  1    3558.5 4268.2 381.81
## + flexprec  1    2615.5 5211.3 401.57
## + imgfvent  1     556.9 7269.9 434.53
## + imgfabri  1     394.2 7432.5 436.72
## <none>                  7826.8 437.24
## + calidadp  1     325.8 7501.0 437.63
## + precio    1      46.2 7780.6 441.25
##
## Step:  AIC=375.71
## fidelida ~ servconj
##
##            Df Sum of Sq    RSS    AIC
## + flexprec  1    2175.6 1837.6 302.97
## + precio    1     831.5 3181.7 357.32
```

```
## + velocida  1     772.3 3240.9 359.15
## + calidadp  1     203.8 3809.4 375.15
## <none>                  4013.2 375.71
## + imgfvent  1      74.8 3938.4 378.44
## + imgfabri  1       2.3 4010.9 380.25
## - servconj  1    3813.6 7826.8 437.24
##
## Step:  AIC=302.97
## fidelida ~ servconj + flexprec
##
##            Df Sum of Sq    RSS    AIC
## + imgfvent  1     129.8 1707.7 300.31
## <none>                  1837.6 302.97
## + imgfabri  1      69.3 1768.3 303.76
## + calidadp  1      50.7 1786.9 304.80
## + precio    1       0.2 1837.4 307.56
## + velocida  1       0.0 1837.5 307.57
## - flexprec  1    2175.6 4013.2 375.71
## - servconj  1    3373.7 5211.3 401.57
##
## Step:  AIC=300.31
## fidelida ~ servconj + flexprec + imgfvent
##
##            Df Sum of Sq    RSS    AIC
## <none>                  1707.7 300.31
## - imgfvent  1    129.82 1837.6 302.97
## + calidadp  1     24.70 1683.0 303.47
## + precio    1      0.96 1706.8 304.85
## + imgfabri  1      0.66 1707.1 304.87
## + velocida  1      0.41 1707.3 304.88
## - flexprec  1   2230.67 3938.4 378.44
## - servconj  1   2850.14 4557.9 392.91
```

```
summary(modelo)
```

```
##
## Call:
## lm(formula = fidelida ~ servconj + flexprec + imgfvent, data = datos)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -12.9301  -2.1395   0.0695   2.9632   7.4286
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -6.7761     3.1343  -2.162   0.0331 *
## servconj      7.4320     0.5902  12.592   <2e-16 ***
## flexprec      3.4503     0.3097  11.140   <2e-16 ***
## imgfvent      1.5369     0.5719   2.687   0.0085 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.24 on 95 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:  0.7818, Adjusted R-squared:  0.7749
## F-statistic: 113.5 on 3 and 95 DF,  p-value: < 2.2e-16
```

Los métodos disponibles son `"backward/forward"`, `"forward/backward"`, `"backward"` y `"forward"`.

Cuando el número de variables explicativas es muy grande (o si el tamaño de la muestra es pequeño en comparación) pueden aparecer problemas al emplear los métodos anteriores (incluso pueden no ser aplicables). Una alternativa son los métodos de regularización (Ridge regression, Lasso) disponibles en el paquete `glmnet`.

8.5 Regresión con variables categóricas

La función `lm()` admite también variables categóricas (factores), lo que equivaldría a modelos de análisis de la varianza o de la covarianza.

Como ejemplo, en el resto del tema emplearemos los datos de empleados:

```
load("datos/empleados.RData")
datos <- with(empleados, data.frame(lnsal = log(salario), lnsalini = log(salini), catlab, sexo))
```

Al incluir variables categóricas la función `lm()` genera las variables indicadoras (variables dummy) que sean necesarias. Por ejemplo, la función `model.matrix()` construye la denominada matriz de diseño X de un modelo lineal:

$$\mathbf{Y} = X\ +$$

En el caso de una variable categórica, por defecto se toma la primera categoría como referencia y se generan variables indicadoras del resto de categorías:

```
X <- model.matrix(lnsal ~ catlab, datos)
head(X)
```

```
##   (Intercept) catlabSeguridad catlabDirectivo
## 1           1               0               1
## 2           1               0               0
## 3           1               0               0
## 4           1               0               0
## 5           1               0               0
## 6           1               0               0
```

En el correspondiente ajuste (análisis de la varianza de un factor):

```
modelo <- lm(lnsal ~ catlab, datos)
summary(modelo)
```

```
##
## Call:
## lm(formula = lnsal ~ catlab, data = datos)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.58352 -0.15983 -0.01012  0.13277  1.08725
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)     10.20254    0.01280 797.245  < 2e-16 ***
## catlabSeguridad  0.13492    0.04864   2.774  0.00576 **
## catlabDirectivo  0.82709    0.02952  28.017  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2438 on 471 degrees of freedom
## Multiple R-squared:  0.625,  Adjusted R-squared:  0.6234
## F-statistic: 392.6 on 2 and 471 DF,  p-value: < 2.2e-16
```

el nivel de referencia no tiene asociado un coeficiente (su efecto se corresponde con `(Intercept)`). Los coeficientes del resto de niveles miden el cambio que se produce en la media al cambiar desde la categoría de referencia (diferencias de efectos respecto al nivel de referencia).

Para contrastar el efecto de los factores, es preferible emplear la función `anova`:

```
modelo <- lm(lnsal ~ catlab + sexo, datos)
anova(modelo)
```

```
## Analysis of Variance Table
##
## Response: lnsal
##            Df Sum Sq Mean Sq F value    Pr(>F)
## catlab      2 46.674 23.3372  489.59 < 2.2e-16 ***
## sexo        1  5.596  5.5965  117.41 < 2.2e-16 ***
## Residuals 470 22.404  0.0477
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Notas:

- Para centrarse en las efectos de los factores, se puede emplear la función `aov` (analysis of variance; ver también `model.tables()` y `TukeyHSD()`). Esta función llama internamente a `lm()` (utilizando la misma parametrización).

- Para utilizar distintas parametrizaciones de los efectos se puede emplear el argumento `contrasts = c("contr.treatment", "contr.poly")` (ver `help(contrasts)`).

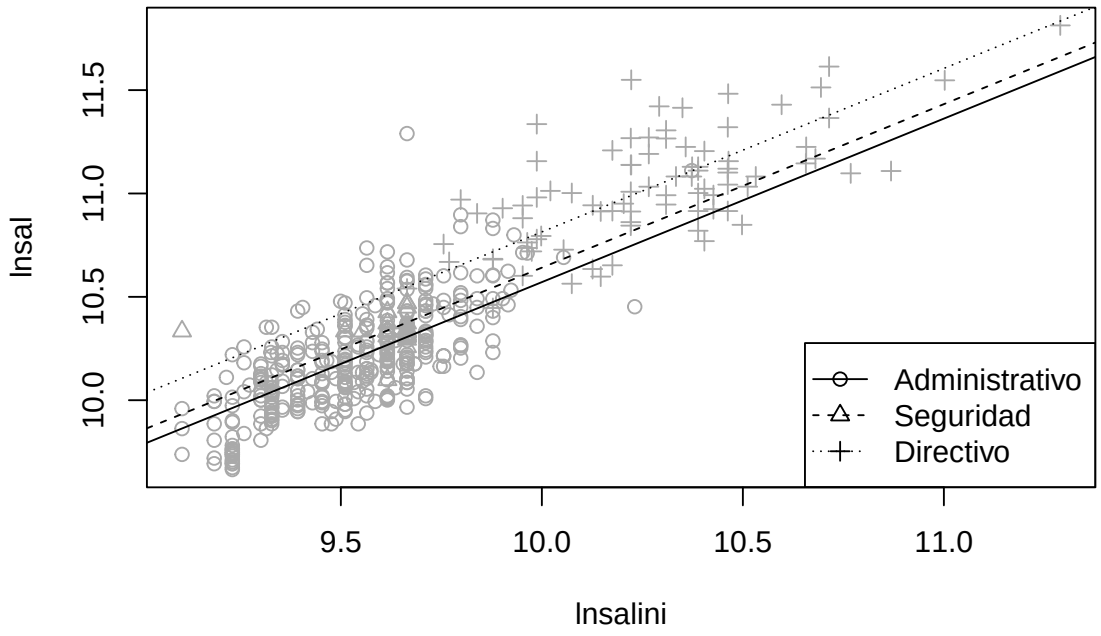
8.6 Interacciones

Al emplear el operador `+` se considera que los efectos de las covariables son aditivos (independientes):

```
modelo <- lm(lnsal ~ lnsalini + catlab, datos)
anova(modelo)
```

```
## Analysis of Variance Table
##
## Response: lnsal
##            Df Sum Sq Mean Sq  F value    Pr(>F)
## lnsalini    1 58.668  58.668 1901.993 < 2.2e-16 ***
## catlab      2  1.509   0.755   24.465 7.808e-11 ***
## Residuals 470 14.497   0.031
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
plot(lnsal ~ lnsalini, data = datos, pch = as.numeric(catlab), col = 'darkgray')
parest <- coef(modelo)
abline(a = parest[1], b = parest[2], lty = 1)
abline(a = parest[1] + parest[3], b = parest[2], lty = 2)
abline(a = parest[1] + parest[4], b = parest[2], lty = 3)
legend("bottomright", levels(datos$catlab), pch = 1:3, lty = 1:3)
```



Para especificar que el efecto de una covariable depende de otra (interacción), se pueden emplear los operadores * ó :.

```
modelo2 <- lm(lnsal ~ lnsalini*catlab, datos)
summary(modelo2)
```

```
##
## Call:
## lm(formula = lnsal ~ lnsalini * catlab, data = datos)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.37440 -0.11335 -0.00524  0.10459  0.97018
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 1.66865    0.43820   3.808 0.000159 ***
## lnsalini                    0.89512    0.04595  19.479  < 2e-16 ***
## catlabSeguridad             8.31808    3.01827   2.756 0.006081 **
## catlabDirectivo             3.01268    0.79509   3.789 0.000171 ***
## lnsalini:catlabSeguridad -0.85864    0.31392  -2.735 0.006470 **
## lnsalini:catlabDirectivo -0.27713    0.07924  -3.497 0.000515 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1727 on 468 degrees of freedom
## Multiple R-squared:  0.8131, Adjusted R-squared:  0.8111
## F-statistic: 407.3 on 5 and 468 DF,  p-value: < 2.2e-16
```

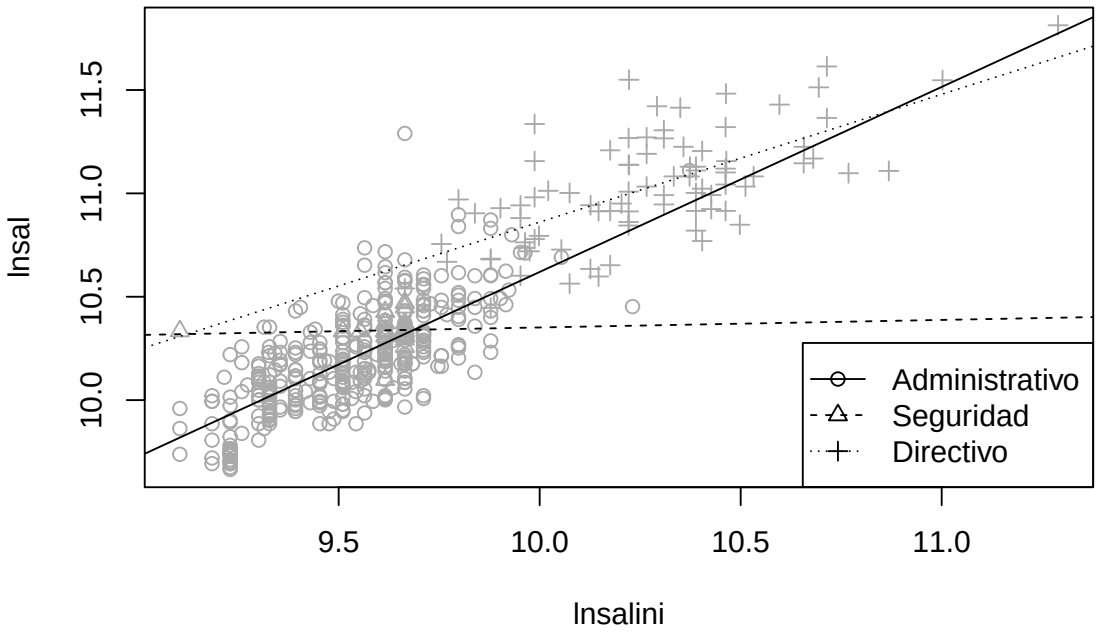
```
anova(modelo2)
```

```
## Analysis of Variance Table
```

```
##
## Response: lnsal
##                   Df Sum Sq Mean Sq   F value    Pr(>F)
## lnsalini           1 58.668  58.668 1967.6294 < 2.2e-16 ***
## catlab             2  1.509   0.755   25.3090 3.658e-11 ***
## lnsalini:catlab    2  0.543   0.272    9.1097 0.0001315 ***
## Residuals        468 13.954   0.030
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

En este caso las pendientes también varían dependiendo del nivel del factor:

```
plot(lnsal ~ lnsalini, data = datos, pch = as.numeric(catlab), col = 'darkgray')
parest <- coef(modelo2)
abline(a = parest[1], b = parest[2], lty = 1)
abline(a = parest[1] + parest[3], b = parest[2] + parest[5], lty = 2)
abline(a = parest[1] + parest[4], b = parest[2] + parest[6], lty = 3)
legend("bottomright", levels(datos$catlab), pch = 1:3, lty = 1:3)
```



Por ejemplo, empleando la fórmula `lnsal ~ lnsalini:catlab` se considerarían distintas pendientes pero el mismo término independiente.

8.7 Diagnosis del modelo

Las conclusiones obtenidas con este método se basan en las hipótesis básicas del modelo:

- Linealidad.
- Normalidad (y homogeneidad).
- Homocedasticidad.
- Independencia.

- Ninguna de las variables explicativas es combinación lineal de las demás.

Si alguna de estas hipótesis no es cierta, las conclusiones obtenidas pueden no ser fiables, o incluso totalmente erróneas. En el caso de regresión múltiple es de especial interés el fenómeno de la multicolinealidad (o colinearidad) relacionado con la última de estas hipótesis.

En esta sección consideraremos como ejemplo el modelo:

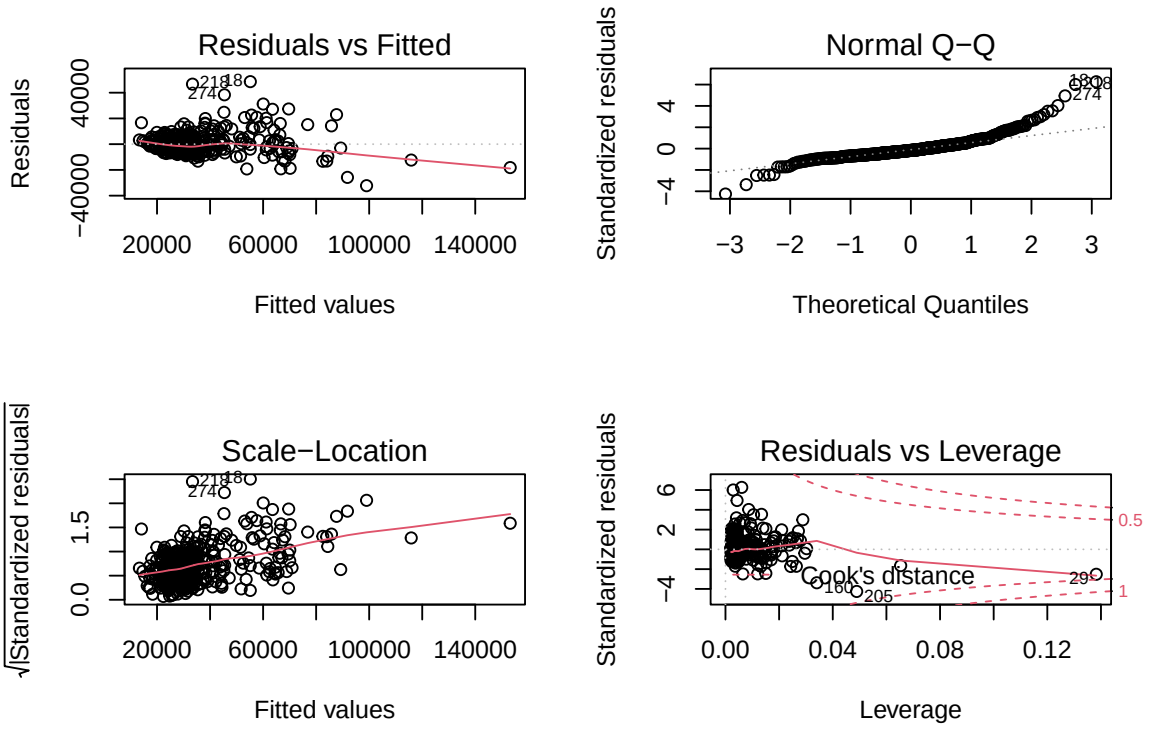
```
modelo <- lm(salario ~ salini + expprev, data = empleados)
summary(modelo)
```

```
##
## Call:
## lm(formula = salario ~ salini + expprev, data = empleados)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -32263  -4219  -1332   2673  48571
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3850.71760  900.63287   4.276 2.31e-05 ***
## salini         1.92291    0.04548  42.283  < 2e-16 ***
## expprev      -22.44482    3.42240  -6.558 1.44e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7777 on 471 degrees of freedom
## Multiple R-squared:  0.7935, Adjusted R-squared:  0.7926
## F-statistic: 904.8 on 2 and 471 DF,  p-value: < 2.2e-16
```

8.7.1 Gráficas básicas de diagnóstico

Con la función `plot` se pueden generar gráficos de interés para la diagnosis del modelo:

```
oldpar <- par( mfrow=c(2,2))
plot(modelo)
```




```
par(oldpar)
```

Por defecto se muestran cuatro gráficos (ver `help(plot.lm)` para más detalles). El primero (residuos frente a predicciones) permite detectar falta de linealidad o heterocedasticidad (o el efecto de un factor omitido: mala especificación del modelo), lo ideal sería no observar ningún patrón.

El segundo gráfico (gráfico QQ), permite diagnosticar la normalidad, los puntos del deberían estar cerca de la diagonal.

El tercer gráfico de dispersión-nivel permite detectar heterocedasticidad y ayudar a seleccionar una transformación para corregirla (más adelante, en la sección *Alternativas*, se tratará este tema), la pendiente de los datos debería ser nula.

El último gráfico permite detectar valores atípicos o influyentes. Representa los residuos estandarizados en función del valor de influencia (a priori) o leverage (hii que depende de los valores de las variables explicativas, debería ser $< 2(p+1)/2$) y señala las observaciones atípicas (residuos fuera de [-2,2]) e influyentes a posteriori (estadístico de Cook >0.5 y >1).

Si las conclusiones obtenidas dependen en gran medida de una observación (normalmente atípica), esta se denomina influyente (a posteriori) y debe ser examinada con cuidado por el experimentador. Para recalcular el modelo sin una de las observaciones puede ser útil la función update:

```
# which.max(cooks.distance(modelo))
modelo2 <- update(modelo, data = empleados[-29, ])
```

Si hay datos atípicos o influyentes, puede ser recomendable emplear regresión lineal robusta, por ejemplo mediante la función `rlm` del paquete `MASS`.

En el ejemplo anterior, se observa claramente heterogeneidad de varianzas y falta de normalidad. Aparentemente no hay observaciones influyentes (a posteriori) aunque si algún dato atípico.

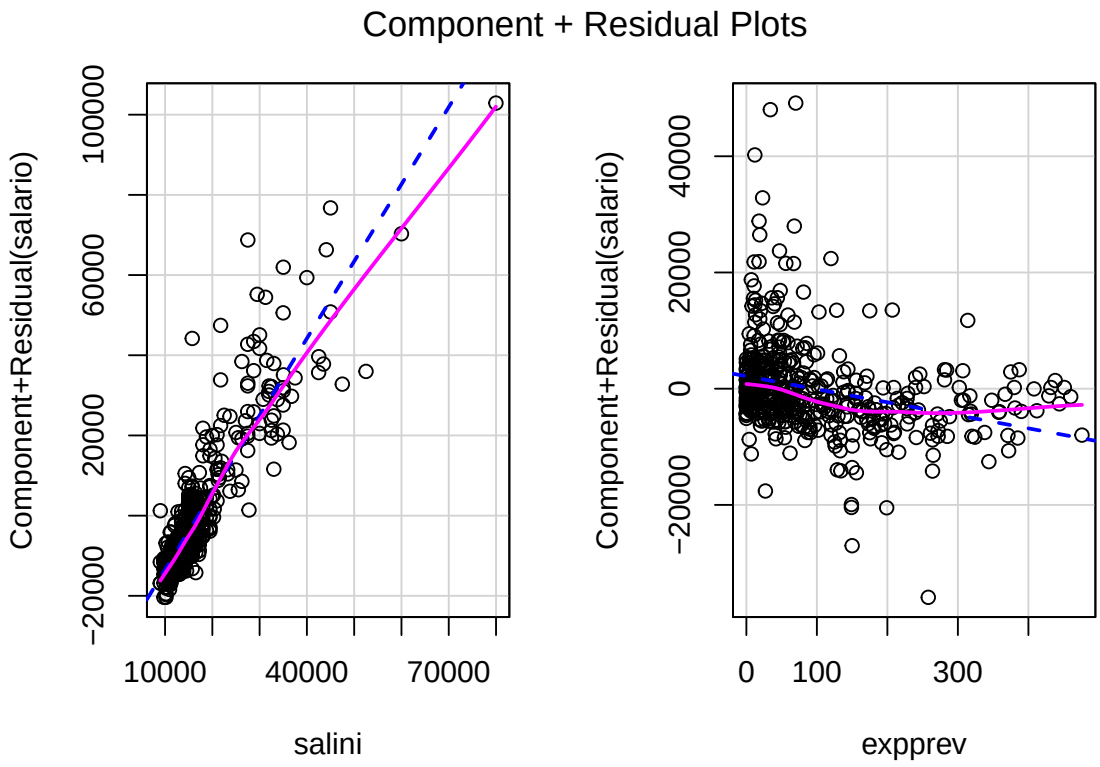
8.7.2 Gráficos parciales de residuos

En regresión lineal múltiple, en lugar de generar gráficos de dispersión simple (p.e. gráficos de dispersión matriciales) para detectar problemas (falta de linealidad, …) y analizar los efectos de las variables explicativas, se pueden generar gráficos parciales de residuos, por ejemplo con el comando:

```
termplot(modelo, partial.resid = TRUE)
```

Aunque puede ser preferible emplear las funciones `crPlots` ó `avPlots` del paquete `car`:

```
library(car)
crPlots(modelo)
```




```
# avPlots(modelo)
```

Estas funciones permitirían además detectar puntos atípicos o influyentes (mediante los argumentos `id.method` e `id.n`).

8.7.3 Estadísticos

Para obtener medidas de diagnosis o resúmenes numéricos de interés se pueden emplear las siguientes funciones:

Función	Descripción
rstandard	residuos estandarizados
rstudent	residuos estudentizados (eliminados)
cooks.distance	valores del estadístico de Cook
influence	valores de influencia, cambios en coeficientes y varianza residual al eliminar cada dato.

Ejecutar `help(influence.measures)` para ver un listado de medidas de diagnóstico adicionales.

Hay muchas herramientas adicionales disponibles en otros paquetes. Por ejemplo, para la detección de multicolinealidad, se puede emplear la función `vif` del paquete `car` para calcular los factores de inflación de varianza para las variables del modelo:

```
# library(car)
vif(modelo)
```

```
##   salini  expprev
## 1.002041 1.002041
```

Valores grandes, por ejemplo > 10, indican la posible presencia de multicolinealidad.

Nota: Las tolerancias (proporciones de variabilidad no explicada por las demás covariables) se pueden calcular con `1/vif(modelo)`.

8.7.4 Contrastes

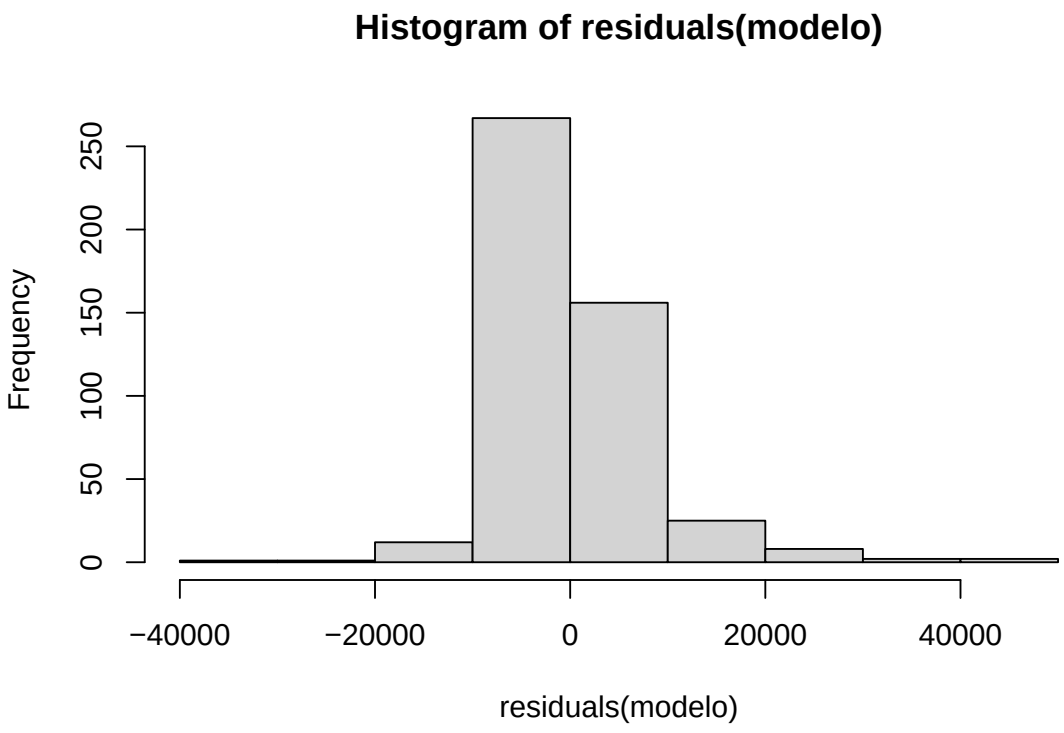
8.7.4.1 Normalidad

Para realizar el contraste de normalidad de Shapiro-Wilk se puede emplear:

```
shapiro.test(residuals(modelo))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(modelo)
## W = 0.85533, p-value < 2.2e-16
```

```
hist(residuals(modelo))
```



8.7.4.2 Homocedasticidad

La librería `lmtest` proporciona herramientas adicionales para la diagnosis de modelos lineales, por ejemplo el test de Breusch-Pagan para heterocedasticidad:

```
library(lmtest)
bptest(modelo, studentize = FALSE)
```

```
##
##  Breusch-Pagan test
##
## data:  modelo
## BP = 290.37, df = 2, p-value < 2.2e-16
```

Si el p-valor es grande aceptaríamos que hay igualdad de varianzas.

8.7.4.3 Autocorrelación

Contraste de Durbin-Watson para detectar si hay correlación serial entre los errores:

```
dwtest(modelo, alternative= "two.sided")
```

```
##
##  Durbin-Watson test
##
## data:  modelo
## DW = 1.8331, p-value = 0.06702
## alternative hypothesis: true autocorrelation is not 0
```

Si el p-valor es pequeño rechazaríamos la hipótesis de independencia.

8.8 Métodos de regularización

[[Pasar a selección de variables explicativas?]]

Estos métodos emplean también un modelo lineal:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \varepsilon$$

En lugar de ajustarlo por mínimos cuadrados (estándar), minimizando:

$$RSS = \sum_{i=1}^{n} \left(y_i - \beta_0 - \beta_1 x_{1i} - \cdots - \beta_p x_{pi}\right)^2$$

Se imponen restricciones adicionales a los parámetros que los "retraen" (shrink) hacia cero:

- Produce una reducción en la varianza de predicción (a costa del sesgo).
- En principio se consideran todas las variables explicativas.

Ridge regression

- Penalización cuadrática: $RSS + \lambda \sum_{j=1}^{p} \beta_j^2$.

Lasso

- Penalización en valor absoluto: $RSS + \lambda \sum_{j=1}^{p} |\beta_j|$.
- Normalmente asigna peso nulo a algunas variables (selección de variables).

El parámetro de penalización se selecciona por **validación cruzada**.

- Normalmente estandarizan las variables explicativas (coeficientes en la misma escala).

8.8.1 Datos

El fichero *hatco.RData* contiene observaciones de clientes de la compañía de distribución industrial (Compañía Hair, Anderson y Tatham). Las variables se pueden clasificar en tres grupos:

```
load('datos/hatco.RData')
as.data.frame(attr(hatco, "variable.labels"))
```

```
##          attr(hatco, "variable.labels")
## empresa                         Empresa
## tamano              Tamaño de la empresa
## adquisic       Estructura de adquisición
## tindustr               Tipo de industria
## tsitcomp    Tipo de situación de compra
## velocida            Velocidad de entrega
## precio                  Nivel de precios
## flexprec        Flexibilidad de precios
## imgfabri          Imagen del fabricante
## servconj              Servicio conjunto
## imgfvent     Imagen de fuerza de ventas
## calidadp             Calidad de producto
## fidelida   Porcentaje de compra a HATCO
## satisfac            Satisfacción global
## nfidelid        Nivel de compra a HATCO
## nsatisfa          Nivel de satisfacción
```

Consideraremos como respuesta la variable *fidelida* y como variables explicativas el resto de variables continuas menos *satisfac*.

```
library(glmnet)
```

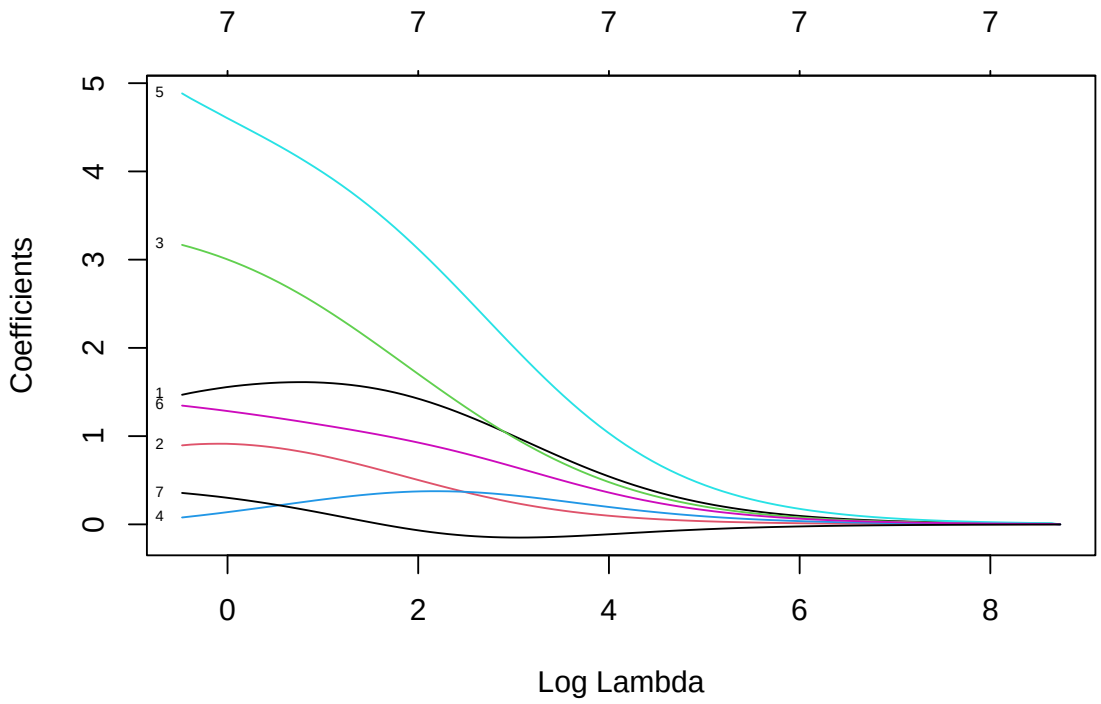
El paquete `glmnet` no emplea formulación de modelos, hay que establecer la respuesta `y` y las variables explicativas `x` (se puede emplear la función `model.matrix()` para construir `x`, la matriz de diseño, a partir de una fórmula). En este caso, eliminamos también la última fila por tener datos faltantes:

```
x <- as.matrix(hatco[-100, 6:12])
y <- hatco$fidelida[-100]
```

8.8.2 Ridge Regression

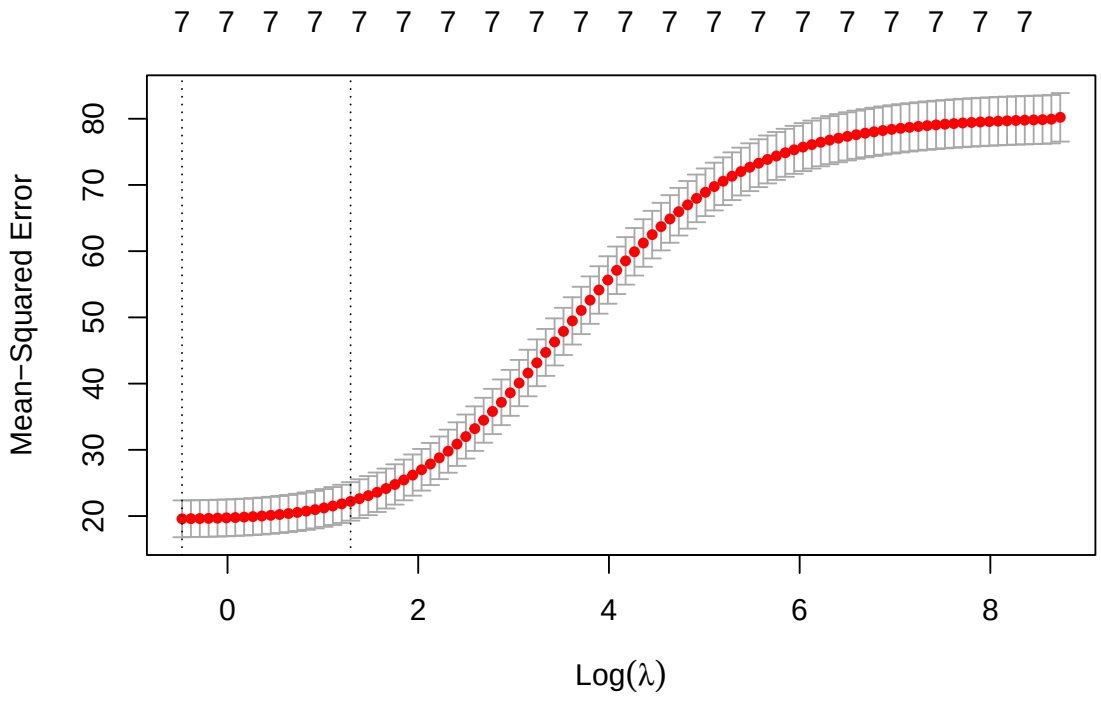
Ajustamos un modelo de regresión ridge con la función `glmnet` con `alpha=0` (ridge penalty).

```
fit.ridge <- glmnet(x, y, alpha = 0)
plot(fit.ridge, xvar = "lambda", label = TRUE)
```



Para seleccionar el parámetro de penalización por validación cruzada se puede emplear la función `cv.glmnet`.

```
cv.ridge <- cv.glmnet(x, y, alpha = 0)
plot(cv.ridge)
```



En este caso el parámetro sería:

```
cv.ridge$lambda.1se
```

```
## [1] 3.635163
```

y el modelo resultante contiene todas las variables explicativas:

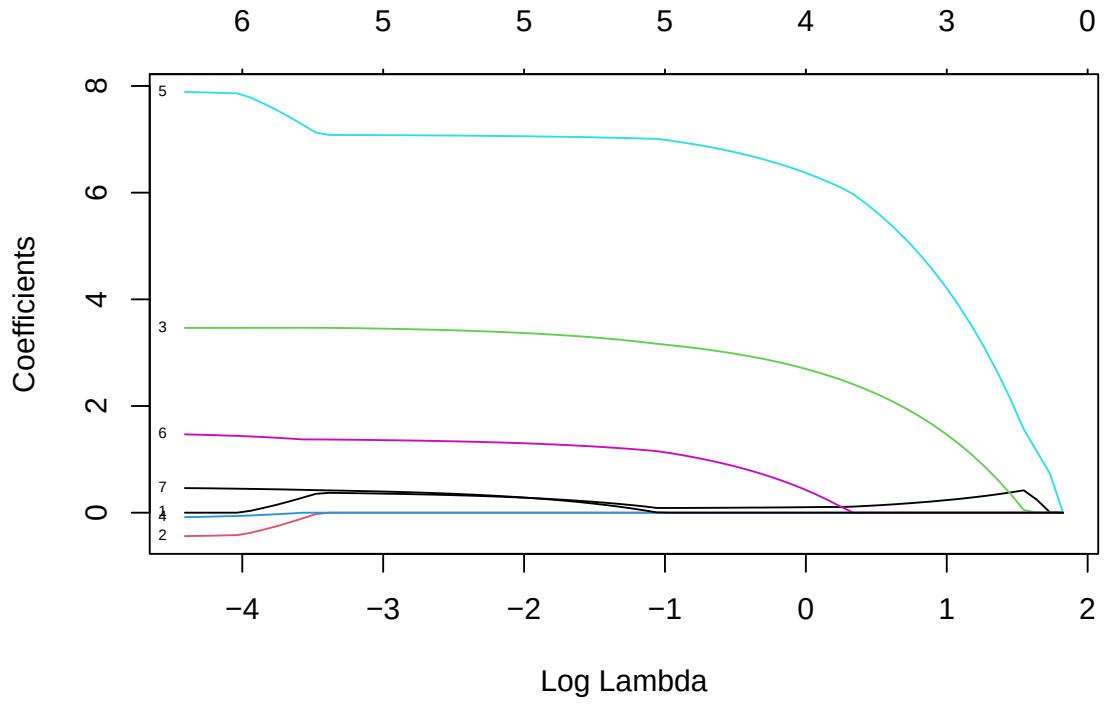
```
coef(cv.ridge)
```

```
## 8 x 1 sparse Matrix of class "dgCMatrix"
##                      s1
## (Intercept) 5.26333438
## velocida    1.58051175
## precio      0.70395775
## flexprec    2.24798481
## imgfabri    0.31897738
## servconj    3.76988236
## imgfvent    1.07304993
## calidadp    0.06641356
```

8.8.3 Lasso

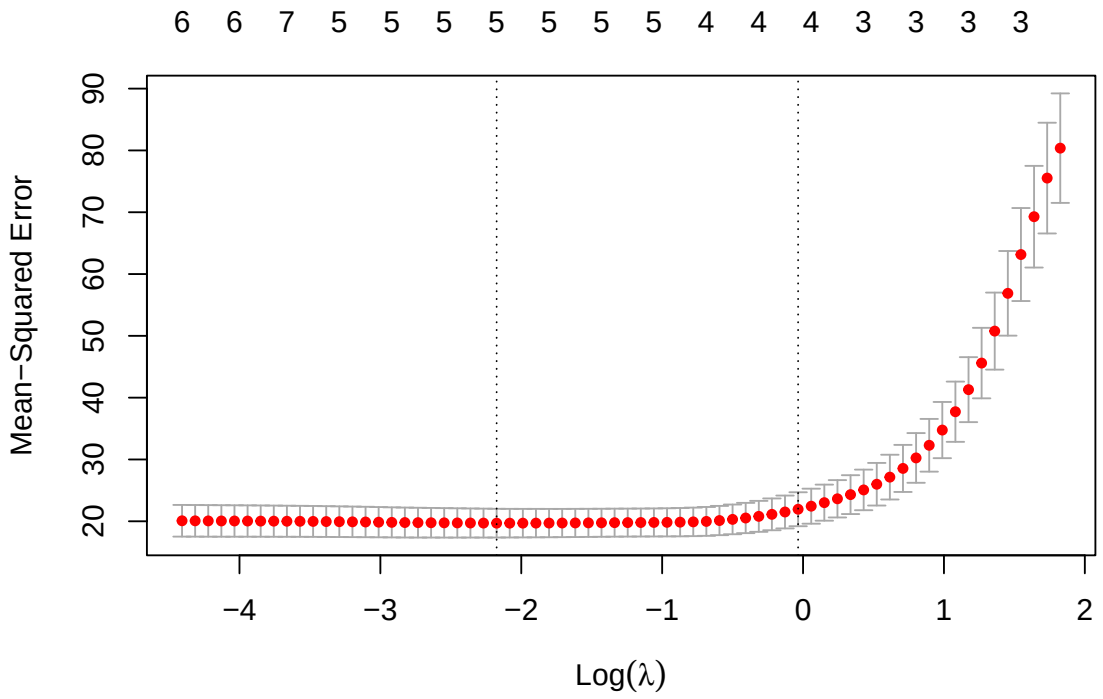
Ajustamos un modelo lasso también con la función `glmnet` (con la opción por defecto `alpha=1`, lasso penalty).

```
fit.lasso <- glmnet(x,y)
plot(fit.lasso, xvar = "lambda", label = TRUE)
```



Seleccionamos el parámetro de penalización por validación cruzada.

```
cv.lasso <- cv.glmnet(x,y)
plot(cv.lasso)
```



En este caso el modelo resultante solo contiene 4 variables explicativas:

```
coef(cv.lasso)
```

```
## 8 x 1 sparse Matrix of class "dgCMatrix"
##                     s1
## (Intercept) 4.4757712
## velocida    0.1020531
## precio      .
## flexprec    2.7202485
## imgfabri    .
## servconj    6.4044378
## imgfvent    0.4651076
## calidadp    .
```

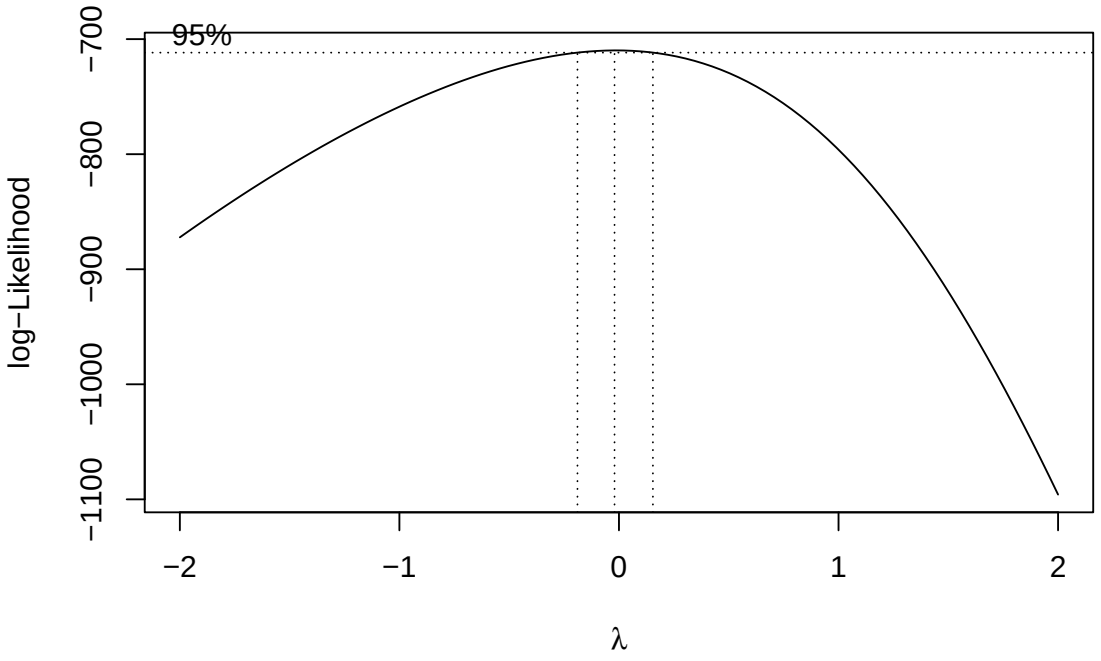
8.9 Alternativas

8.9.1 Transformación (modelos linealizables)

Cuando no se satisfacen los supuestos básicos se puede intentar transformar los datos para corregir la falta de linealidad, la heterocedasticidad y/o la falta de normalidad (normalmente estas últimas "suelen ocurrir en la misma escala"). Por ejemplo, la función `boxcox` del paquete `MASS` permite seleccionar la transformación de Box-Cox más adecuada:

$$Y^{(\lambda)} = \begin{cases} \dfrac{Y^{\lambda}-1}{\lambda} & \text{si } \lambda \neq 0 \\ \ln(Y) & \text{si } \lambda = 0 \end{cases}$$

```
# library(MASS)
boxcox(modelo)
```



En este caso una transformación logarítmica parece adecuada.

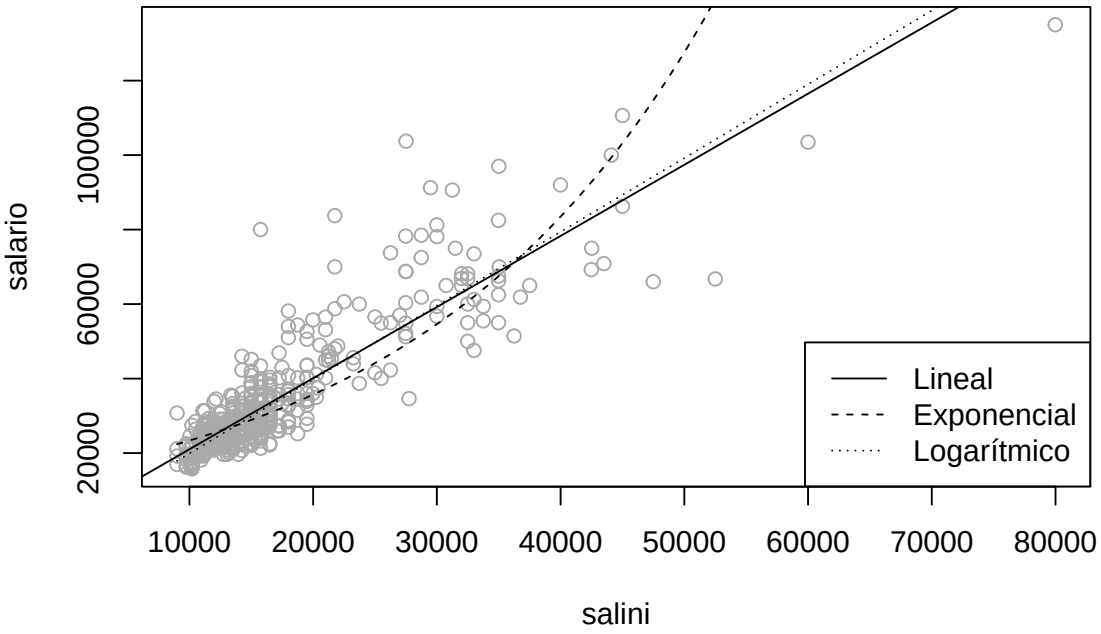
En ocasiones para obtener una relación lineal (o heterocedasticidad) también es necesario transformar las covariables además de la respuesta. Algunas de las relaciones fácilmente linealizables se muestran a continuación:

modelo	ecuación	covariable	respuesta
logarítmico	$y = a + b\ log(x)$	$log(x)$	_
inverso	$y = a + b/x$	$1/x$	_
potencial	$y = ax^b$	$log(x)$	$log(y)$
exponencial	$y = ae^{bx}$	_	$log(y)$
curva-S	$y = ae^{b/x}$	$1/x$	$log(y)$

8.9.1.1 Ejemplo:

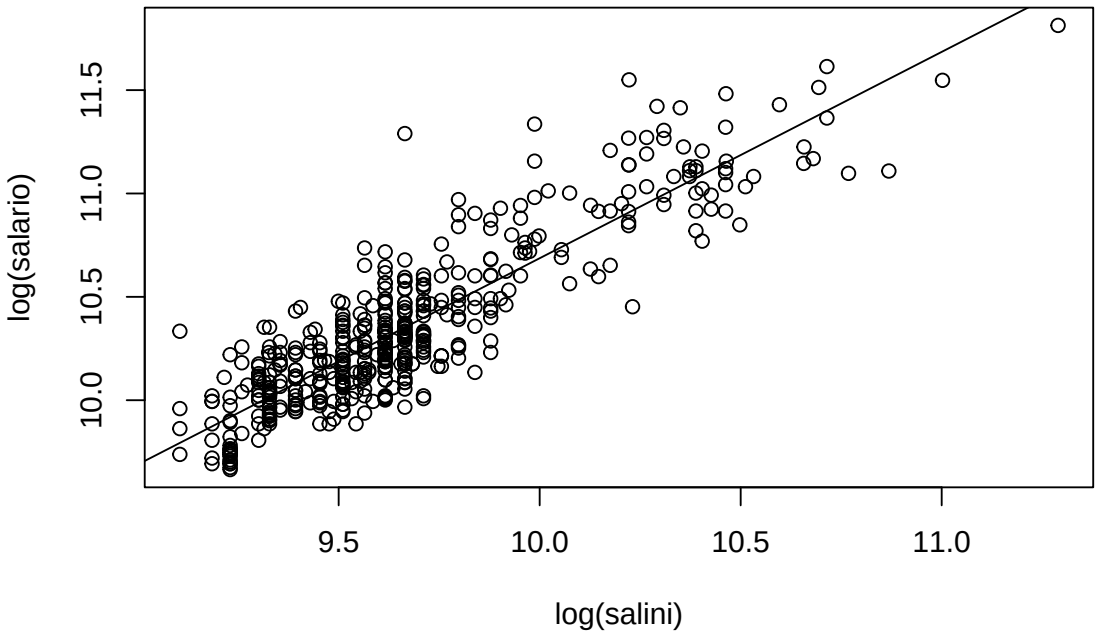
```
plot(salario ~ salini, data = empleados, col = 'darkgray')

# Ajuste lineal
abline(lm(salario ~ salini, data = empleados))

# Modelo exponencial
modelo1 <- lm(log(salario) ~ salini, data = empleados)
parest <- coef(modelo1)
curve(exp(parest[1] + parest[2]*x), lty = 2, add = TRUE)

# Modelo logarítmico
modelo2 <- lm(log(salario) ~ log(salini), data = empleados)
parest <- coef(modelo2)
curve(exp(parest[1]) * x^parest[2], lty = 3, add = TRUE)
```

```
legend("bottomright", c("Lineal","Exponencial","Logarítmico"), lty = 1:3)
```



Con estos datos de ejemplo, el principal problema es la falta de homogeneidad de varianzas (y de normalidad) y se corrige sustancialmente con el segundo modelo:

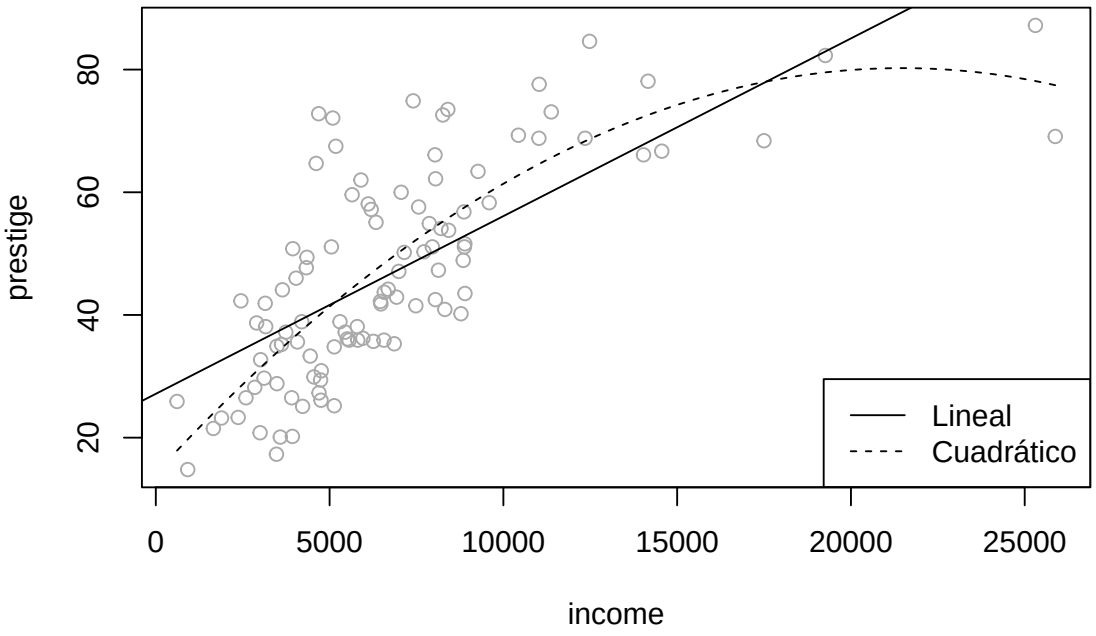
```
plot(log(salario) ~ log(salini), data = empleados)
abline(modelo2)
```



8.9.2 Ajuste polinómico

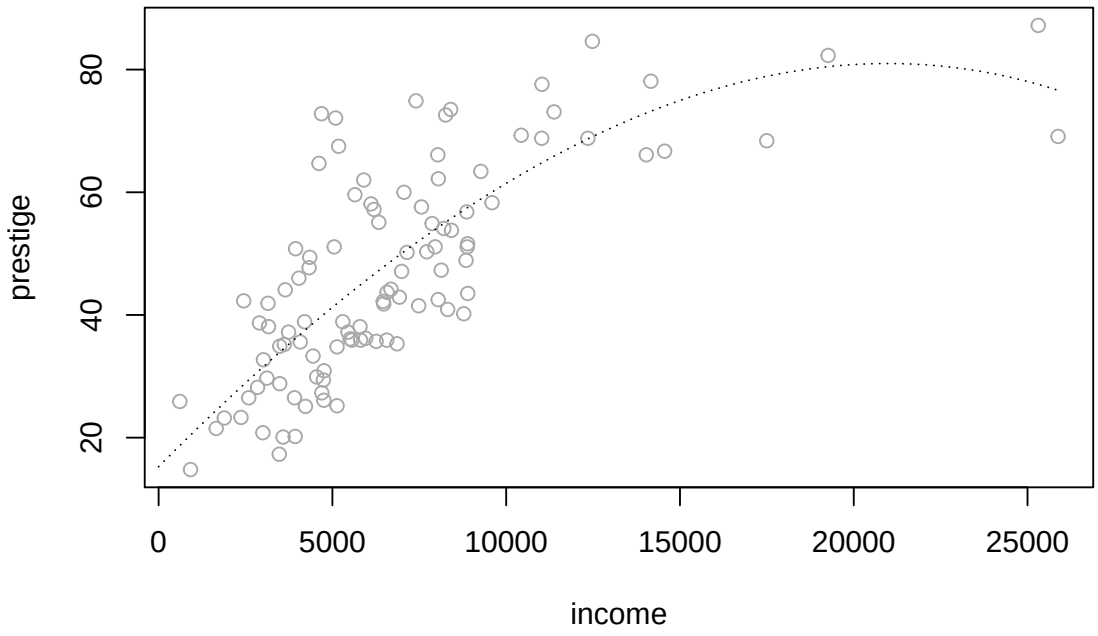
En este apartado utilizaremos como ejemplo el conjunto de datos `Prestige` de la librería `car`. Al tratar de explicar `prestige` (puntuación de ocupaciones obtenidas a partir de una encuesta) a partir de `income` (media de ingresos en la ocupación), un ajuste cuadrático puede parecer razonable:

```
# library(car)
plot(prestige ~ income, data = Prestige, col = 'darkgray')
# Ajuste lineal
abline(lm(prestige ~ income, data = Prestige))
# Ajuste cuadrático
modelo <- lm(prestige ~ income + I(income^2), data = Prestige)
parest <- coef(modelo)
curve(parest[1] + parest[2]*x + parest[3]*x^2, lty = 2, add = TRUE)

legend("bottomright", c("Lineal","Cuadrático"), lty = 1:2)
```



Alternativamente se podría emplear la función `poly`:

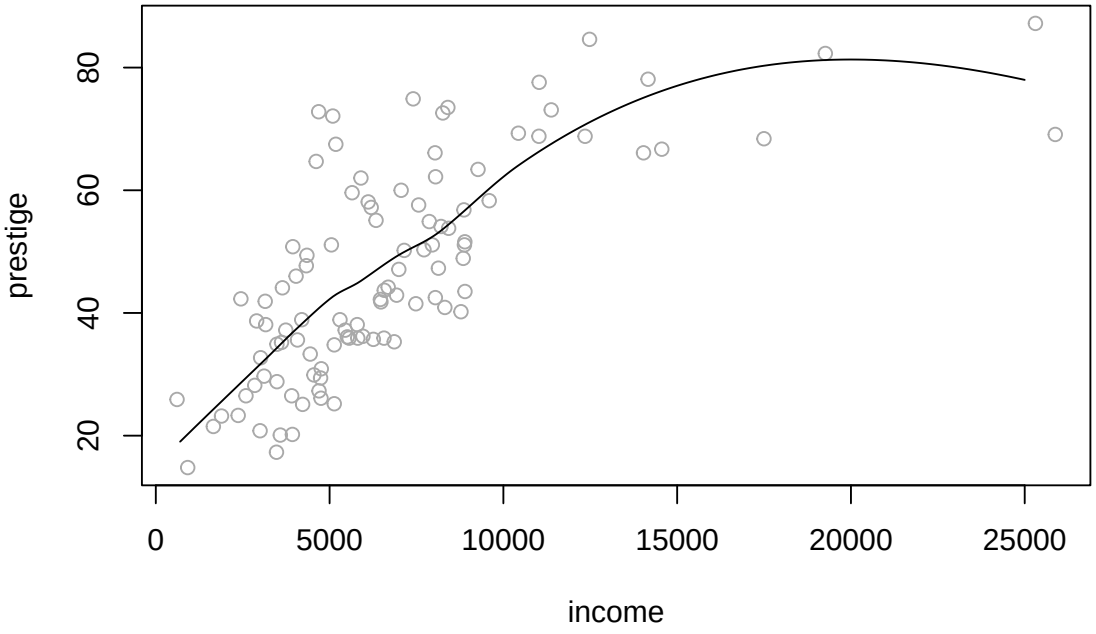
```
plot(prestige ~ income, data = Prestige, col = 'darkgray')
# Ajuste cúbico
modelo <- lm(prestige ~ poly(income, 3), data = Prestige)
valores <- seq(0, 26000, len = 100)
pred <- predict(modelo, newdata = data.frame(income = valores))
lines(valores, pred, lty = 3)
```



8.9.3 Ajuste polinómico local (robusto)

Si no se logra un buen ajuste empleando los modelos anteriores se puede pensar en utilizar métodos no paramétricos (p.e. regresión aditiva no paramétrica). Por ejemplo, en`R` es habitual emplear la función `loess` (sobre todo en gráficos):

```
plot(prestige ~ income, Prestige, col = 'darkgray')
fit <- loess(prestige ~ income, Prestige, span = 0.75)
valores <- seq(0, 25000, 100)
pred <- predict(fit, newdata = data.frame(income = valores))
lines(valores, pred)
```



Este tipo de modelos los trataremos con detalle más adelante…

Capítulo 9

Modelos lineales generalizados

Los modelos lineales generalizados son una extensión de los modelos lineales para el caso de que la distribución condicional de la variable respuesta no sea normal (por ejemplo discreta: Bernouilli, Binomial, Poisson, …)

En los modelo lineales se supone que:

$$E(Y|\mathbf{X}) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p$$

En los modelos lineales generalizados se introduce una función invertible g, denominada función enlace (o link):

$$g\left(E(Y|\mathbf{X})\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p$$

9.1 Ajuste: función `glm`

Para el ajuste (estimación de los parámetros) de un modelo lineal generalizado a un conjunto de datos (por máxima verosimilitud) se emplea la función `glm`:

```
ajuste <- glm(formula, family = gaussian, datos, ...)
```

El parámetro `family` indica la distribución y el link. Por ejemplo:

- `gaussian(link = "identity")`, `gaussian(link = "log")`
- `binomial(link = "logit")`, `binomial(link = "probit")`
- `poisson(link = "log")`
- `Gamma(link = "inverse")`

Para cada distribución se toma por defecto una función link (mostrada en primer lugar; ver `help(family)` para más detalles).

Muchas de las herramientas y funciones genéricas disponibles para los modelos lineales son válidas también para este tipo de modelos: `summary`, `coef`, `confint`, `predict`, `anova`, ….

Veremos con más detalle el caso particular de la regresión logística.

9.2 Regresión logística

9.2.1 Ejemplo

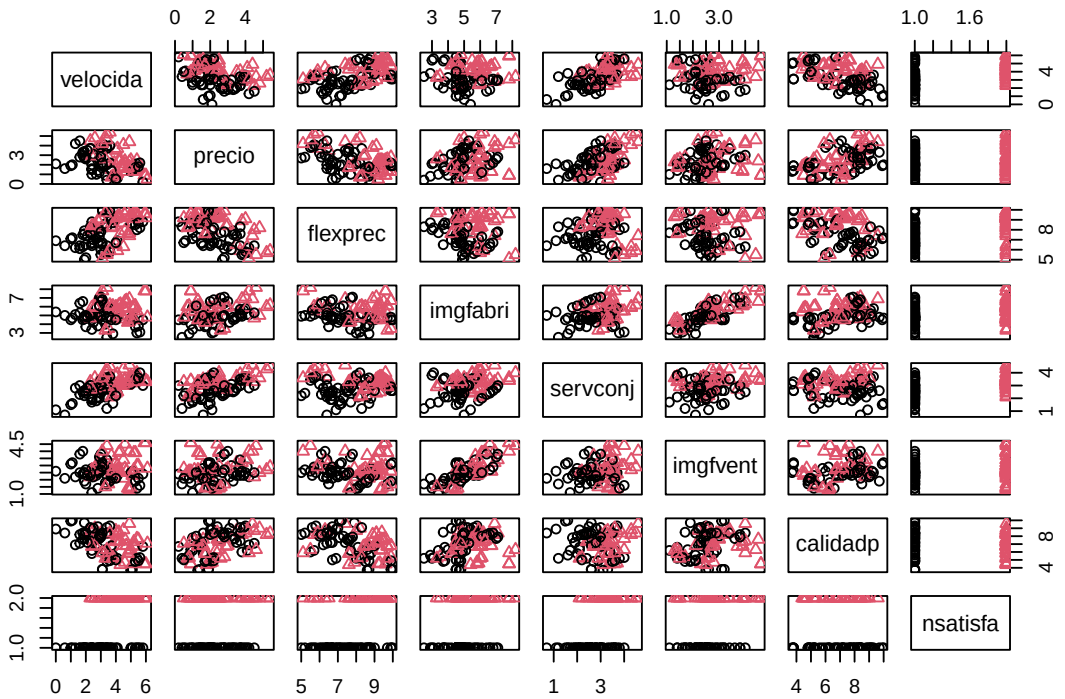
Como ejemplo emplearemos los datos de clientes de la compañía de distribución industrial (Compañía Hair, Anderson y Tatham).

```
load("datos/hatco.RData")
as.data.frame(attr(hatco, "variable.labels"))
```

```
##          attr(hatco, "variable.labels")
## empresa                         Empresa
## tamano              Tamaño de la empresa
## adquisic       Estructura de adquisición
## tindustr               Tipo de industria
## tsitcomp    Tipo de situación de compra
## velocida            Velocidad de entrega
## precio                   Nivel de precios
## flexprec         Flexibilidad de precios
## imgfabri           Imagen del fabricante
## servconj               Servicio conjunto
## imgfvent      Imagen de fuerza de ventas
## calidadp              Calidad de producto
## fidelida   Porcentaje de compra a HATCO
## satisfac              Satisfacción global
## nfidelid         Nivel de compra a HATCO
## nsatisfa           Nivel de satisfacción
```

Consideraremos como respuesta la variable *nsatisfa* y como variables explicativas el resto de variables continuas menos *fidelida* y *satisfac*. Eliminamos también la última fila por tener datos faltantes (realmente no sería necesario).

```
datos <- hatco[-100, c(6:12, 16)]
plot(datos, pch = as.numeric(datos$nsatisfa), col = as.numeric(datos$nsatisfa))
```



9.2.2 Ajuste de un modelo de regresión logística

Se emplea la función `glm` seleccionando `family = binomial` (la función de enlace por defecto será *logit*):

```
modelo <- glm(nsatisfa ~ velocida + imgfabri , family = binomial, data = datos)
modelo
```

```
##
## Call:  glm(formula = nsatisfa ~ velocida + imgfabri, family = binomial,
##     data = datos)
##
## Coefficients:
## (Intercept)     velocida     imgfabri
##     -10.127        1.203        1.058
##
## Degrees of Freedom: 98 Total (i.e. Null);  96 Residual
## Null Deviance:       136.4
## Residual Deviance: 88.64     AIC: 94.64
```

La razón de ventajas (OR) permite cuantificar el efecto de las variables explicativas en la respuesta (Incremento proporcional en la ventaja o probabilidad de éxito, al aumentar una unidad la variable manteniendo las demás fijas):

```
exp(coef(modelo))  # Razones de ventajas ("odds ratios")
```

```
##  (Intercept)     velocida     imgfabri
## 3.997092e-05 3.329631e+00 2.881619e+00
```

```
exp(confint(modelo))
```

```
## Waiting for profiling to be done...
```

```
##                    2.5 %      97.5 %
## (Intercept) 3.828431e-07 0.001621259
## velocida    2.061302e+00 5.976208357
## imgfabri    1.737500e+00 5.247303813
```

Para obtener un resumen más completo del ajuste también se utiliza `summary()`

```
summary(modelo)
```

```
##
## Call:
## glm(formula = nsatisfa ~ velocida + imgfabri, family = binomial,
##     data = datos)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8941  -0.6697  -0.2098   0.7865   2.3378
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -10.1274     2.1062  -4.808 1.52e-06 ***
## velocida      1.2029     0.2685   4.479 7.49e-06 ***
## imgfabri      1.0584     0.2792   3.790 0.000151 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 136.42  on 98  degrees of freedom
```

```
## Residual deviance:  88.64  on 96  degrees of freedom
## AIC: 94.64
##
## Number of Fisher Scoring iterations: 5
```

La desvianza (deviance) es una medida de la bondad del ajuste de un modelo lineal generalizado (sería equivalente a la suma de cuadrados residual de un modelo lineal; valores más altos indican peor ajuste). La *Null deviance* se correspondería con un modelo solo con la constante y la *Residual deviance* con el modelo ajustado. En este caso hay una reducción de 47.78 con una pérdida de 2 grados de libertad (una reducción significativa).

Para contrastar globalmente el efecto de las covariables también podemos emplear:

```
modelo.null <- glm(nsatisfa ~ 1, binomial, datos)
anova(modelo.null, modelo, test = "Chi")
```

```
## Analysis of Deviance Table
##
## Model 1: nsatisfa ~ 1
## Model 2: nsatisfa ~ velocida + imgfabri
##   Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
## 1        98     136.42
## 2        96      88.64  2   47.783 4.207e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

9.3 Predicción

Las predicciones se obtienen también con la función `predict`:

```
p.est <- predict(modelo, type = "response")
```

El parámetro `type = "response"` permite calcular las probabilidades estimadas de la segunda categoría.

Podríamos obtener una tabla de clasificación:

```
cat.est <- as.numeric(p.est > 0.5)
tabla <- table(datos$nsatisfa, cat.est)
tabla
```

```
##       cat.est
##         0  1
##   bajo 44 10
##   alto  7 38
```

```
print(100*prop.table(tabla), digits = 2)
```

```
##       cat.est
##           0    1
##   bajo 44.4 10.1
##   alto  7.1 38.4
```

Por defecto `predict` obtiene las predicciones correspondientes a las observaciones (`modelo$fitted.values`). Para otros casos hay que emplear el argumento `newdata`.

9.4 Selección de variables explicativas

El objetivo sería conseguir un buen ajuste con el menor número de variables explicativas posible.

Para actualizar un modelo (p.e. eliminando o añadiendo variables) se puede emplear la función `update`:

```
modelo.completo <- glm(nsatisfa ~ . , family = binomial, data = datos)
summary(modelo.completo)
```

```
##
## Call:
## glm(formula = nsatisfa ~ ., family = binomial, data = datos)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.01370  -0.31260  -0.02826   0.35423   1.74741
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -32.6317     7.7121  -4.231 2.32e-05 ***
## velocida      3.9980     2.3362   1.711 0.087019 .
## precio        3.6042     2.3184   1.555 0.120044
## flexprec      1.5769     0.4433   3.557 0.000375 ***
## imgfabri      2.1669     0.6857   3.160 0.001576 **
## servconj     -4.2655     4.3526  -0.980 0.327096
## imgfvent     -1.1496     0.8937  -1.286 0.198318
## calidadp      0.1506     0.2495   0.604 0.546147
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 136.424  on 98  degrees of freedom
## Residual deviance:  60.807  on 91  degrees of freedom
## AIC: 76.807
##
## Number of Fisher Scoring iterations: 7
```

```
modelo.reducido <- update(modelo.completo, . ~ . - calidadp)
summary(modelo.reducido)
```

```
##
## Call:
## glm(formula = nsatisfa ~ velocida + precio + flexprec + imgfabri +
##     servconj + imgfvent, family = binomial, data = datos)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0920  -0.3518  -0.0280   0.3876   1.7885
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -31.6022     7.3962  -4.273 1.93e-05 ***
## velocida      4.1831     2.2077   1.895 0.058121 .
## precio        3.8872     2.1685   1.793 0.073044 .
## flexprec      1.5452     0.4361   3.543 0.000396 ***
## imgfabri      2.1984     0.6746   3.259 0.001119 **
## servconj     -4.6985     4.0597  -1.157 0.247125
## imgfvent     -1.1387     0.8784  -1.296 0.194849
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
```

```
##
##     Null deviance: 136.424  on 98  degrees of freedom
## Residual deviance:  61.171  on 92  degrees of freedom
## AIC: 75.171
##
## Number of Fisher Scoring iterations: 7
```

Para obtener el modelo "óptimo" lo ideal sería evaluar todos los modelos posibles. En este caso no se puede emplear la función `regsubsets` del paquete `leaps` (sólo para modelos lineales), pero por ejemplo el paquete `bestglm` proporciona una herramienta equivalente (`bestglm()`).

9.4.1 Selección por pasos

La función `stepwise` del paquete `RcmdrMisc` (interfaz de `stepAIC` del paquete `MASS`) permite seleccionar el modelo por pasos según criterio AIC o BIC:

```
library(MASS)
library(RcmdrMisc)
modelo <- stepwise(modelo.completo, direction='backward/forward', criterion='BIC')
```

```
##
## Direction:  backward/forward
## Criterion:  BIC
##
## Start:  AIC=97.57
## nsatisfa ~ velocida + precio + flexprec + imgfabri + servconj +
##     imgfvent + calidadp
##
##            Df Deviance     AIC
## - calidadp  1   61.171  93.337
## - servconj  1   61.565  93.730
## - imgfvent  1   62.668  94.834
## - precio    1   62.712  94.878
## - velocida  1   63.105  95.271
## <none>          60.807  97.568
## - imgfabri  1   76.251 108.416
## - flexprec  1   82.443 114.609
##
## Step:  AIC=93.34
## nsatisfa ~ velocida + precio + flexprec + imgfabri + servconj +
##     imgfvent
##
##            Df Deviance     AIC
## - servconj  1   62.205  89.776
## - imgfvent  1   63.055  90.625
## - precio    1   63.698  91.269
## - velocida  1   63.983  91.554
## <none>          61.171  93.337
## + calidadp  1   60.807  97.568
## - imgfabri  1   77.823 105.394
## - flexprec  1   82.461 110.032
##
## Step:  AIC=89.78
## nsatisfa ~ velocida + precio + flexprec + imgfabri + imgfvent
##
##            Df Deviance     AIC
## - imgfvent  1   64.646  87.622
## <none>          62.205  89.776
```

```
## + servconj  1   61.171  93.337
## + calidadp  1   61.565  93.730
## - imgfabri  1   78.425 101.401
## - precio    1   79.699 102.675
## - flexprec  1   82.978 105.954
## - velocida  1   88.731 111.706
##
## Step:  AIC=87.62
## nsatisfa ~ velocida + precio + flexprec + imgfabri
##
##            Df Deviance     AIC
## <none>          64.646  87.622
## + imgfvent  1   62.205  89.776
## + servconj  1   63.055  90.625
## + calidadp  1   63.890  91.460
## - precio    1   80.474  98.854
## - flexprec  1   83.663 102.044
## - imgfabri  1   85.208 103.588
## - velocida  1   89.641 108.021
```

```
summary(modelo)
```

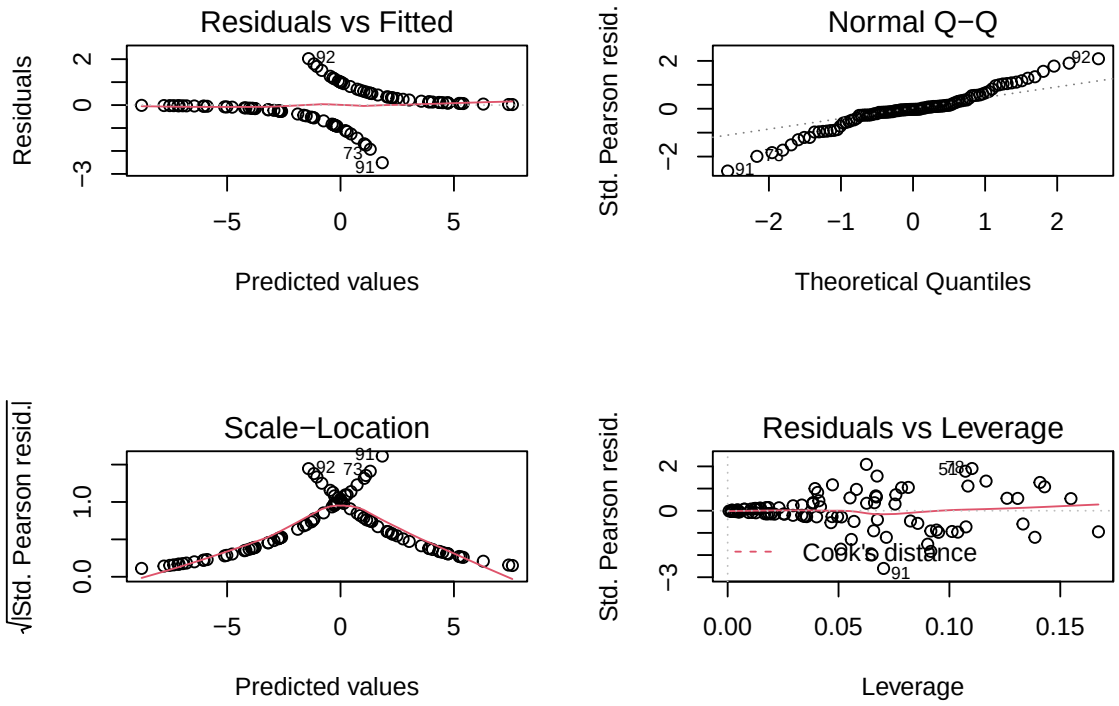
```
##
## Call:
## glm(formula = nsatisfa ~ velocida + precio + flexprec + imgfabri,
##     family = binomial, data = datos)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.99422  -0.36209  -0.03932   0.44249   1.80432
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -28.0825     6.4767  -4.336 1.45e-05 ***
## velocida      1.6268     0.4268   3.812 0.000138 ***
## precio        1.3749     0.4231   3.250 0.001155 **
## flexprec      1.3364     0.3785   3.530 0.000415 ***
## imgfabri      1.5168     0.4252   3.567 0.000361 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 136.424  on 98  degrees of freedom
## Residual deviance:  64.646  on 94  degrees of freedom
## AIC: 74.646
##
## Number of Fisher Scoring iterations: 6
```

9.5 Diagnosis del modelo

9.5.1 Gráficas básicas de diagnóstico

Con la función `plot` se pueden generar gráficos de interés para la diagnosis del modelo:

```
oldpar <- par( mfrow=c(2,2))
plot(modelo)
```



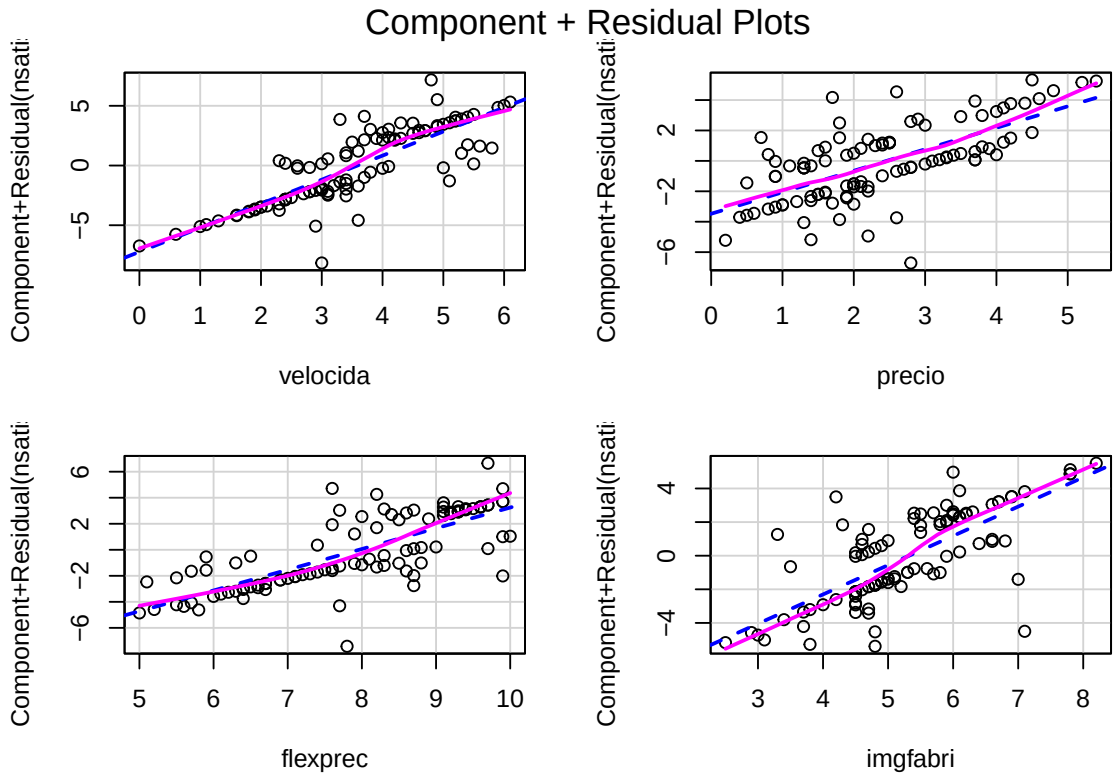

```
par(oldpar)
```

Aunque su interpretación difiere un poco de la de los modelos lineales...

9.5.2 Gráficos parciales de residuos

Se pueden generar gráficos parciales de residuos (p.e. `crPlots()` del paquete `car`):

```
# library(car)
crPlots(modelo)
```



9.5.3 Estadísticos

Se pueden emplear las mismas funciones vistas en los modelos lineales para obtener medidas de diagnosis de interés (ver `help(influence.measures)`). Por ejemplo:

```
residuals(model, type = "deviance")
```

proporciona los residuos deviance.

En general, muchas de las herramientas para modelos lineales son también válidas para estos modelos. Por ejemplo:

```
# library(car)
vif(modelo)
```

```
## velocida   precio flexprec imgfabri
## 2.088609 2.653934 2.520042 1.930409
```

9.6 Alternativas

Además de considerar ajustes polinómicos, pueden ser de interés emplear métodos no paramétricos. Por ejemplo, puede ser recomendable la función `gam` del paquete `mgcv`.

Capítulo 10

Regresión no paramétrica

No se supone ninguna forma concreta en el efecto de las variables explicativas:

$$Y = f(\mathbf{X}) + \varepsilon,$$

con f función "cualquiera" (suave).

- Métodos disponibles en `R`:
  - Regresión local (métodos de suavizado): `loess()`, `KernSmooth`, `sm`, ...
  - Modelos aditivos generalizados (GAM): `gam`, `mgcv`, ...
  - ...

10.1 Modelos aditivos

Se supone que:

$$Y = \beta_0 + f_1(\mathbf{X}_1) + f_2(\mathbf{X}_2) + \cdots + f_p(\mathbf{X}_p) + \varepsilon,$$

con f_i, $i = 1, ..., p$, funciones cualesquiera.

- Los modelos lineales son un caso particular considerando $f_i(x) = \beta_i \cdot x$.
- Adicionalmente se puede considerar una función link: **Modelos aditivos generalizados** (GAM)
  - Hastie, T.J. y Tibshirani, R.J. (1990). Generalized Additive Models. Chapman & Hall.
  - Wood, S. N. (2006). Generalized Additive Models: An Introduction with R. Chapman & Hall/CRC

10.1.1 Ajuste: función `gam`

La función `gam` del paquete `mgcv` permite ajustar modelos aditivos (generalizados) empleando regresión por splines (ver `help("mgcv-package")`):

```
library(mgcv)
ajuste <- gam(formula, family = gaussian, datos, pesos, seleccion, na.action, ...)
```

Algunas posibilidades de uso son las que siguen:

- Modelo lineal:

  ```
  ajuste <- gam(y ~ x1 + x2 + x3)
  ```

- Modelo aditivo con efectos no paramétricos para x1 y x2, y un efecto lineal para x3:

```
ajuste <- gam(y ~ s(x1) + s(x2) + x3)
```

- Modelo no aditivo (con interacción):

```
ajuste <- gam(y ~ s(x1, x2))
```

- Modelo con distintas combinaciones:

```
ajuste <- gam(y ~ s(x1, x2) + s(x3) + x4)
```

10.1.2 Ejemplo

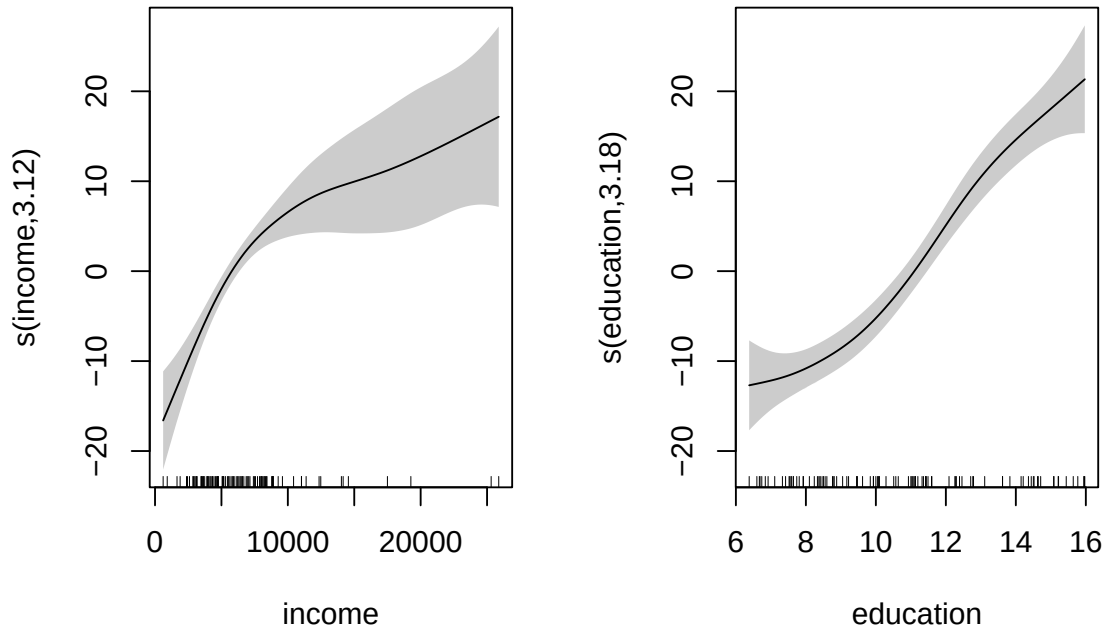
En esta sección utilizaremos como ejemplo el conjunto de datos `Prestige` de la librería `car`. Se tratará de explicar `prestige` (puntuación de ocupaciones obtenidas a partir de una encuesta) a partir de `income` (media de ingresos en la ocupación) y `education` (media de los años de educación).

```
library(mgcv)
library(car)
modelo <- gam(prestige ~ s(income) + s(education), data = Prestige)
summary(modelo)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## prestige ~ s(income) + s(education)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  46.8333     0.6889   67.98   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                edf Ref.df     F p-value
## s(income)    3.118  3.877 14.61  <2e-16 ***
## s(education) 3.177  3.952 38.78  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.836   Deviance explained = 84.7%
## GCV = 52.143  Scale est. = 48.414    n = 102
```

En este caso la función `plot` representa los efectos (parciales) estimados de cada covariable:

```
par.old <- par(mfrow = c(1, 2))
plot(modelo, shade = TRUE) #
```




```
par(par.old)
```

10.1.3 Superficie de predicción

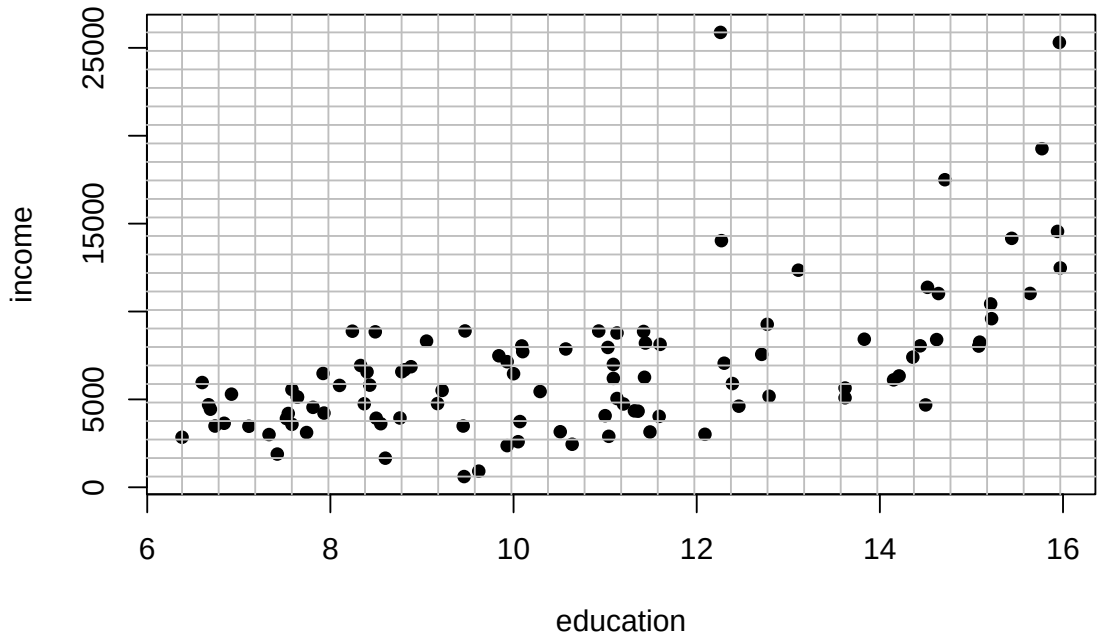
Las predicciones se obtienen también con la función `predict`:

```
pred <- predict(modelo)
```

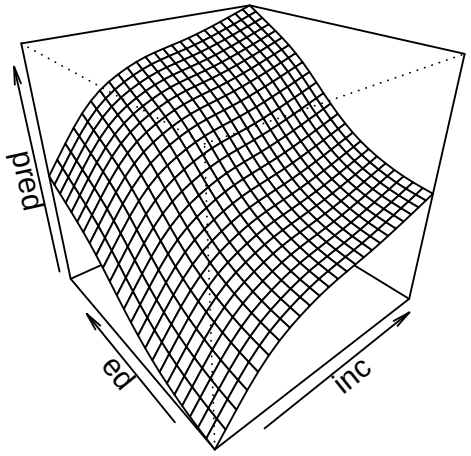
Por defecto `predict` obtiene las predicciones correspondientes a las observaciones (`modelo$fitted.values`). Para otros casos hay que emplear el argumento `newdata`.

Para representar las estimaciones (la superficie de predicción) obtenidas con el modelo se puede utilizar la función `persp`. Esta función necesita que los valores (x,y) de entrada estén dispuestos en una rejilla bidimensional. Para generar esta rejilla se puede emplear la función `expand.grid(x,y)` que crea todas las combinaciones de los puntos dados en x e y.

```
inc <- with(Prestige, seq(min(income), max(income), len = 25))
ed <- with(Prestige, seq(min(education), max(education), len = 25))
newdata <- expand.grid(income = inc, education = ed)
# Representamos la rejilla
plot(income ~ education, Prestige, pch = 16)
abline(h = inc, v = ed, col = "grey")
```

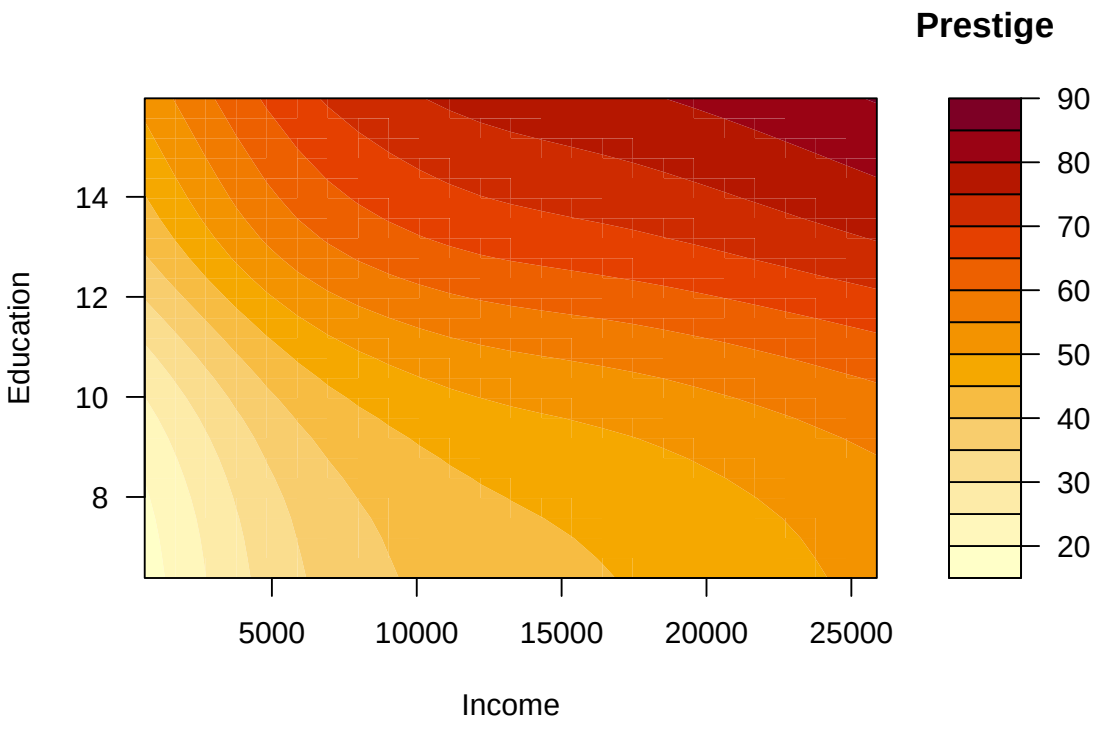



```
# Se calculan las predicciones
pred <- predict(modelo, newdata)
# Se representan
pred <- matrix(pred, nrow = 25)
persp(inc, ed, pred, theta = -40, phi = 30)
```



Alternativamente se podría emplear la función `contour` o `filled.contour`:

```
# contour(inc, ed, pred, xlab = "Income", ylab = "Education")
filled.contour(inc, ed, pred, xlab = "Income", ylab = "Education", key.title = title("Prestige"))
```



Puede ser más cómodo emplear el paquete `modelr` junto a los gráficos `ggplot2` para trabajar con modelos y predicciones.

10.1.4 Comparación de modelos

Además de las medidas de bondad de ajuste como el coeficiente de determinación ajustado, también se puede emplear la función `anova` para la comparación de modelos. Por ejemplo, viendo el gráfico de los efectos se podría pensar que el efecto de `education` podría ser lineal:

```
# plot(modelo)
modelo0 <- gam(prestige ~ s(income) + education, data = Prestige)
summary(modelo0)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## prestige ~ s(income) + education
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   4.2240     3.7323   1.132    0.261
## education     3.9681     0.3412  11.630   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##            edf Ref.df    F p-value
## s(income) 3.58  4.441 13.6  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.825   Deviance explained = 83.3%
## GCV = 54.798  Scale est. = 51.8      n = 102
```

```
anova(modelo0, modelo, test="F")
```

```
## Analysis of Deviance Table
##
## Model 1: prestige ~ s(income) + education
## Model 2: prestige ~ s(income) + s(education)
##   Resid. Df Resid. Dev     Df Deviance      F Pr(>F)
## 1    95.559     4994.6
## 2    93.171     4585.0 2.3886   409.58 3.5418 0.0257 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

En este caso aceptaríamos que el modelo original es significativamente mejor.

Alternativamente, podríamos pensar que hay interacción:

```
modelo2 <- gam(prestige ~ s(income, education), data = Prestige)
summary(modelo2)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
```

```
## prestige ~ s(income, education)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  46.8333     0.7138   65.61   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                        edf Ref.df     F p-value
## s(income,education) 4.94  6.303 75.41  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.824   Deviance explained = 83.3%
## GCV = 55.188  Scale est. = 51.974    n = 102
```

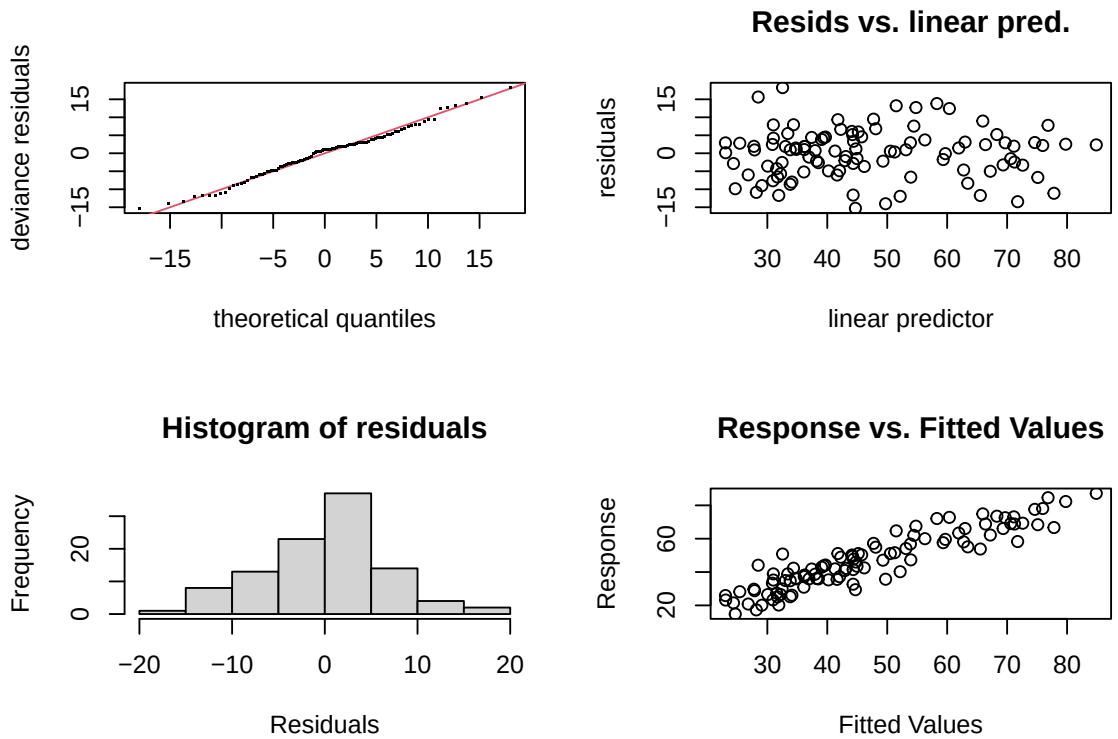
```
# plot(modelo2, se = FALSE)
```

En este caso el coeficiente de determinación ajustado es menor…

10.1.5 Diagnosis del modelo

La función `gam.check` realiza una diagnosis del modelo:

```
gam.check(modelo)
```




```
##
## Method: GCV   Optimizer: magic
## Smoothing parameter selection converged after 4 iterations.
## The RMS GCV score gradient at convergence was 9.783945e-05 .
## The Hessian was positive definite.
```

```
## Model rank =  19 / 19
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                 k'  edf k-index p-value
## s(income)    9.00 3.12    0.98    0.44
## s(education) 9.00 3.18    1.03    0.57
```

Lo ideal sería observar normalidad en los dos gráficos de la izquierda, falta de patrón en el superior derecho, y ajuste a una recta en el inferior derecho. En este caso parece que el modelo se comporta adecuadamente.

Capítulo 11

Programación

En este capítulo se introducirán los comandos básicos de programación en R...

11.1 Funciones

El lenguaje `R` permite al usuario definir sus propias funciones. El esquema de una función es el que sigue:

```
nombre <- function(arg1, arg2, ... ) {expresión}
```

- En la expresión anterior `arg1, arg2, ...` son los argumentos de entrada (también llamados parámetros).
- La `expresión` está compuesta de comandos que utilizan los argumentos de entrada para dar la **salida** deseada.
- La salida de una función puese ser un número, un vector, una grafica, un mensaje, etc.

11.1.1 Ejemplo: progresión geométrica

Para introducirnos en las funciones, vamos a escribir una función que permita trabajar con las llamadas **progresiones geométricas**.

Una progresión geométrica es una sucesión de números $a_1, a_2, a_3 ...$ tales que cada uno de ellos (salvo el primero) es igual al anterior multiplicado por una constante llamada **razón**, que representaremos por r. Ejemplos:

- $a_1 = 1$, $r = 2$:

  1, 2, 4, 8, 16,...

- $a_1 = -1$, $r = -2$:

  1, -2, 4, -8, 16,...

Según la definición anterior, se verifica que:

$$a_2 = a_1 \cdot r; \quad a_3 = a_2 \cdot r = a_1 \cdot r^2; \quad ...$$

y generalizando este proceso se obtiene el llamado término general:

$$a_n = a_1 \cdot r^{n-1}$$

También se puede comprobar que la suma de los n términos de la progresión es:

$$S_n = a_1 + ...+ a_n = \frac{a_1(r^n - 1)}{r - 1}$$

La siguiente función, que llamaremos `an` calcula el término a_n de una progresión geométrica pasando como entrada el primer elemento `a1`, la razón `r` y el valor `n`:

```
an <- function(a1, r, n) {
        a1 * r^(n - 1)
      }
```

A continuación algún ejemplo para comprobar su funcionamiento:

```
an(a1 = 1, r = 2, n = 5)
```

```
## [1] 16
```

```
an(a1 = 4, r = -2, n = 6)
```

```
## [1] -128
```

```
an(a1 = -50, r = 4, n = 6)
```

```
## [1] -51200
```

Con la función anterior se pueden obtener, con una sola llamada, varios valores de la progresión:

```
an(a1 = 1, r = 2, n = 1:5)    # a1, ..., a5
```

```
## [1]  1  2  4  8 16
```

```
an(a1 = 1, r = 2, n = 10:15)  # a10, ..., a15
```

```
## [1]   512  1024  2048  4096  8192 16384
```

La función `Sn` calcula la suma de los primeros `n` elementos de la progresión:

```
Sn <- function(a1, r, n) {
        a1 * (r^n - 1) / (r - 1)
      }

Sn(a1 = 1, r = 2, n = 5)
```

```
## [1] 31
```

```
an(a1 = 1, r = 2, n = 1:5)    # Valores de la progresión
```

```
## [1]  1  2  4  8 16
```

```
Sn(a1 = 1, r = 2, n = 1:5)    # Suma de los valores
```

```
## [1]  1  3  7 15 31
```

```
# cumsum(an(a1 = 1, r = 2, n = 1:5))
```

11.1.2 Argumentos de entrada

Como ya hemos comentado, los argumentos son los valores de entrada de una función.

- Por ejemplo, en la función anterior:

  ```
  an <- function(a1, r, n) {a1 * r^(n - 1)}
  ```

  los argumentos de entrada son `a1`, `r` y `n`.

Veamos alguna consideración sobre los argumentos:

- No es necesario utilizar el nombre de los argumentos. En este caso es obligatorio mantener el orden de entrada. Por ejemplo, las siguientes llamadas son equivalentes:

```
an(1, 2, 5)
```

```
## [1] 16
```

```
an(a1 = 1, r = 2, n = 5)
```

```
## [1] 16
```

- Si se nombran los argumentos, se pueden pasar en cualquier orden:

```
an(r = 2, n = 5, a1 = 1)
```

```
## [1] 16
```

```
an(n = 5, r = 2, a1 = 1)
```

```
## [1] 16
```

11.1.2.1 Argumentos por defecto

En muchas ocasiones resulta muy interesante que las funciones tengan argumentos por defecto.

Por ejemplo, si se quiere que en una función:

```
nombre <- function(arg1, arg2, arg3, arg4, ...) { expresión }
```

los argumento `arg2` y `arg3` tomen por defecto los valores `a` y `b` respectivamentebastaría con escribir:

```
nombre <- function(arg1, arg2 = a, arg3 = b, arg4, ...) { expresión }
```

Para comprender mejor esto considérese el siguiente ejemplo ilustrativo:

```
xy2 <- function(x = 2, y = 3) { x * y^2 }
xy2()
```

```
## [1] 18
```

```
xy2(x = 1, y = 4)
```

```
## [1] 16
```

```
xy2(y = 4)
```

```
## [1] 32
```

11.1.2.2 El argumento ...

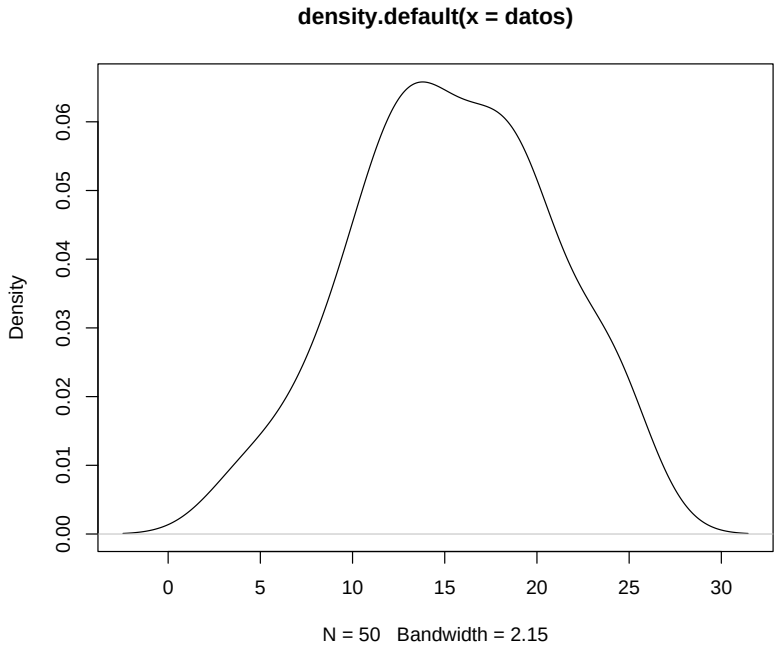
El argumento "`...`" permite pasar de manera "libre" argumentos adicionales para ser utilizados por otra "subfunción" dentro de la función principal.

Por ejemplo, en la función:

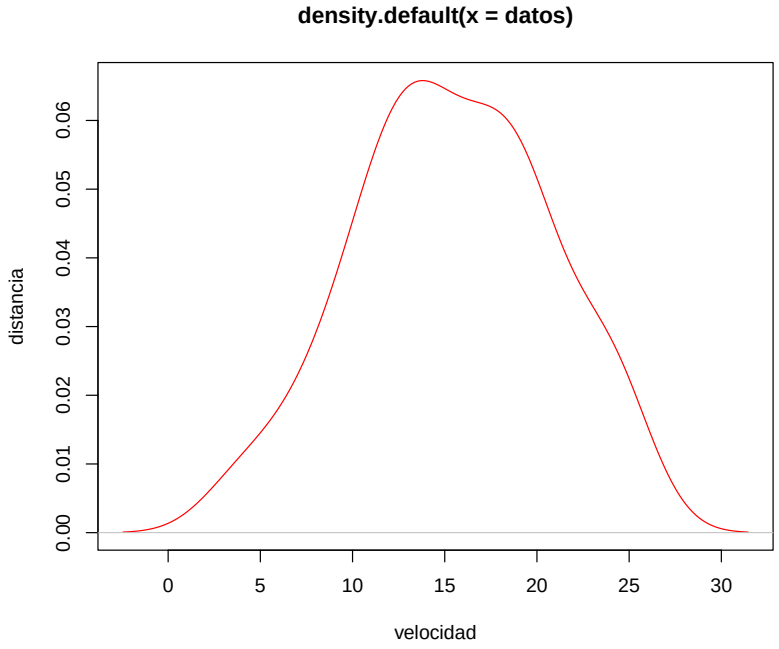
```
Density.Plot <- function(datos, ...) { plot(density(datos), ...) }
```

a partir del primer argumento, los argumentos se incluirán en `...` y serán utilizados por la función `plot`.

```
data(cars)
Density.Plot(cars$speed)
```




```
Density.Plot(cars$speed, col = 'red', xlab = "velocidad", ylab = "distancia")
```



Los argumentos de entrada de una función se obtienen ejecutando `args(funcion)`:

```
args(an)
```

```
## function (a1, r, n)
## NULL
```

```
args(xy2)
```

```
## function (x = 2, y = 3)
## NULL
```

```
str(args(Density.Plot))
```

```
## function (datos, ...)
```

Por otro lado, al escribir el nombre de una función se obtiene su contenido:

```
an
```

```
## function(a1, r, n) {
##         a1 * r^(n - 1)
##       }
## <bytecode: 0x000000001d6ff5f0>
```

11.1.3 Salida

El valor que devolverá una función será:

- el último objeto evaluado dentro de ella, o
- lo indicado dentro de la sentencia `return`.

Como las funciones pueden devolver objetos de varios tipos es hatibual que la salida sea una lista.

```
an <- function(a1, r, n) { a1 * r^(n - 1) }
Sn <- function(a1, r, n) { a1 * (r^n - 1) / (r - 1) }

asn <- function(a1 = 1, r = 2, n = 5) {
  A <- an(a1, r, n)
  S <- Sn(a1, r, n)
  ii <- 1:n
  AA <- an(a1, r, ii)
  SS <- Sn(a1, r, ii)
  return(list(an = A, Sn = S, salida = data.frame(valores = AA, suma = SS)))
}
```

La función `asn` utiliza las funiones `an` y `Sn` programadas antes y devuelve como salida una lista con las siguientes componentes:

- `an`: valor de a_n
- `Sn`: valor de S_n
- `salida`: data.frame con dos variables
  - `salida`: vector con las n primeras componentes de la progresión
  - `suma`: suma de las n primeras componentes

```
asn()
```

```
## $an
## [1] 16
##
## $Sn
## [1] 31
##
## $salida
##   valores suma
## 1       1    1
## 2       2    3
## 3       4    7
## 4       8   15
## 5      16   31
```

La salida de la función anterior es una lista y se puede acceder a los elementos de la misma:

```
res <- asn()
res$an
```

```
## [1] 16
```

```
res$Sn
```

```
## [1] 31
```

```
res$salida
```

```
##   valores suma
## 1       1    1
## 2       2    3
## 3       4    7
## 4       8   15
## 5      16   31
```

11.1.4 Otros ejemplos

11.1.4.1 Ejemplo: letra del DNI

A continuación se calculará la letra del DNI a partir de su correspondiente número. El método utilizado para obtener la letra del DNI consiste en dividir el número entre 23 y según el resto obtenido adjudicar la letra que figura en la siguiente tabla:

resto	letra	resto	letra	resto	letra
0	T	8	P	16	Q
1	R	9	D	17	V
2	W	10	X	18	H
3	A	11	B	19	L
4	G	12	N	20	C
5	M	13	J	21	K
6	Y	14	Z	22	E
7	F	15	S		

La siguiente función permite obtener la letra del DNI:

```
DNI <- function(numero) {
  letras <- c("T", "R", "W", "A", "G", "M", "Y", "F",
              "P", "D", "X", "B", "N", "J", "Z", "S",
              "Q", "V", "H", "L", "C", "K", "E")
  return(letras[numero %% 23 + 1])
}

DNI(50247828)
```

```
## [1] "G"
```

11.1.4.2 Ejemplo: simulación del lanzamiento de un dado

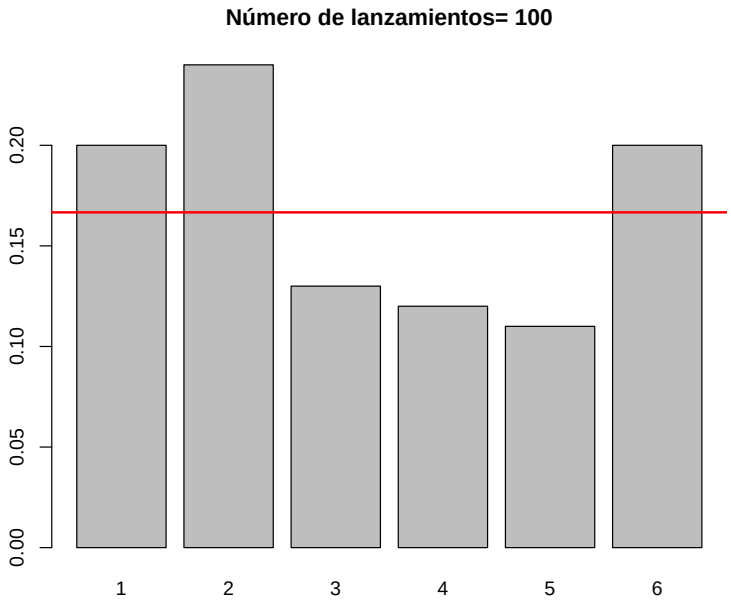
La siguiente función simula n (por defecto $n = 100$) lanzamientos de un dado. La función devuelve la tabla de frecuencias y realiza el correspondiente gráfico:

```
dado <- function(n = 100) {
  lanzamientos <- sample(1:6, n, rep = TRUE)
  frecuencias <- table(lanzamientos) / n
  barplot(frecuencias, main = paste("Número de lanzamientos=", n))
  abline(h = 1 / 6, col = 'red', lwd = 2)
```

```
  return(frecuencias)
}
```

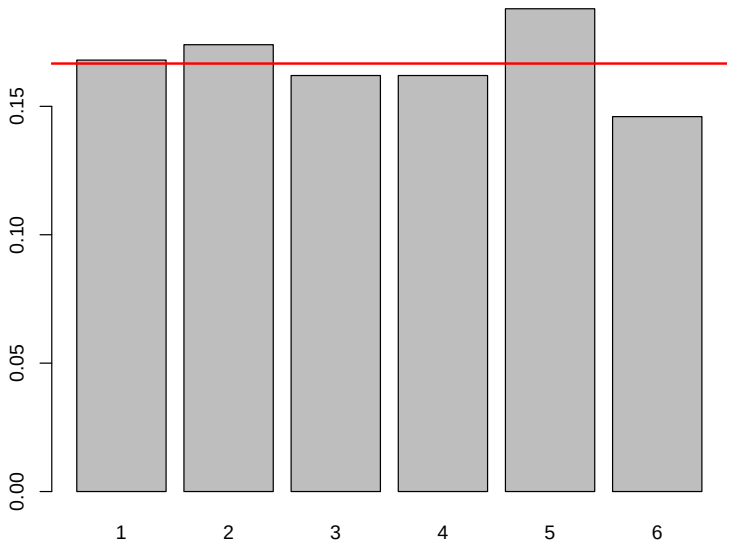
A continuación se muestran los resultados obtenidos para varias simulaciones:

```
dado(100)
```




```
## lanzamientos
##    1    2    3    4    5    6
## 0.20 0.24 0.13 0.12 0.11 0.20
```

```
dado(500)
```



```
## lanzamientos
##     1     2     3     4     5     6
## 0.168 0.174 0.162 0.162 0.188 0.146
```

```
dado(10000)
```


```
## lanzamientos
##      1      2      3      4      5      6
## 0.1690 0.1695 0.1657 0.1655 0.1629 0.1674
```

Se puede comprobar que al aumentar el valor de n las frecuencias se aproximan al valor teórico $1/6 = 0.1667$.

11.1.5 Variables locales y globales

En `R` no es necesario declarar las variables usadas dentro de una función. Se utiliza la regla llamada "ámbito lexicográfico" para decidir si un objeto es local a una función o global.

Para entender mejor esto se consideran los siguientes ejemplos:

```
fun <- function() print(x)
x <- 1
fun()
```

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

La variable `x` no está definida dentro de `fun`, así que `R` busca `x` en el entorno en el que se llamó a la función e imprimirá su valor.

Si `x` es utilizado como el nombre de un objeto dentro de la función, el valor de `x` en el ambiente global (fuera de la función) no cambia.

```
x <- 1
fun2 <- function() {
    x <- 2
    print(x)
}
```

```
fun2()
```

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

```
x
```

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

Para que el valor "global" de una variable pueda ser cambidado dentro de una función se utiliza la doble asignación `<<-`.

```
x <- 1
y <- 3
fun2 <- function() {
    x <- 2
    y <<- 5
    print(x)
    print(y)
}

fun2()
```

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

```
x # No cambió su valor
```

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

```
y # Cambió su valor
```

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

11.2 Ejecución condicional

Para hacer ejecuciones condicionales de código se usa el comando `if` con sintaxis:

```
if (condicion1) {expresión1} else {expresión2}
```

La siguiente función comprueba si un número es múltiplo de dos:

```
multiplo2 = function(x) {
  if (x %% 2 == 0) {
    print(paste(x,'es múltiplo de dos'))
  } else {
    print(paste(x,'no es múltiplo de dos'))
  }
}

multiplo2(5)
```

```
## [1] "5 no es múltiplo de dos"
```

```
multiplo2(-2.3)
```

```
## [1] "-2.3 no es múltiplo de dos"
```

```
multiplo2(10)
```

```
## [1] "10 es múltiplo de dos"
```

11.3 Bucles y vectorización

11.3.1 Bucles

`R` permite crear bucles repetitivos (loops) y la ejecución condicional de sentencias. `R` admite bucles `for`, `repeat` and `while`.

11.3.1.1 El bucle `for`

La sintaxis de un bucle `for` es la que sigue:

```
for (i in lista_de_valores)  { expresión }
```

Por ejemplo, dado un vector x se puede calcular $y = x^2$ con el código:

```
x <- seq(-2, 2, 0.5)
n <- length(x)
y <- numeric(n) # Es necesario crear el objeto para acceder a los componentes...
for (i in 1:n) { y[i] <- x[i] ^ 2 }
x
```

```
## [1] -2.0 -1.5 -1.0 -0.5  0.0  0.5  1.0  1.5  2.0
```

```
y
```

```
## [1] 4.00 2.25 1.00 0.25 0.00 0.25 1.00 2.25 4.00
```

```
x^2
```

```
## [1] 4.00 2.25 1.00 0.25 0.00 0.25 1.00 2.25 4.00
```

Otro ejemplo:

```
for(i in 1:5) print(i)
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

El siguiente código simula gráficamente el segundero de un reloj:

```
angulo <- seq(0, 360, by = 6)
radianes <- angulo * pi / 180
x <- sin(radianes)
y <- cos(radianes)

sec <- seq(6, 61, by = 5)
for (i in 1:61) {
  plot(x, y, axes = FALSE, xlab = "", ylab = "", type = 'l', col = 'grey')
  points(x[i], y[i])
  # Añadir "decoración"
  text(x[sec]*0.9, y[sec]*0.9, labels = sec - 1)
  arrows(0, 0, x[i]*0.85, y[i]*0.85, col = 'blue')
  # Esperar un segundo
  Sys.sleep(1)
}
```

11.3.1.2 El bucle `while`

La sintaxis del bucle `while` es la que sigue:

```
while (condición lógica)  { expresión }
```

Por ejemplo, si queremos calcular el primer número entero positivo cuyo cuadrado no excede de 5000, podemos hacer:

```
cuadrado <- 0
n <- 0
while (cuadrado <= 5000) {
  n <- n + 1
  cuadrado <- n^2
}
cuadrado
```

```
## [1] 5041
```

```
n
```

```
## [1] 71
```

```
n^2
```

```
## [1] 5041
```

Nota: Dentro de un bucle se puede emplear el comando `break` para terminarlo y el comando `next` para saltar a la siguiente iteración.

11.3.2 Vectorización

Como hemos visto en `R` se pueden hacer bucles. Sin embargo, es preferible evitar este tipo de estructuras y tratar de utilizar **operaciones vectorizadas** que son mucho más eficientes desde el punto de vista computacional.

Por ejemplo para sumar dos vectores se puede hacer con un `for`:

```
x <- c(1, 2, 3, 4)
y <- c(0, 0, 5, 1)
n <- length(x)
z <- numeric(n)
for (i in 1:n) {
  z[i] <- x[i] + y[i]
}
z
```

```
## [1] 1 2 8 5
```

Sin embargo, la operación anterior se podría hacer de modo más eficiente en modo vectorial:

```
z <- x + y
z
```

```
## [1] 1 2 8 5
```

11.3.3 Funciones `apply`

11.3.3.1 La función `apply`

Una forma de evitar la utilización de bucles es utilizando la función `apply()` que permite evaluar una misma función en todas las filas, columnas, …. de un array de forma simultánea.

La sintaxis de esta función es:

```
apply(X, MARGIN, FUN, ...)
```

- `X`: matriz (o array)

- `MARGIN`: Un vector indicando las dimensiones donde se aplicará la función. 1 indica filas, 2 indica columnas, y `c(1,2)` indica filas y columnas.
- `FUN`: función que será aplicada.
- `...`: argumentos opcionales que serán usados por `FUN`.

Veamos la utilización de la función `apply` con un ejemplo:

```
x <- matrix(1:9, nrow = 3)
x
```

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

```
apply(x, 1, sum)    # Suma por filas
```

```
## [1] 12 15 18
```

```
apply(x, 2, sum)    # Suma por columnas
```

```
## [1]  6 15 24
```

```
apply(x, 2, min)    # Mínimo de las columnas
```

```
## [1] 1 4 7
```

```
apply(x, 2, range)  # Rango (mínimo y máximo) de las columnas
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    3    6    9
```

11.3.3.2 La función `tapply`

La function `tapply` es similar a la función `apply` y permite aplicar una función a los datos desagregados, utilizando como criterio los distintos niveles de una variable factor. La sintaxis de esta función es como sigue:

```
    tapply(X, INDEX, FUN, ...,)
```

- `X`: matriz (o array).
- `INDEX`: factor indicando los grupos (niveles).
- `FUN`: función que será aplicada.
- `...`: argumentos opcionales .

Consideremos, por ejemplo, el data.frame `ChickWeight` con datos de un experimento relacionado con la repercusión de varias dietas en el peso de pollos.

```
data(ChickWeight)
head(ChickWeight)
```

```
##   weight Time Chick Diet
## 1     42    0     1    1
## 2     51    2     1    1
## 3     59    4     1    1
## 4     64    6     1    1
## 5     76    8     1    1
## 6     93   10     1    1
```

```
peso <- ChickWeight$weight
dieta <- ChickWeight$Diet
levels(dieta) <- c("Dieta 1", "Dieta 2", "Dieta 3", "Dieta 4")
tapply(peso, dieta, mean)  # Peso medio por dieta
```

```
##  Dieta 1  Dieta 2  Dieta 3  Dieta 4
## 102.6455 122.6167 142.9500 135.2627
```

```
tapply(peso, dieta, summary)
```

```
## $`Dieta 1`
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   35.00   57.75   88.00  102.65  136.50  305.00
##
## $`Dieta 2`
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    39.0    65.5   104.5   122.6   163.0   331.0
##
## $`Dieta 3`
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    39.0    67.5   125.5   142.9   198.8   373.0
##
## $`Dieta 4`
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   39.00   71.25  129.50  135.26  184.75  322.00
```

Otro ejemplo:

```
provincia <- as.factor(c(1, 3, 4, 2, 4, 3, 2, 1, 4, 3, 2))
levels(provincia) = c("A Coruña", "Lugo", "Orense", "Pontevedra")
hijos <- c(1, 2, 0, 3, 4, 1, 0, 0, 2, 3, 1)
data.frame(provincia, hijos)
```

```
##     provincia hijos
## 1    A Coruña     1
## 2      Orense     2
## 3  Pontevedra     0
## 4        Lugo     3
## 5  Pontevedra     4
## 6      Orense     1
## 7        Lugo     0
## 8    A Coruña     0
## 9  Pontevedra     2
## 10     Orense     3
## 11       Lugo     1
```

```
tapply(hijos, provincia, mean) # Número medio de hijos por provincia
```

```
##   A Coruña       Lugo     Orense Pontevedra
##   0.500000   1.333333   2.000000   2.000000
```

11.4 Aplicación: validación cruzada

Si deseamos evaluar la calidad predictiva de un modelo, lo ideal es disponer de suficientes datos para poder hacer dos grupos con ellos: una muestra de entrenamiento y otra de validación. Cuando hacer esto no es posible, disponemos como alternativa de la *validación cruzada*, una herramienta que permite estimar los errores de predicción utilizando una única muestra de datos. En su versión más simple (llamada en inglés *leave-one-out*):

- se utilizan todos los datos menos uno para realizar el ajuste, y se mide su error de predicción en el único dato no utilizado;

- a continuación se repite el proceso utilizando, uno a uno, todos los puntos de la muestra de datos;
- y finalmente se combinan todos los errores en un único error de predicción.

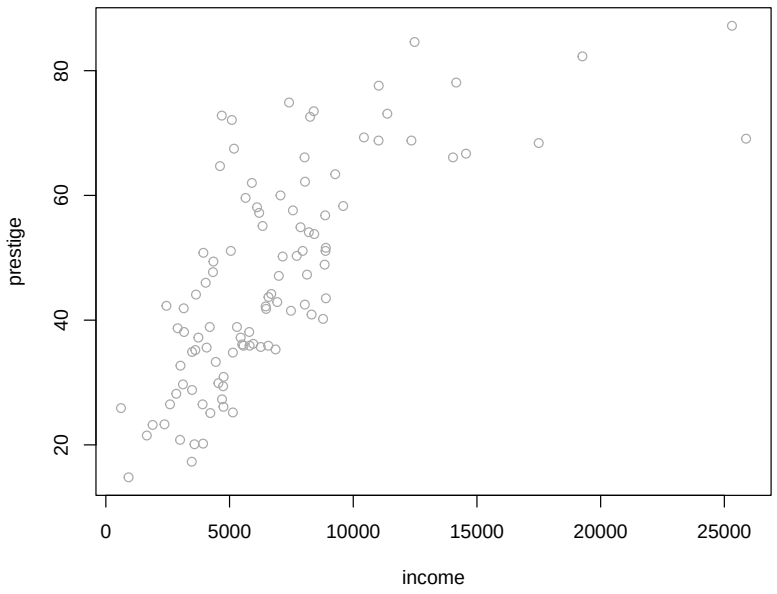
El proceso anterior se puede generalizar repartiendo los datos en distintos grupos, más o menos del mismo tamaño, y sustituyendo en la explicación anterior dato por grupo.

11.4.1 Primer ejemplo

Cuando disponemos de unos datos y los queremos ajustar utilizando un modelo que depende de un parámetro, por ejemplo un modelo de regresión polinómico que depende del grado del polinomio, podemos utilizar la validación cruzada para seleccionar el grado del polinomio que debemos utilizar.

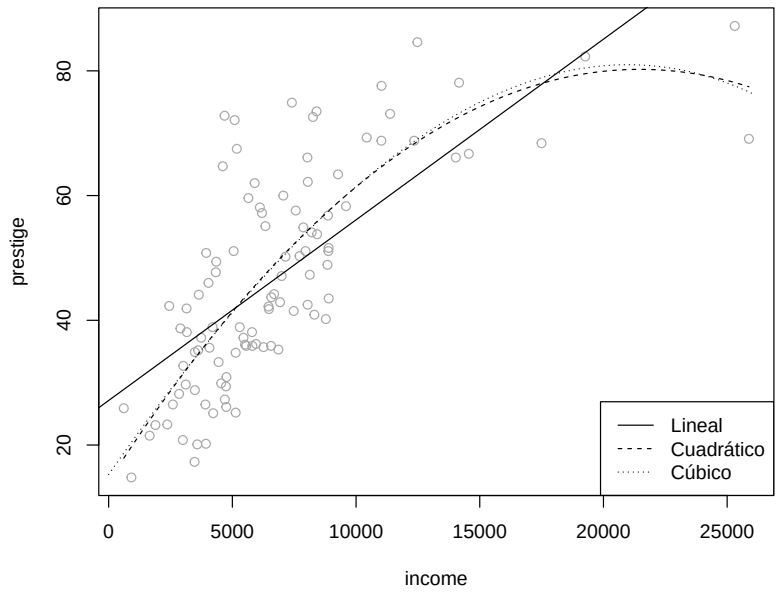
Veámoslo utilizando las variables *income* y *prestige* de la base de datos *Prestige*, incluida en el paquete *car*.

```
library(car)
plot(prestige ~ income, data = Prestige, col = 'darkgray')
```



Representemos, gráficamente, los ajustes lineal, cuadrático y cúbico.

```
plot(prestige ~ income, data = Prestige, col = 'darkgray')
# Ajuste lineal
abline(lm(prestige ~ income, data = Prestige))
# Ajuste cuadrático
modelo <- lm(prestige ~ income + I(income^2), data = Prestige)
parest <- coef(modelo)
curve(parest[1] + parest[2]*x + parest[3]*x^2, lty = 2, add = TRUE)
# Ajuste cúbico
modelo <- lm(prestige ~ poly(income, 3), data = Prestige)
valores <- seq(0, 26000, len = 100)
pred <- predict(modelo, newdata = data.frame(income = valores))
lines(valores, pred, lty = 3)
legend("bottomright", c("Lineal","Cuadrático","Cúbico"), lty = 1:3)
```

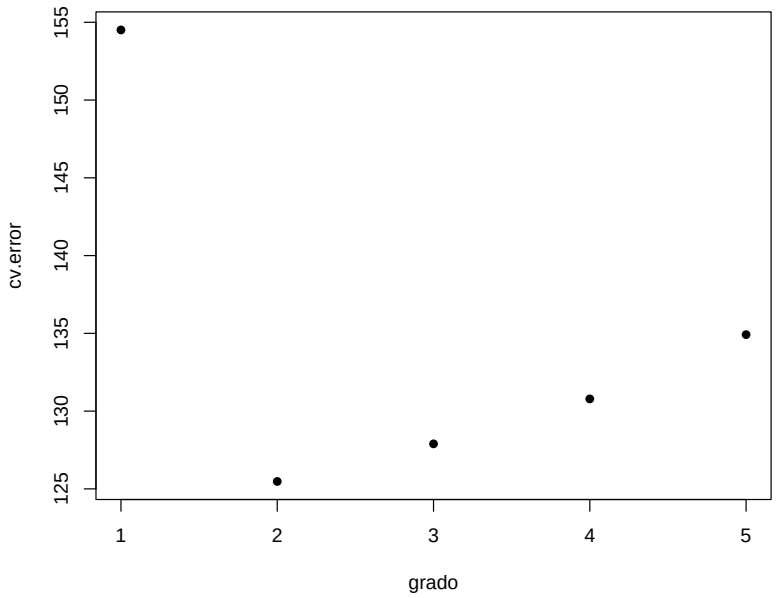


Vamos a escribir una función que nos devuelva, para cada dato (fila) de *Prestige*, la predicción en ese punto ajustando el modelo con todos los demás puntos.

```
cv.lm <- function(formula, datos) {
      n <- nrow(datos)
      cv.pred <- numeric(n)
      for (i in 1:n) {
          modelo <- lm(formula, datos[-i, ])
          cv.pred[i] <- predict(modelo, newdata = datos[i, ])
      }
      return(cv.pred)
}
```

Por último, calculamos el error de predicción (en este caso el *error cuadrático medio*) en los datos de validación. Repetimos el proceso para cada valor del parámetro (grado del ajuste polinómico) y minimizamos.

```
grado <- 1:5
cv.error <- numeric(5)
for(p in grado){
  cv.pred <- cv.lm(prestige ~ poly(income, p), Prestige)
  cv.error[p] <- mean((cv.pred - Prestige$prestige)^2)
}
plot(grado, cv.error, pch=16)
```




```
grado[which.min(cv.error)]
```

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

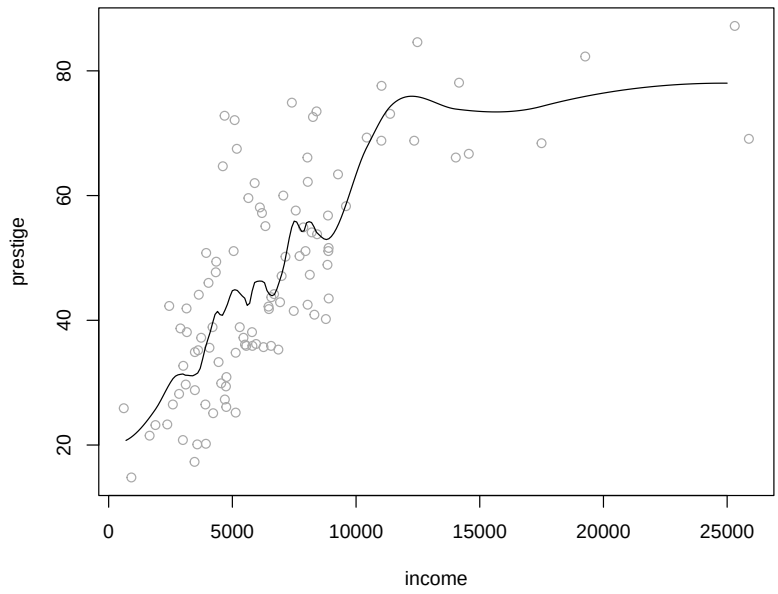
11.4.2 Segundo ejemplo

En este segundo ejemplo vamos a aplicar una técnica de modelado *local* al problema de regresión del ejemplo anterior. El enfoque es *data-analytic* en el sentido de que no nos limitamos a una familia de funciones que dependen de unos parámetros (enfoque paramétrico), que son los que tenemos que determinar, sino que las funciones de regresión están determinadas por los datos. Aun así, sigue habiendo un parámetro que controla el proceso, cuyo valor debemos fijar siguiendo algún criterio de optimalidad.

Vamos a realizar, utilizando la función `loess`, un *ajuste polinómico local robusto*, que depende del parámetro `span`, que podemos interpretar como la proporción de datos empleada en el ajuste.

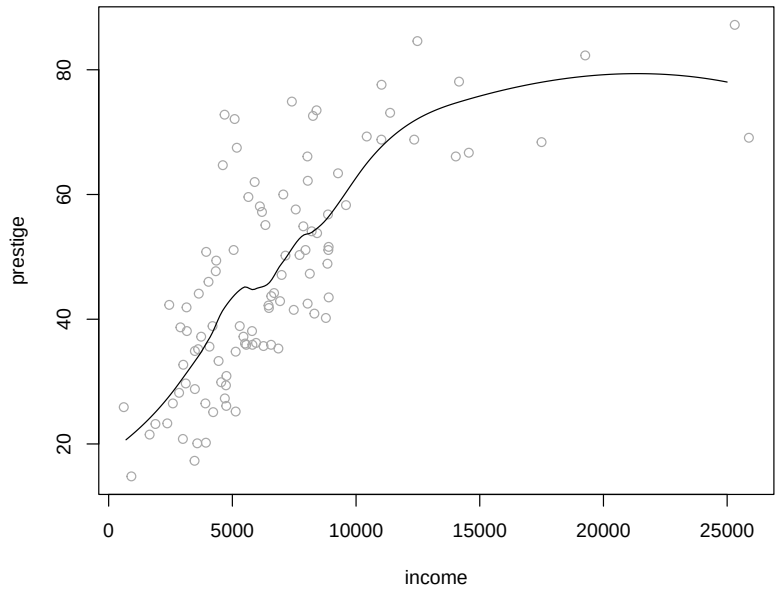
Utilizando un valor `span=0.25`:

```
plot(prestige ~ income, Prestige, col = 'darkgray')
fit <- loess(prestige ~ income, Prestige, span = 0.25)
valores <- seq(0, 25000, 100)
pred <- predict(fit, newdata = data.frame(income = valores))
lines(valores, pred)
```



Si utilizamos `span=0.5`:

```
plot(prestige ~ income, Prestige, col = 'darkgray')
fit <- loess(prestige ~ income, Prestige, span = 0.5)
valores <- seq(0, 25000, 100)
pred <- predict(fit, newdata = data.frame(income = valores))
lines(valores, pred)
```



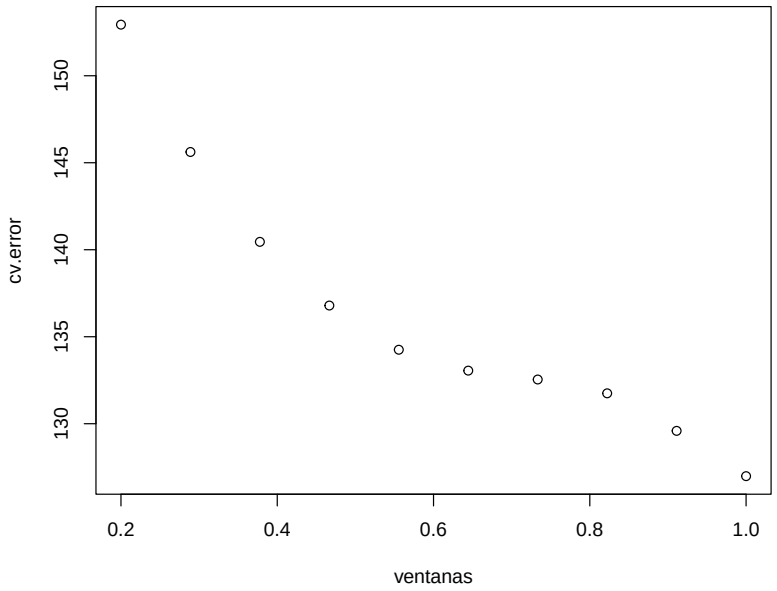
Nuestro objetivo es seleccionar un valor razonable para **span**, y lo vamos a hacer utilizando validación cruzada y minimizando el error cuadrático medio de la predicción en los datos de validación.

Utilizando la función

```
cv.loess <- function(formula, datos, p) {
  n <- nrow(datos)
  cv.pred <- numeric(n)
  for (i in 1:n) {
    modelo <- loess(formula, datos[-i, ], span = p,
                    control = loess.control(surface = "direct"))
    # control = loess.control(surface = "direct") permite extrapolaciones
    cv.pred[i] <- predict(modelo, newdata = datos[i, ])
  }
  return(cv.pred)
}
```

y procediendo de modo similar al caso anterior:

```
ventanas <- seq(0.2, 1, len = 10)
np <- length(ventanas)
cv.error <- numeric(np)
for(p in 1:np){
  cv.pred <- cv.loess(prestige ~ income, Prestige, ventanas[p])
  cv.error[p] <- mean((cv.pred - Prestige$prestige)^2)
  # cv.error[p] <- median(abs(cv.pred - Prestige$prestige))
}

plot(ventanas, cv.error)
```



obtenemos la ventana "óptima" (en este caso el valor máximo):

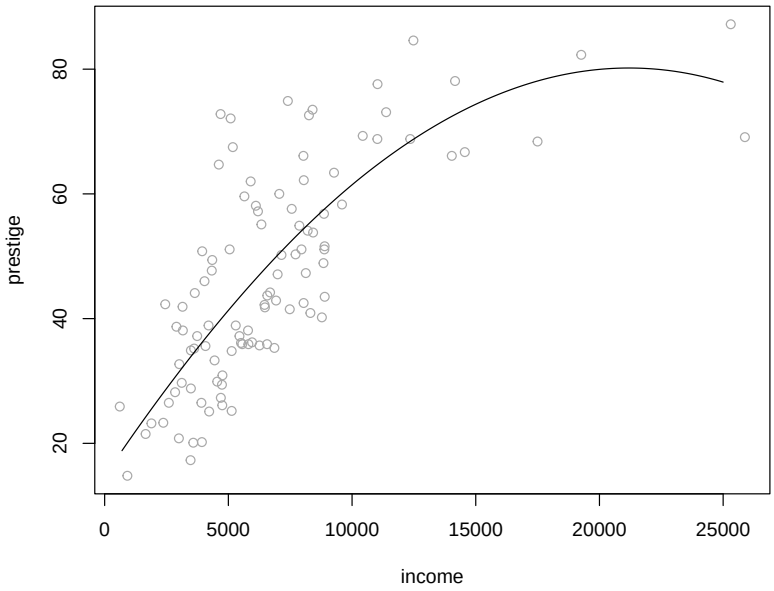
```
span <- ventanas[which.min(cv.error)]
span
```

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

y la correspondiente estimación:

```
plot(prestige ~ income, Prestige, col = 'darkgray')
fit <- loess(prestige ~ income, Prestige, span = span)
```

```
valores <- seq(0, 25000, 100)
pred <- predict(fit, newdata = data.frame(income = valores))
lines(valores, pred)
```



Capítulo 12

Generación de informes

Una versión más completa de este capítulo está disponible en el apéndice del libro Escritura de libros con bookdown.

12.1 R Markdown

R-Markdown es recomendable para difundir análisis realizados con R en formato HTML, PDF y DOCX (Word), entre otros.

12.1.1 Introducción

R-Markdown permite combinar Markdown con R. Markdown se diseñó inicialmente para la creación de páginas web a partir de documentos de texto de forma muy sencilla y rápida (tiene unas reglas sintácticas muy simples). Actualmente gracias a múltiples herramientas como pandoc permite generar múltiples tipos de documentos (incluido LaTeX; ver Pandoc Markdown)

Para más detalles ver http://rmarkdown.rstudio.com.

También se dispone de información en la ayuda de *RStudio*:

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

Al renderizar un fichero rmarkdown se generará un documento que incluye el código R y los resultados incrustados en el documento. En *RStudio* basta con hacer clic en el botón **Knit HTML**. En R se puede emplear la funcion `render` del paquete *rmarkdown* (por ejemplo: `render("8-Informes.Rmd")`). También se puede abrir directamente el informe generado:

```
library(rmarkdown)
browseURL(url = render("8-Informes.Rmd"))
```

12.1.2 Inclusión de código R

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

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

```
##                   mpg cyl disp
## Mazda RX4         21.0   6  160
## Mazda RX4 Wag     21.0   6  160
## Datsun 710        22.8   4  108
```

```
## Hornet 4 Drive    21.4   6  258
## Hornet Sportabout 18.7   8  360
## Valiant           18.1   6  225
```

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

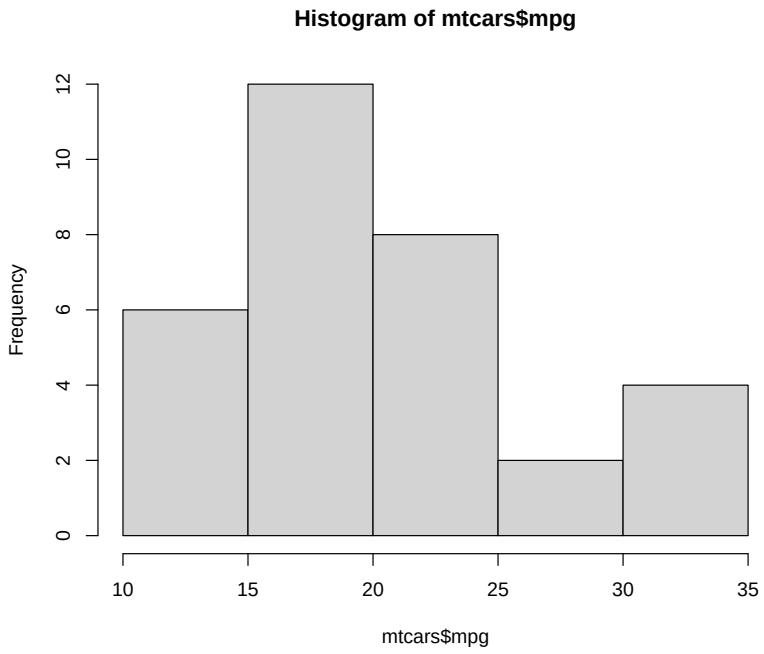
```
##       mpg             cyl             disp
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8
##  Median :19.20   Median :6.000   Median :196.3
##  Mean   :20.09   Mean   :6.188   Mean   :230.7
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0
##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

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

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

12.1.3 Inclusión de gráficos

Se pueden generar gráficos:



Los trozos de código pueden tener nombre y opciones, se establecen en la cabecera de la forma ```` ```{r nombre, op1, op2} ```` (en el caso anterior no se muestra el código, al haber empleado ```` ```{r, echo=FALSE} ````). Para un listado de las opciones disponibles ver http://yihui.name/knitr/options.

En *RStudio* se puede pulsar en los iconos a la derecha del chunk para establecer opciones, ejecutar todo el código anterior o sólo el correspondiente trozo.

12.1.4 Inclusión de tablas

Las tablas en markdown son de la forma:

```
| First Header  | Second Header |
| ------------- | ------------- |
| Row1 Cell1    | Row1 Cell2    |
| Row2 Cell1    | Row2 Cell2    |
```

Tabla 12.2: Una kable knitr

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

Por ejemplo:

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

Para convertir resultados de R en tablas de una forma simple se puede emplear la función `kable` del paquete *knitr*:

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

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

12.1.5 Extracción del código R

Para generar un fichero con el código R se puede emplear la función `purl` del paquete *knitr*. Por ejemplo:

```
purl("8-Informes.Rmd")
```

Si se quiere además el texto rmarkdown como comentarios tipo *spin*, se puede emplear:

```
purl("8-Informes.Rmd", documentation = 2)
```

12.2 Spin

Una forma rápida de crear este tipo de informes a partir de un fichero de código R es emplear la funcion `spin` del paquete *knitr* (ver p.e. http://yihui.name/knitr/demo/stitch).

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

- El texto rmarkdown se comenta con `#'`. Por ejemplo: `#' # Este es un título de primer nivel  #' ## Este es un título de segundo nivel`
- Las opciones de un trozo de código se comentan con `#+`. Por ejemplo: `#+ setup, include=FALSE opts_chunk$set(comment=NA, prompt=TRUE, dev='svg', fig.height=6, fig.width=6)`

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

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

Pero puede ser recomendable renderizarlo con rmarkdown:

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

En *RStudio* basta con pulsar "Ctrl + Shift + K" o seleccionar *File > Knit Document* (en las últimas versiones también *File > Compile Notebook* o hacer clic en el icono correspondiente).

Referencias

Fernández-Casal R., Costa J. y Oviedo de la Fuente, M. (2021). *Aprendizaje Estadístico*. github.

Gil Bellosta C.J. (2018). *R para profesionales de los datos: una introducción*.

Grolemund, G. (2014). *Hands-on programming with R: Write your own functions and simulations*, O'Reilly.

Matloff, N. (2011). *The art of R programming: A tour of statistical software design*, No Starch Press.

Quintela del Rio A. (2019). *Estadística Básica Edulcorada*

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

Enlaces

Repositorio: rubenfcasal/intror

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

Bookdown:

- Fernández-Casal, R. y Cotos-Yáñez, T.R. (2018). *Escritura de libros con bookdown*, github. Incluye un apéndice con una Introducción a RMarkdown.
- Kuhn, M. y Silge, J. (2022). *Tidy Modeling with R*. O'Reill.
- Wickham, H. (2015). *R packages: organize, test, document, and share your code* (actualmente 2ª edición en desarrollo con H. Bryan), O'Reilly, 1ª edición.
- Wickham, H. (2019). *Advanced R, 2ª edición*, Chapman & Hall, 1ª edición..

*__Posit (RStudio)__

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

- tidymodels
- sparklyr
- shiny

Bibliografía complementaria

Beeley (2015). *Web Application Development with R Using Shiny*. Packt Publishing.

Bivand *et al.* (2008). *Applied Spatial Data Analysis with R*. Springer.

James *et al.* (2008). *An Introduction to Statistical Learning: with Aplications in R*. Springer.

Kolaczyk y Csárdi (2014). *Statistical analysis of network data with R*. Springer.

Munzert *et al.* (2014). *Automated Data Collection with R: A Practical Guide to Web Scraping and Text Mining*. Wiley.

Ramsay *et al.* (2009). *Functional Data Analysis with R and MATLAB*. Springer.

Van der Loo y de Jonge (2012). *Learning RStudio for R Statistical Computing*. Packt Publishing.

Williams (2011). *Data Mining with Rattle and R*. Springer.

Wood (2006). *Generalized Additive Models: An Introduction with R*. Chapman.

Yihui Xie (2015). *Dynamic Documents with R and knitr*. Chapman.

Apéndice A

Instalación de R

En la web del proyecto R (www.r-project.org) está disponible mucha información sobre este entorno estadístico.

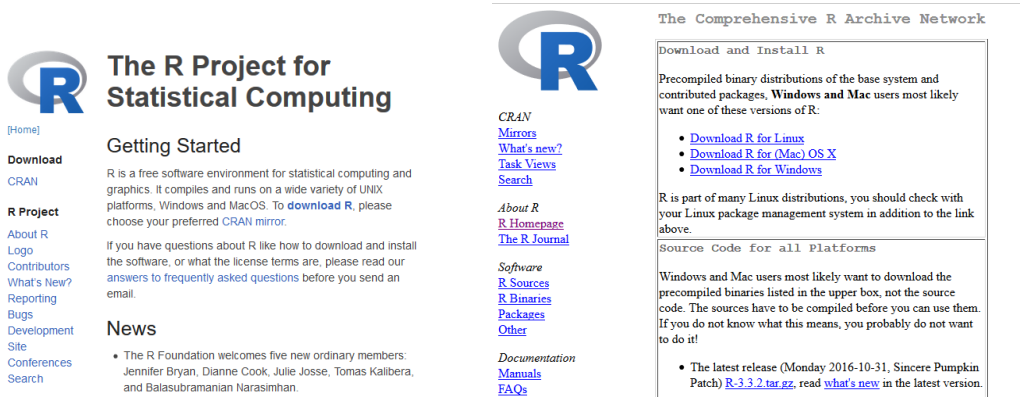


Figura A.1: Web de [R-project](https://r-project.org) y [CRAN](https://cran.r-project.org).

Las descargas se realizan a través de la web del CRAN (The Comprehensive R Archive Network), con múltiples mirrors:

- *Oficina de Software Libre (A Coruña)* (CIXUG): ftp.cixug.es/CRAN.
- *Spanish National Research Network (Madrid)* (RedIRIS): cran.es.r-project.org.

A.1 Instalación de R en Windows

Seleccionando Download R for Windows y posteriormente base accedemos al enlace con el instalador de R para Windows (actualmente de la versión 4.2.2).

A.1.1 Asistente de instalación

Durante el proceso de instalación la recomendación (para evitar posibles problemas) es seleccionar ventanas simples SDI en lugar de múltiples ventanas MDI (hay que utilizar *Opciones de configuración*).

Una vez terminada la instalación, al abrir R aparece la ventana de la consola (simula una ventana de comandos de Unix) que permite ejecutar comandos de R.

Por defecto se instalan un conjunto de paquetes base de R (que se cargan automáticamente al iniciarlo) y un conjunto de paquetes recomendados (que se pueden cargar empleando el comando `library()`), pero hay disponibles miles de paquetes que cubren literalmente todos los campos del análisis de datos. Ver por ejemplo:

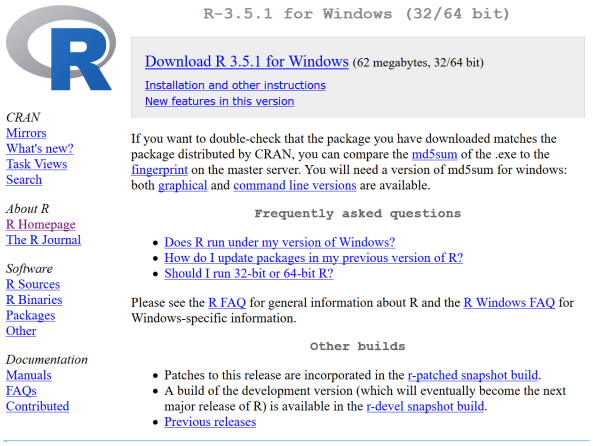


Figura A.2: Web de [descarga de R para Windows](http://ftp.cixug.es/CRAN/bin/windows).

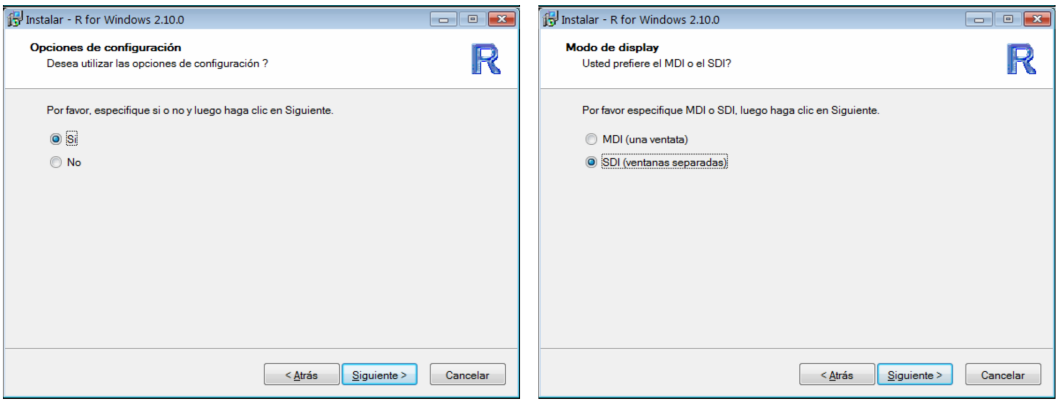


Figura A.3: Pasos del asistente para instalación de R en Windows.

- CRAN: Packages
- CRAN: Task Views

A.1.2 Instalación de paquetes

Después de la instalación de R suele ser necesario instalar paquetes adicionales (puede ser recomendable *Ejecutar como administrador* R para evitar problemas de permiso de escritura en la carpeta *C:\Program Files\R\R-X.Y.Z\library*, o cambiar previamente los permisos de esta carpeta como se indica aquí).

Para ejecutar los ejemplos mostrados en el libro sería necesario tener instalados los siguientes paquetes: `lattice`, `ggplot2`, `foreign`, `car`, `leaps`, `MASS`, `RcmdrMisc`, `lmtest`, `glmnet`, `mgcv`, `rmarkdown`, `knitr`, `dplyr`, `tidyr`. Por ejemplo mediante los siguientes comandos:

```
pkgs <- c("lattice", "ggplot2", "foreign", "car", "leaps", "MASS", "RcmdrMisc",
          "lmtest", "glmnet", "mgcv", "rmarkdown", "knitr", "dplyr", "tidyr")
install.packages(setdiff(pkgs, installed.packages()[,"Package"]), dependencies = TRUE)
```

(puede que haya que seleccionar el repositorio de descarga, e.g. *Spain (Madrid)*).

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

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

A.1.2.1 Cambiar los permisos de la carpeta *library* (opcional)

Para evitar problemas con la instalación de paquetes en Windows (y evitar también que los paquetes se instalen en *Documentos\R\win-library\X.Y*) se puede dar permiso de *control total* a los usuarios del equipo en el subdirectorio *library* de la instalación de R. Para ello, pulsar con el botón derecho en esta carpeta (e.g. *C:\Program Files\R\R-4.2.2\library*), seleccionar *Propiedades* > *Seguridad* > *Editar*, seleccionar los *Usuarios* del equipo, marcar *Control total* y *Aplicar*.

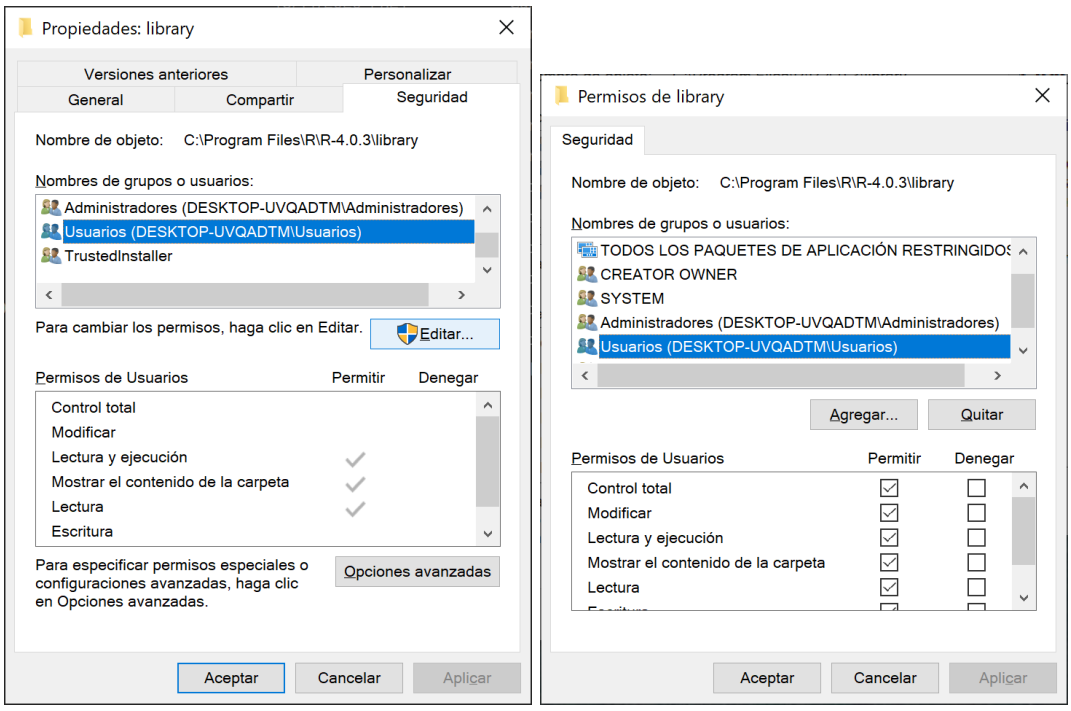


Figura A.4: Pasos en Windows para cambiar permisos en la carpeta library.

A.1.3 Instalación de RStudio Desktop

Aunque la consola de R dispone de un editor básico de código (script), puede ser recomendable trabajar con un editor de comandos más cómodo y flexible. El entorno de desarrollo (*Integrated Development Environment*, IDE) recomendado es RStudio. Está disponible para la mayoría de plataformas[1] e integra una gran cantidad de herramientas, que permiten desde la generación de informes, hasta la gestión de distintos tipos de proyectos, depuración de código, control de versiones, etc. También es compatible con otros lenguajes, incluido Python.

Una vez instalado R, para instalar RStudio Desktop basta con descargar el correspondiente archivo de instalación de *https://posit.co/download/rstudio-desktop* y seguir las instrucciones.

Este entorno se describe en la Sección 1.3.

A.1.3.1 Configuración adicional de RStudio (opcional)

En lugar de emplear los visores de gráficos, ayuda y navegador web integrados, nos puede interesar que los gráficos se muestren en ventanas independientes y las páginas web en el navegador del equipo. Esto se puede conseguir modificando los archivos de configuración (en el directorio *C:\Program Files\RStudio\R* en Windows y */Applications/RStudio.app/Contents/Resources/R* en Linux), que normalmente habrá que editar como administrador.

[1] También hay una versión para servidores: RStudio Server

Por defecto los gráficos generados desde RStudio se mostrarán en la pestaña *Plots* panel inferior derecho y por ejemplo puede aparecer errores si el área gráfica es demasiado pequeña. Para utilizar el dispositivo gráfico de R habría que modificar las siguientes líneas de *C:\Program Files\RStudio\R\Tools.R*:

```
# set our graphics device as the default and cause it to be created/set
.rs.addFunction( "initGraphicsDevice", function()
{
   # options(device="RStudioGD")
   # grDevices::deviceIsInteractive("RStudioGD")
  grDevices::deviceIsInteractive()
})
```

El visor integrado de RStudio no resulta muy cómodo para navegar por la ayuda de las funciones (por ejemplo no permite hacer zoom o abrir múltiples ventanas). Para utilizar en su lugar el navegador del equipo habría que comentar las siguientes líneas de *C:\Program Files\RStudio\R\Options.R*:

```
# # custom browseURL implementation.
# .rs.setOption("browser", function(url)
# {
   # .Call("rs_browseURL", url, PACKAGE = "(embedding)")
# })
```

A.2 Instalación de R en Ubuntu/Devian

Instalar dependencias:

```
sudo apt install libcurl4-gnutls-dev libgit2-dev libxml2-dev libssl-dev
```

Si aparecen problemas asegurarse de que los repositorios *universe* y *multiverse* están disponibles:

```
sudo add-apt-repository universe
sudo add-apt-repository multiverse
sudo apt update
```

Se puede instalar R desde estos repositorios, pero normalmente no será la versión más actualizada y no lo recomendaría.

A.2.1 Instalación de R desde CRAN

Añadir la llave de firma GPG, añadir el repositorio CRAN a la lista de fuentes (para ver la versión de ubuntu se puede ejecutar `lsb_release -a`, el siguiente código ya la obtiene directamente) e instalar R:

```
# Cambiar a root (alternativamente añadir `sudo` al principio de los comandos)
sudo -i
# update indices
apt update -qq
# install two helper packages we need
apt install --no-install-recommends software-properties-common dirmngr
# add the signing key (by Michael Rutter) for these repos
# To verify key, run gpg --show-keys /etc/apt/trusted.gpg.d/cran_ubuntu_key.asc
# Fingerprint: 298A3A825C0D65DFD57CBB651716619E084DAB9
wget -qO- https://cloud.r-project.org/bin/linux/ubuntu/marutter_pubkey.asc | sudo tee -a /etc/apt/tr
# add the R 4.0 repo from CRAN -- adjust 'focal' to 'groovy' or 'bionic' as needed
add-apt-repository "deb https://cloud.r-project.org/bin/linux/ubuntu $(lsb_release -cs)-cran40/"
apt-get update
apt-get install r-base r-base-dev
logout
```

A.2.2 Instalación de devtools y demás paquetes

Ejecutar R en modo administrador para que los paquetes que se instalen estén disponibles para todos los usuarios

```
sudo -i R
```

Ejecutar en la consola de R

```
install.packages('devtools', dependencies = TRUE)
```

Si aparecen problemas mirar stackoverflow - Problems installing the devtools package.

Instalar el resto de paquetes como se muestra en la sección de Windows.

A.2.3 Ayuda html

Si queremos la ayuda html (en un entorno gráfico con un navegador web instalado):

```
echo "options(help_type='html')" | sudo tee -a /etc/R/Rprofile.site
```

A.2.4 Actualizar R

```
sudo apt-get update
sudo apt-get upgrade
```

A.2.5 Instalacion de RStudio Desktop

Antes de nada, puede ser recomendable crear un directorio donde descargar el instalador:

```
mkdir installr
cd installr
```

Buscar la versión actualizada de RStudio en Download RStudio correspondiente a la versión de Ubuntu, en este caso emplearemos "https://download1.rstudio.org/electron/bionic/amd64/rstudio-2022.12.0-353-amd64.deb", para Ubuntu 18+/Debian 10.

```
sudo apt-get install gdebi-core
wget https://download1.rstudio.org/electron/bionic/amd64/rstudio-2022.12.0-353-amd64.deb
sudo gdebi -n rstudio-2022.12.0-353-amd64.deb
```

Al igual que como se mostró para el caso de Windows, nos puede interesar modificar la configuración de de RStudio para mostrar los gráficos en ventanas independientes y las páginas web en el navegador del equipo. En este caso se procedería de forma idéntica, modificando los archivos de configuración en */Applications/RStudio.app/Contents/Resources/R*, editándolos como administrador.

A.3 Instalación en Mac OS X

Instalar R de http://cran.es.r-project.org/bin/macosx siguiendo los pasos habituales.

Para instalar R-Commander (https://socialsciences.mcmaster.ca/jfox/Misc/Rcmdr/installation-notes.html) es necesario disponer de las librerías gráficas X11, como a partir de OS X Lion ya no están instaladas por defecto en el sistema, hay que instalar las librerías Open Source XQuartz https://www.xquartz.org.

Finalmente, para instalar `Rcmdr` ejecutar en la consola de R:

```
install.packages("Rcmdr", dependencies = TRUE)
```

Apéndice B

Manipulación de datos con dplyr

B.1 El paquete dplyr

```
library(dplyr)
```

dplyr Permite sustituir funciones base de R (como `split()`, `subset()`, `apply()`, `sapply()`, `lapply()`, `tapply()` y `aggregate()`) mediante una "gramática" más sencilla para la manipulación de datos:

- `select()` seleccionar variables/columnas (también `rename()`).
- `mutate()` crear variables/columnas (también `transmute()`).
- `filter()` seleccionar casos/filas (también `slice()`).
- `arrange()` ordenar o organizar casos/filas.
- `summarise()` resumir valores.
- `group_by()` permite operaciones por grupo empleando el concepto "dividir-aplicar-combinar" (`ungroup()` elimina el agrupamiento).

Puede trabajar con conjuntos de datos en distintos formatos:

- `data.frame`, `data.table`, `tibble`, ...
- bases de datos relacionales (lenguaje SQL), ...
- bases de datos *Hadoop* (paquete `plyrmr`).

En lugar de operar sobre vectores como las funciones base, opera sobre objetos de este tipo (solo nos centraremos en `data.frame`).

B.1.1 Datos de ejemplo

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

```
load("datos/empleados.RData")
data.frame(Etiquetas = attr(empleados, "variable.labels"))  # Listamos las etiquetas
```

```
##                              Etiquetas
## id                   Código de empleado
## sexo                              Sexo
## fechnac             Fecha de nacimiento
## educ              Nivel educativo (años)
## catlab                 Categoría Laboral
## salario                   Salario actual
## salini                   Salario inicial
## tiempemp       Meses desde el contrato
## expprev     Experiencia previa (meses)
## minoria           Clasificación étnica
```

```
## sexoraza Clasificación por sexo y raza
attr(empleados, "variable.labels") <- NULL                    # Eliminamos las etiquetas para que no m
```

B.2 Operaciones con variables (columnas)

B.2.1 Seleccionar variables con select()

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

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

Se puede cambiar el nombre (ver también *?rename()*)

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

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

Se pueden emplear los nombres de variables como índices:

```
head(select(empleados, sexo:salario))
```

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

```
head(select(empleados, -(sexo:salario)))
```

```
##   id salini tiempemp expprev minoria     sexoraza
## 1  1  27000       98     144      No Blanca varón
## 2  2  18750       98      36      No Blanca varón
## 3  3  12000       98     381      No Blanca mujer
## 4  4  13200       98     190      No Blanca mujer
## 5  5  21000       98     138      No Blanca varón
## 6  6  13500       98      67      No Blanca varón
```

Hay opciones para considerar distintos criterios: `starts_with()`, `ends_with()`, `contains()`, `matches()`, `one_of()` (ver *?select*).

```
head(select(empleados, starts_with("s")))
```

```
##     sexo salario salini     sexoraza
## 1 Hombre   57000  27000 Blanca varón
## 2 Hombre   40200  18750 Blanca varón
```

```
## 3  Mujer   21450  12000 Blanca mujer
## 4  Mujer   21900  13200 Blanca mujer
## 5 Hombre   45000  21000 Blanca varón
## 6 Hombre   32100  13500 Blanca varón
```

B.2.2 Generar nuevas variables con mutate()

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

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

B.3 Operaciones con casos (filas)

B.3.1 Seleccionar casos con filter()

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

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

B.3.2 Organizar casos con arrange()

```
head(arrange(emplea2, salario))
```

```
##    id  sexo minoria tiempemp salini salario
## 1 378 Mujer      No       70  10200   15750
## 2 338 Mujer      No       74  10200   15900
## 3  90 Mujer      No       92   9750   16200
## 4 224 Mujer      No       82  10200   16200
## 5 411 Mujer      No       68  10200   16200
## 6 448 Mujer      Sí       66  10200   16350
```

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

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

B.4 Resumir valores con summarise()

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

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

B.5 Agrupar casos con group_by()

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

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

B.6 Operador *pipe* %>% (tubería, redirección)

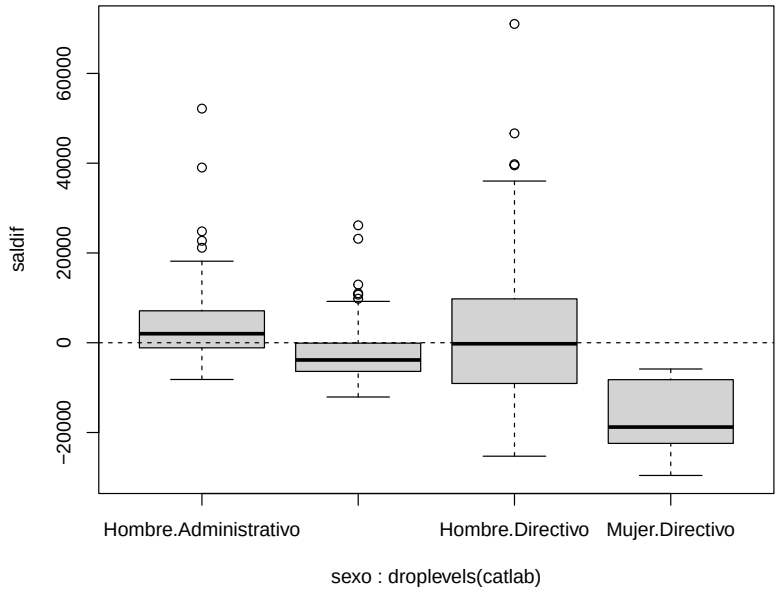
Este operador le permite canalizar la salida de una función a la entrada de otra función. `segundo(primero(datos))` se traduce en `datos %>% primero %>% segundo` (lectura de funciones de izquierda a derecha).

Ejemplos:

```
empleados %>%  filter(catlab == "Directivo") %>%
          group_by(sexo, minoria) %>%
          summarise(sal.med = mean(salario), n = n())
```

```
## # A tibble: 3 x 4
## # Groups:   sexo [2]
##   sexo   minoria sal.med     n
##   <fct>  <fct>     <dbl> <int>
## 1 Hombre No       65684.    70
## 2 Hombre Sí       76038.     4
## 3 Mujer  No       47214.    10
```

```
empleados %>% select(sexo, catlab, salario) %>%
          filter(catlab != "Seguridad") %>%
          group_by(catlab) %>%
          mutate(saldif = salario - mean(salario)) %>%
          ungroup() %>%
          boxplot(saldif ~ sexo*droplevels(catlab), data = .)
abline(h = 0, lty = 2)
```



Para mas información sobre *dplyr* ver por ejemplo la 'vignette' del paquete: Introduction to dplyr.

Apéndice C

Compañías que usan R

Cada vez son más las empresas que utilizan R.

- Grupo de empresas que apoyan a la Fundación R y a la comunidad R.

- Otras compañías:
  - Facebook, Twitter, Bank of America, Monsanto, ...

C.1 Microsoft

- Herramientas para entornos Big Data y computación de altas prestaciones.
- Versión de R con rendimiento mejorado.
  - Microsoft R Application Network:

    MRAN: https://mran.microsoft.com
- Integracion de R con: SQL Server, PowerBI, Azure y Cortana Analytics.

C.2 RStudio (Posit)

Además del entorno de desarrollo (IDE) con múltiples herramientas, descrito en la Sección 1.3:

- RStudio Server: permite ejecutar RStudio en un servidor mediante una interfaz web.
  - Evita el movimiento de datos a los clientes.
  - Ediciones Open Source y Professional (RStudio Workbench).
- Compañía muy activa en el desarrollo de R:
  - Múltiples paquetes: tidyverse (dplyr, tidyr, ggplot2, knitr, …), tidymodels, shiny, rmarkdown, …
  - Hadley Wickham (Jefe científico de RStudio).

En la Sección Enlaces de las Referencias se incluyen recursos adicionales.