

Introducción al software estadístico R

Dae-Jin Lee < dlee@bcamath.org >

Noviembre 2019

Índice general

1. Información y pre-requisitos	5
2. Introducción al software estadístico R	7
2.1. Algunas cuestiones a tener en cuenta sobre R	7
2.2. Rstudio	7
2.3. Instalar un paquete de R	9
2.4. Empezando con R	9
2.5. Cargar librerías en R	10
2.6. Lectura de datos	10
2.7. Importar datos	11
2.8. Exportar datos	12
2.9. Vectores	12
2.10. Estadística básica	13
2.11. Vectores caracteres y variables factor	14
2.12. Data frames	14
2.13. Vectores lógicos	16
2.14. Trabajando con vectores	16
2.15. Matrices y arrays	17
2.16. Factores	19
2.17. Indexando vectores con condiciones lógicas	20
2.18. Valores faltantes	20
2.19. Trabajando con data frames	21
3. Análisis de datos básico en R	25
3.1. Gráficos sencillos	25
3.2. Scatterplots	31
3.3. Más opciones gráficas	33
3.4. Tablas de clasificación cruzada o de contingencia	41
3.5. Datos cualitativos	43
3.6. Datos cuantitativos	47
4. Introducción a la programación básica con R	55
4.1. Condicionales	55
4.2. Operadores Lógicos	57

4.3. <code>if</code>	58
4.4. <code>ifelse</code>	58
4.5. Loops o Bucles	58
5. Distribuciones de probabilidad en R	61
5.1. Distribución binomial $Bin(n, p)$	62
5.2. Distribución de Poisson $Pois(\lambda)$	65
5.3. Distribution Exponencial $Exp(\lambda)$	67
5.4. Distribution Normal $\mathcal{N}(\mu, \sigma^2)$	69
5.5. Distribución Uniforme $U(a, b)$	71
6. Modelos lineales y análisis de la varianza	75
6.1. Principios de la modelización estadística	75
6.2. El modelo lineal general	76
6.3. Definición de modelos en R	79
7. Regresión logística	113
7.1. Ejemplo: Datos de crédito en Alemania (<i>Credit scoring</i>)	114
7.2. Ejemplo: Predecir el salario de los trabajadores	119
7.3. Ejemplo: Datos de los supervivientes del Titanic	124
8. Modelos Aditivos Generalizados	131
8.1. Suavizado	131
8.2. GAMs	135
8.3. Modelos semi-paramétricos	139
8.4. Ejemplo: Calidad del Aire	147
9. Análisis multivariante	163
9.1. Análisis de Componentes Principales	163
9.2. K-medias	173
9.3. Cluster jerárquico	176
10. Introducción a las redes neuronales artificiales	179
10.1. Arquitecturas de las RNA	181
10.2. Ejemplo: datos Boston housing	181
10.3. Ajuste de la red neuronal	183
10.4. Predicción con una red neuronal	184
10.5. Evaluando la predicción mediante validación cruzada	186

Capítulo 1

Información y pre-requisitos

- **Requisitos:**

- Última versión del software R (<https://www.r-project.org>)
- Rstudio (<https://www.rstudio.com>).

- **Objetivos del curso**

El curso proporcionará conocimientos básicos y habilidades para utilizar el software estadístico R como entorno de trabajo y análisis de datos. Para ello, el curso ofrecerá una visión general de las herramientas ofrecidas por R, por ejemplo, desde el manejo de matrices y conjuntos de datos, a representaciones gráficas, pruebas y análisis estadísticos, programación en R y técnicas de modelización estadística y aprendizaje automático.

El objetivo es que al final del curso todos se hayan familiarizado con el software estadístico R.

- **Material complementario (en inglés)**

- Table of useful R commands
- Base R cheatsheet
- Not so short list of R commands
- Additional material: R for Data Science or here
- Some interesting links:
 - Karl Broman's talks

- Plots to avoid
- RSeek - a Google search engine for R documentation and help.
A MUST see!

Capítulo 2

Introducción al software estadístico R

R es un entorno y lenguaje de programación orientado al análisis estadístico, al cálculo, manipulación de datos, y representaciones gráficas. Es multiplataforma y parte del sistema GNU y se distribuye con licencia GNU-GPL.

Más información aquí.

2.1. Algunas cuestiones a tener en cuenta sobre R

- R distingue mayúsculas y minúsculas.
- Para asignar contenido a un objeto usamos `<-`. Por ejemplo, `x <- 10` asigna a `x` el valor 10. También podemos usar `=`.
- Para ver el contenido de un objeto simplemente escribimos su nombre.
- Para usar los comandos escribimos el nombre del comando seguido de sus argumentos entre paréntesis. Por ejemplo, `ls()` da una lista de los objetos en el área de trabajo. Como no usamos argumentos (diferentes a los que el comando tenga por defecto) no escribimos nada en el paréntesis.
- Para obtener ayuda usamos el comando `help`. Por ejemplo, `help(mean)` para obtener ayuda sobre el comando `mean` que calcula la media.

2.2. Rstudio

RStudio es un *IDE* (*Integrated Development Environment*, o Entorno de Desarrollo Integrado) de código abierto para R, que permite interactuar con R de

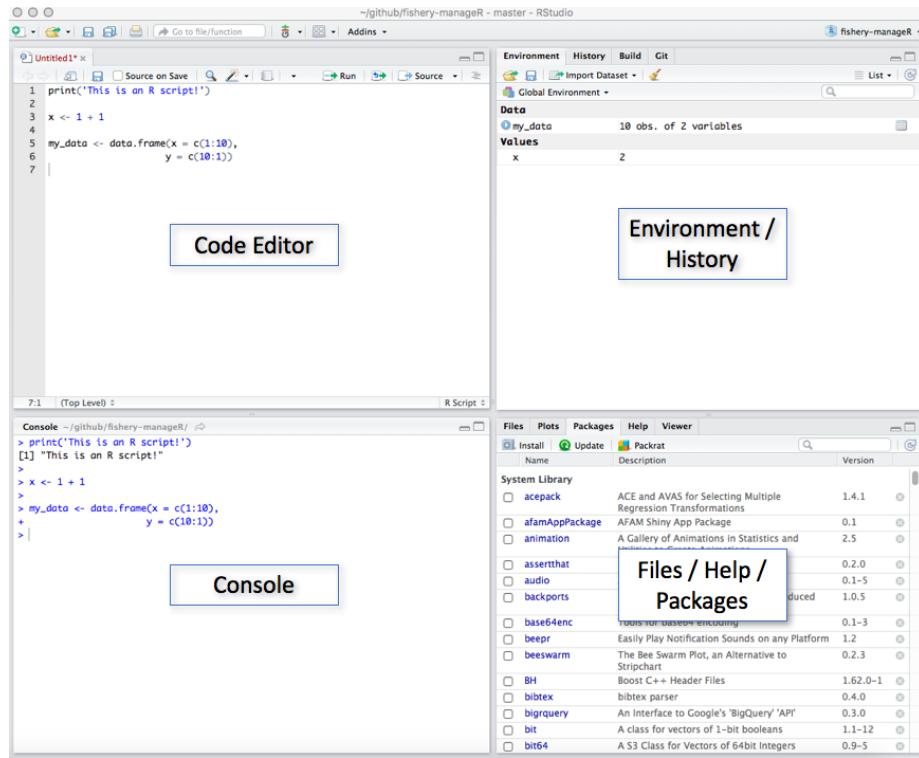


Figura 2.1: Interfaz RStudio

manera muy simple.

Entre otras ventajas, Rstudio utiliza diferentes colores para las distintas clases de objetos de R, permite autocompletar código, incluye un sistema de menús de ayuda muy completo, cuenta con un potente sistema para la gestión, descarga y construcción de librerías, dispone de un depurador de código que detecta posibles errores de sintaxis, es multiplataforma (existen versiones para Windows, Linux y Mac).

A la izquierda, la consola donde se ejecutan los comandos de R.

A la derecha, en la parte superior, tenemos una ventana que muestra nuestro entorno (environment) de trabajo, en el que iremos viendo las variables y funciones que vayamos cargando, creando, etc. Obsérvese que esta ventana tiene algunos iconos que permiten guardar el contenido de la memoria, cargar el contenido de la memoria de una sesión de trabajo anterior, importar archivos de datos que se hayan guardado como texto, y limpiar el contenido de la memoria.

A la derecha, en la parte inferior, se muestra el contenido de nuestro directorio home donde R arranca por defecto. Observemos que esta ventana tiene varias

pestañas:

- **Files:** archivos en el directorio actual.
- **Plots:** en esta ventana se irán mostrando los gráficos que generemos con el programa.
- **Packages:** permite ver qué librerías (colecciones de funciones que extienden la funcionalidad de R) tenemos instaladas; asimismo nos permite descargar e instalar nuevas librerías.
- **Help:** permite acceder a ayuda sobre 'R'.
- **Viewer:** Permite acceder a contenido web local.

2.3. Instalar un paquete de R

- Desde la consola

```
install.packages("< Nombre del paquete >")
```

- Ejemplo:

```
install.packages("DAAG")
# Varios paquetes con el comando c("< Paquete 1 >","< Paquete 2 >")
install.packages(c("DAAG","psych","gdata","foreign","Hmisc","xlsx",
                  "MASS","calibrate","corrplot","fields","RColorBrewer",
                  "ggplot2","lattice","visreg",
                  "car","InformationValue","ROCR"))
```

- Desde Rstudio: Packages > Install

2.4. Empezando con R

- Obtener el directorio de trabajo o *working directory*

```
getwd()
```

- listar los objetos en el espacio de trabajo o *workspace*

```
ls()
```

- Definir el *working directory*

```
setwd("/Users/dlee")
```

- Desde Rstudio: Files > Click en ... > More > Set as workind directory

- Ver los últimos comandos utilizados en la consola

```
history() # mostrar los últimos 25 comandos
```

```
history(max.show=Inf) # mostrar todos los comandos anteriores
```

- Guardar el historial de comandos

```
savehistory(file="myfile") # el valor por defecto es ".Rhistory"
```

- Cargar los commandos guardados en una sesión anterior

```
loadhistory(file="myfile") # el valor por defecto es ".Rhistory"
```

- Salvar todo el **workspace** en un fichero .RData

```
save.image()
```

- Guardar objetos específicos a un fichero (si no se especifica la ruta en el ordenador, se guardará en el directorio actual de trabajo).

```
save(<object list>,file="myfile.RData")
```

- Cargar un *workspace* en la sesión

```
load("myfile.RData")
```

- Salir de R. Por defecto R pregunta si deseas guardar la sesión.

```
q()
```

2.5. Cargar librerías en R

- Una vez instalada la librería, tenemos que cargarla con el comando **library** o **require**

```
library(DAAG) # o require(DAAG)
```

2.6. Lectura de datos

Consola de R

```
x <- c(7.82,8.00,7.95) # c de "combinar"
x
```

```
## [1] 7.82 8.00 7.95
```

Otra forma es mediante la función **scan()**

```
x <- scan() # introducir números seguidos de ENTER y terminar con un ENTER
1: 7.82
2: 8.00
3: 7.95
4:
Read 3 items
```

Para crear un vector de caracteres

```
id <- c("John", "Paul", "George", "Ringo")
```

To read a character vector

```
id <- scan(, "")  
1: John  
2: Paul  
3: George  
4: Ringo  
5:  
Read 4 items  
  
id  
  
## [1] "John"    "Paul"     "George"   "Ringo"
```

2.7. Importar datos

En ocasiones, necesitaremos leer datos de un fichero independiente. Existen varias formas de hacerlo:

- `scan()` (`?scan` ver la ayuda)

```
# creamos el fichero ex.txt  
cat("Example:", "2 3 5 7", "11 13 17", file = "ex.txt", sep = "\n")  
scan("ex.txt", skip = 1)  
  
## [1] 2 3 5 7  
scan("ex.txt", skip = 1, nlines = 1) # only 1 line after the skipped one  
  
## [1] 2 3 5 7  
unlink("ex.data") # tidy up
```

- Existen diferentes formatos (.txt, .csv, .xls, .xlsx, SAS, Stata, etc...)
- Algunas librerías de R para importar datos:

```
library(gdata)  
library(foreign)
```

- Generalmente leemos datos en formato .txt o .csv

Vamos a crear una carpeta que llamaremos `data` y descargaremos los datos `cardata` en el siguiente enlace.

```
mydata1 <- read.table("data/cardata.txt")  
mydata2 <- read.csv("data/cardata.csv")
```

- Otros formatos .xls and .xlsx

```
library(gdata)
mydata3 <- read.xls ("cardata/cardata.xls", sheet <- 1, header = TRUE)
```

- Minitab, SPSS, SAS or Stata

```
library(foreign)
mydata = read.mtp("mydata.mtp") # Minitab
mydata = read.spss("myfile", to.data.frame=TRUE) # SPSS
mydata = read.dta("mydata.dta") # Stata
```

- O también

```
library(Hmisc)
mydata = spss.get("mydata.por", use.value.labels=TRUE) # SPSS
```

2.8. Exportar datos

- Existen diferentes maneras de exportar datos desde R en diferentes formatos. Para SPSS, SAS y Stata. Por ejemplo, mediante la librería `foreign`. En Excel, la librería `xlsx`.
- Texto delimitado por tabulaciones:

```
mtcars
?mtcars
write.table(mtcars, "cardata.txt", sep="\t")
```

- Hoja de cálculo de Excel:

```
library(xlsx)
write.xlsx(mydata, "mydata.xlsx")
```

2.9. Vectores

- Crear dos vectores

```
weight<-c(60,72,57,90,95,72)
class(weight)
```

```
## [1] "numeric"
height<-c(1.75,1.80,1.65,1.90,1.74,1.91)
```

- calcular el Body Mass Index (*índice de masa corporal*)

```
bmi<- weight/height^2
bmi
```

```
## [1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.73630
```

2.10. Estadística básica

- mean, median, st dev, variance

```
mean(weight)
median(weight)
sd(weight)
var(weight)
```

- Resumen de un vector

```
summary(weight)
```

```
##      Min. 1st Qu. Median     Mean 3rd Qu.    Max.
## 57.00   63.00   72.00   74.33   85.50   95.00
```

- o también

```
min(weight)
max(weight)
range(weight)
sum(weight)
length(weight)
```

- Cuantiles y percentiles

```
quantile(weight) # por defecto cuantil 25%, 50% y 75%
```

```
## 0% 25% 50% 75% 100%
## 57.0 63.0 72.0 85.5 95.0
```

```
quantile(weight,c(0.32,0.57,0.98))
```

```
## 32% 57% 98%
## 67.2 72.0 94.5
```

- Covarianza y correlación

La covarianza (σ_{xy}) indica el grado de variacion conjunta de dos variables aleatorias respecto a sus medias

- Si $\sigma_{xy} > 0$, hay dependencia directa (positiva), es decir, a grandes valores de x corresponden grandes valores de y .
- Si $\sigma_{xy} = 0$, hay una covarianza 0 se interpreta como la no existencia de una relación lineal entre las dos variables estudiadas.
- Si $\sigma_{xy} < 0$ hay dependencia inversa o negativa, es decir, a grandes valores de x corresponden pequeños valores de y .

$$\text{Cov}(x, y) = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$$

```
cov(weight,height)
```

```
## [1] 0.6773333
```

El *coeficiente de correlación* mide la relación lineal (positiva o negativa) entre dos variables. Formalmente es el cociente entre la covarianza y el producto de las desviaciones típicas de ambas variables. Siendo σ_x y σ_y las desviaciones estandar y σ_{xy} la covarianza entre x e y .

$$\rho_{xy} = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

```
cor(weight,height)
```

```
## [1] 0.437934
```

2.11. Vectores caracteres y variables factor

```
subject <- c("John","Peter","Chris","Tony","Mary","Jane")
sex <- c("MALE","MALE","MALE","MALE","FEMALE","FEMALE")
class(subject)

## [1] "character"

table(sex)

## sex
## FEMALE    MALE
##      2        4
```

2.12. Data frames

```
Dat <- data.frame(subject,sex,weight,height)
# añadir el bmi a Dat
Dat$bmi <- bmi  # o Dat$bmi <- weight/height^2
class(Dat)

## [1] "data.frame"

str(Dat) # Ver la estructura del data.frame

## 'data.frame':   6 obs. of  5 variables:
## $ subject: Factor w/ 6 levels "Chris","Jane",...: 3 5 1 6 4 2
## $ sex     : Factor w/ 2 levels "FEMALE","MALE": 2 2 2 2 1 1
## $ weight  : num  60 72 57 90 95 72
## $ height : num  1.75 1.8 1.65 1.9 1.74 1.91
```

```

## $ bmi      : num  19.6 22.2 20.9 24.9 31.4 ...
# cambiar el nombre de las filas
rownames(Dat)<-c("A","B","C","D","E","F")

# Acceder a los elementos del data.frame
Dat[,1]      # columna 1

## [1] John Peter Chris Tony Mary Jane
## Levels: Chris Jane John Mary Peter Tony

Dat[,1:3]    # columnas 1 a 3

##   subject sex weight
## A     John  MALE    60
## B    Peter  MALE    72
## C   Chris  MALE    57
## D    Tony  MALE    90
## E   Mary FEMALE   95
## F   Jane FEMALE   72

Dat[1:2,]    # filas 1 a 2

##   subject sex weight height      bmi
## A     John MALE    60    1.75 19.59184
## B    Peter MALE    72    1.80 22.22222

```

2.12.1. Trabajando con data frames

Ejemplo: analizar datos por grupos

- Obtener el peso (`weight`), altura (`height`) y `bmi` por FEMALES y MALES:

1. Seleccionado cada grupo y calculando la media por grupos

```

Dat[sex=="MALE",]
Dat[sex=="FEMALE",]

mean(Dat[sex=="MALE",3])  # weight average of MALES
mean(Dat[sex=="MALE","weight"])

```

2. Mediante la función `apply` por columnas

```

apply(Dat[sex=="FEMALE",3:5],2,mean)
apply(Dat[sex=="MALE",3:5],2,mean)

# podemos utilizar la función apply con cualquier función
apply(Dat[sex=="FEMALE",3:5],2,function(x){x+2})

```

3. función `by` o `colMeans`

```
# 'by' divide los datos en factores y realiza
#   los cálculos para cada grupo
by(Dat[,3:5],sex, colMeans)
```

4. función aggregate

```
# otra opción
aggregate(Dat[,3:5], by=list(sex),mean)
```

2.13. Vectores lógicos

- Elegir los individuos con BMI>22

```
bmi
bmi>22
as.numeric(bmi>22) # convierte a numerico 0/1
which(bmi>22) # nos devuelve la posicion del valor donde bmi>22
```

- ¿Qué valores están entre 20 y 25?

```
bmi > 20 & bmi < 25
which(bmi > 20 & bmi < 25)
```

2.14. Trabajando con vectores

- Concatenar

```
x <- c(2, 3, 5, 2, 7, 1)
y <- c(10, 15, 12)
z <- c(x,y) # concatena x e y
```

- Lista de 2 vectores

```
zz <- list(x,y) # crea una lista
unlist(zz) # deshace la lista convirtiéndola en un vector concatenado
```

```
## [1] 2 3 5 2 7 1 10 15 12
```

- Subconjunto de vectores

```
x[c(1,3,4)]
```

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

```
x[-c(2,6)] # simbolo - omite los elementos
```

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

- Secuencias

```

seq(1,9) # δ 1:9
## [1] 1 2 3 4 5 6 7 8 9
seq(1,9,by=1)

## [1] 1 2 3 4 5 6 7 8 9
seq(1,9,by=0.5)

## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5 9.0
seq(1,9,length=20)

## [1] 1.000000 1.421053 1.842105 2.263158 2.684211 3.105263 3.526316
## [8] 3.947368 4.368421 4.789474 5.210526 5.631579 6.052632 6.473684
## [15] 6.894737 7.315789 7.736842 8.157895 8.578947 9.000000

■ Rélicas

oops <- c(7,9,13)
rep(oops,3) # repite el vector "oops" 3 veces
rep(oops,1:3) # repite cada elemento del vector las veces indicadas

rep(c(2,3,5), 4)
rep(1:2,c(10,15))

rep(c("MALE","FEMALE"),c(4,2)) # también funciona con caracteres
c(rep("MALE",3), rep("FEMALE",2))

```

2.15. Matrices y arrays

```

x<- 1:12
x

## [1] 1 2 3 4 5 6 7 8 9 10 11 12
dim(x)<-c(3,4) # 3 filas y 4 columnas

X <- matrix(1:12,nrow=3,byrow=TRUE)
X

##      [,1] [,2] [,3] [,4]
## [1,]     1     2     3     4
## [2,]     5     6     7     8
## [3,]     9    10    11    12
X <- matrix(1:12,nrow=3,byrow=FALSE)
X

```

```

##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
# rownames, colnames

rownames(X) <- c("A", "B", "C")
X

##      [,1] [,2] [,3] [,4]
## A     1    4    7   10
## B     2    5    8   11
## C     3    6    9   12
colnames(X) <- LETTERS[4:7]
X

## D E F G
## A 1 4 7 10
## B 2 5 8 11
## C 3 6 9 12
colnames(X) <- month.abb[4:7]
X

## Apr May Jun Jul
## A   1    4    7   10
## B   2    5    8   11
## C   3    6    9   12

```

- Concatenar filas y columnas `rbind()`, `cbind()`

```

Y <- matrix(0.1*(1:12), 3, 4)

cbind(X,Y) # bind column-wise

```

```

##      Apr May Jun Jul
## A   1    4    7   10 0.1 0.4 0.7 1.0
## B   2    5    8   11 0.2 0.5 0.8 1.1
## C   3    6    9   12 0.3 0.6 0.9 1.2

```

```

rbind(X,Y) # bind row-wise

```

```

##      Apr May Jun Jul
## A 1.0 4.0 7.0 10.0
## B 2.0 5.0 8.0 11.0
## C 3.0 6.0 9.0 12.0
## 0.1 0.4 0.7 1.0
## 0.2 0.5 0.8 1.1

```

```
##  0.3 0.6 0.9 1.2
```

2.16. Factores

```
gender<-c(rep("female",691),rep("male",692))
class(gender)

## [1] "character"
# cambiar vector a factor (por ejemplo a una categoria)
gender<- factor(gender)
levels(gender)

## [1] "female" "male"
summary(gender)

## female    male
##   691     692
table(gender)

## gender
## female    male
##   691     692

status<- c(0,3,2,1,4,5)      # Crear vector numerico,
                           # transformarlo a niveles.
fstatus <- factor(status, levels=0:5)
levels(fstatus) <- c("student","engineer",
                     "unemployed","lawyer","economist","dentist")

Dat$status <- fstatus
Dat

##   subject sex weight height     bmi status
## A    John  MALE     60  1.75 19.59184 student
## B   Peter  MALE     72  1.80 22.22222 lawyer
## C   Chris  MALE     57  1.65 20.93664 unemployed
## D    Tony  MALE     90  1.90 24.93075 engineer
## E   Mary FEMALE    95  1.74 31.37799 economist
## F   Jane FEMALE    72  1.91 19.73630 dentist
```

2.17. Indexando vectores con condiciones lógicas

```
a <- c(1,2,3,4,5)
b <- c(TRUE,FALSE,FALSE,TRUE,FALSE)

max(a[b])

## [1] 4
sum(a[b])

## [1] 5
```

2.18. Valores faltantes

En R, los valores faltante (o *missing values*) se representan como NA (*not available*). Los valores imposibles (e.g., valores divididos por cero) se representan con el simbolo NaN (*not a number*).

```
a <- c(1,2,3,4,NA)
sum(a)
```

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

El argumento na.rm=TRUE excluye los valores NA en el cálculo de algunos valores

```
sum(a,na.rm=TRUE)
```

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

```
a <- c(1,2,3,4,NA)
is.na(a) # YES or NO
```

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

La función complete.cases() devuelve un vector lógico que indica los casos completos.

```
complete.cases(a)
```

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

La función na.omit() devuelve un objeto sin los elementos NA.

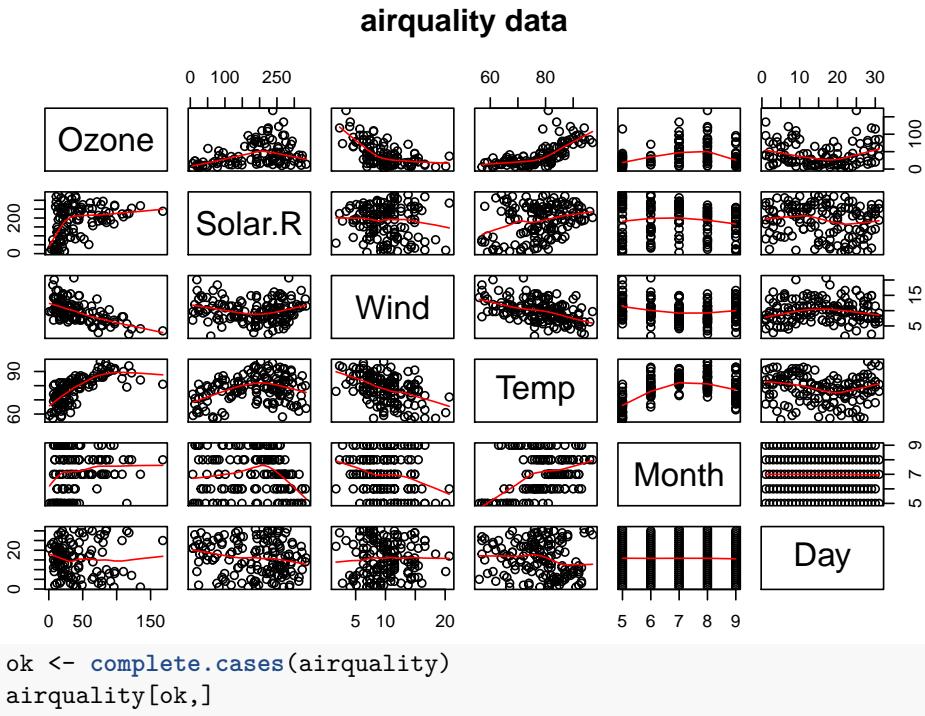
```
na.omit(a)
```

```
## [1] 1 2 3 4
## attr(),"na.action")
## [1] 5
## attr(),"class")
```

```
## [1] "omit"
```

NA en data frames:

```
require(graphics)
?airquality
pairs(airquality, panel = panel.smooth, main = "airquality data")
```



```
ok <- complete.cases(airquality)
airquality[ok,]
```

2.19. Trabajando con data frames

- Los data frame se utilizan para guardar tablas de datos. Contiene elementos de la misma longitud.

```
mtcars
?mtcars      # help(mtcars)
```

- Observemos las primeras filas

```
head(mtcars)
```

```
##          mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4     21.0   6 160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21.0   6 160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710    22.8   4 108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive 21.4   6 258 110 3.08 3.215 19.44  1  0    3    1
```

```
## Hornet Sportabout 18.7   8   360 175 3.15 3.440 17.02  0   0     3   2
## Valiant          18.1   6   225 105 2.76 3.460 20.22  1   0     3   1
```

- Estructura de un data frame

```
str(mtcars) # visualiza la estructura del marco de datos
```

```
## 'data.frame':   32 obs. of  11 variables:
##   $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##   $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##   $ disp: num  160 160 108 258 360 ...
##   $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##   $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##   $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##   $ qsec: num  16.5 17 18.6 19.4 17 ...
##   $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##   $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##   $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##   $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

- Select a car model:

```
mtcars["Mazda RX4",] # usando nombres de las filas y las columnas
```

```
##           mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4  21   6 160 110 3.9 2.62 16.46  0  1    4    4
mtcars[c("Datsun 710", "Camaro Z28"),]
```

```
##           mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Datsun 710 22.8   4 108 93 3.85 2.32 18.61  1  1    4    1
## Camaro Z28 13.3   8 350 245 3.73 3.84 15.41  0  0    3    4
```

- O variables concretas

```
mtcars[,c("mpg", "am")]
```

```
##           mpg am
## Mazda RX4 21.0  1
## Mazda RX4 Wag 21.0  1
## Datsun 710 22.8  1
## Hornet 4 Drive 21.4  0
## Hornet Sportabout 18.7  0
## Valiant 18.1  0
## Duster 360 14.3  0
## Merc 240D 24.4  0
## Merc 230 22.8  0
## Merc 280 19.2  0
## Merc 280C 17.8  0
## Merc 450SE 16.4  0
```

```
## Merc 450SL      17.3  0
## Merc 450SLC     15.2  0
## Cadillac Fleetwood 10.4  0
## Lincoln Continental 10.4  0
## Chrysler Imperial 14.7  0
## Fiat 128        32.4  1
## Honda Civic       30.4  1
## Toyota Corolla    33.9  1
## Toyota Corona     21.5  0
## Dodge Challenger   15.5  0
## AMC Javelin       15.2  0
## Camaro Z28        13.3  0
## Pontiac Firebird   19.2  0
## Fiat X1-9         27.3  1
## Porsche 914-2      26.0  1
## Lotus Europa       30.4  1
## Ford Pantera L     15.8  1
## Ferrari Dino       19.7  1
## Maserati Bora      15.0  1
## Volvo 142E         21.4  1

library(psych)
describe(mtcars)

##                                     25%          50%
## 352.00000 39.60853 84.20792 0.00000 2.42875 4.00000
## 75%
## 18.97500 472.00000 -341.00000
```


Capítulo 3

Análisis de datos básico en R

3.1. Gráficos sencillos

- Scatterplot

```
attach(mtcars)

## The following object is masked from Auto (pos = 3):
##
##      mpg

## The following objects are masked from mtcars (pos = 11):
##
##      am, carb, cyl, disp, drat, gear, hp, mpg, qsec, vs, wt

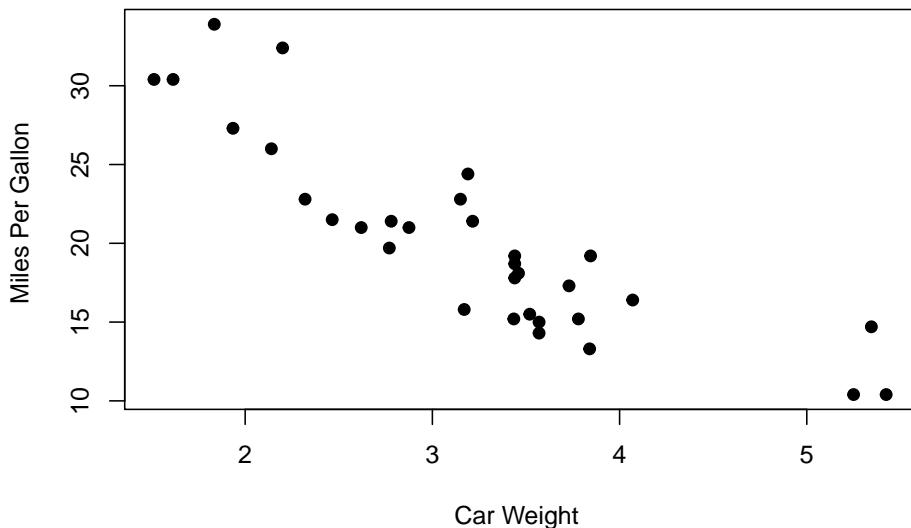
## The following object is masked from Auto (pos = 18):
##
##      mpg

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

## The following objects are masked from mtcars (pos = 48):
##
##      am, carb, cyl, disp, drat, gear, hp, mpg, qsec, vs, wt

plot(wt, mpg, main="Scatterplot Example",
      xlab="Car Weight ", ylab="Miles Per Gallon ", pch=19)
```

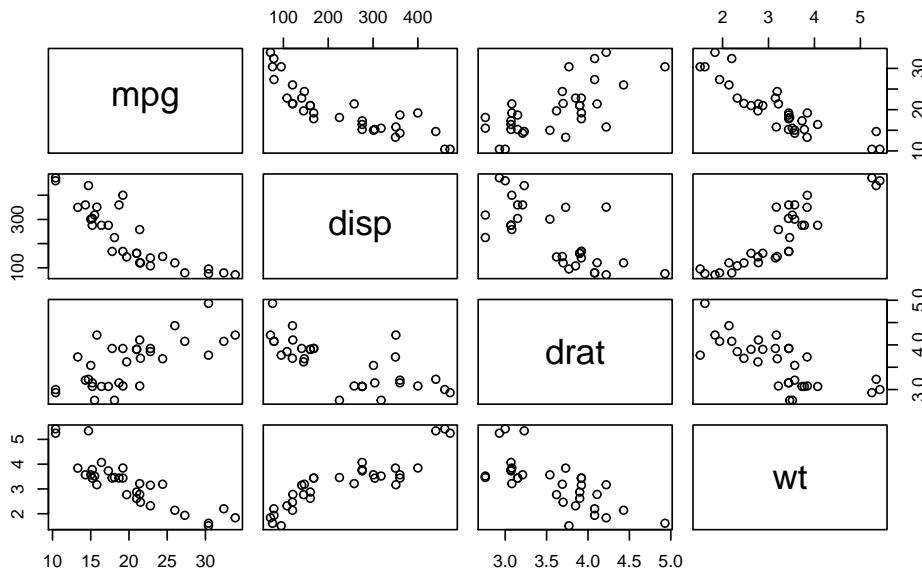
Scatterplot Example



- Matriz scatterplot

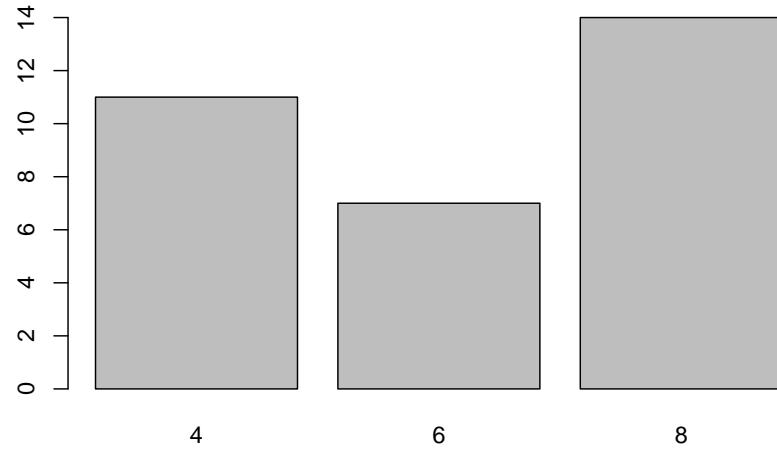
```
pairs(~mpg+disp+drat+wt, data=mtcars,
      main="Simple Scatterplot Matrix")
```

Simple Scatterplot Matrix



- Barplot o diagrama de barras

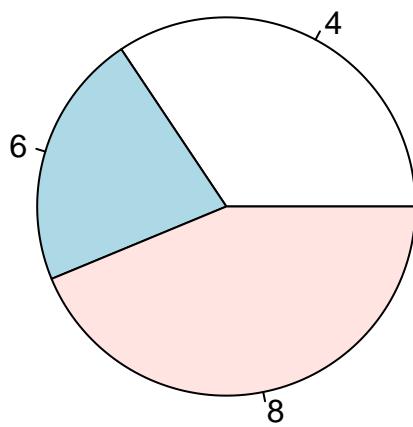
```
tab <- table(mtcars[,c("cyl")])
barplot(tab)
```



-1.bb

- Piechart o diagrama de tarta

```
pie(tab)
```



Ejercicio:

1. El `data.frame VADeaths` contiene las tasas de mortalidad por cada 1000 habitantes en Virginia (EEUU) en 1940
 - Las tasas de mortalidad se miden cada 1000 habitantes por año. Se encuen-

tran clasificadas por grupo de edad (filas) y grupo de población (columnas).

Los grupos de edad son: 50-54, 55-59, 60-64, 65-69, 70-74 y los grupos de población: Rural/Male, Rural/Female, Urban/Male and Urban/Female.

```
data(VADeaths)
VADeaths
```

	Rural	Male	Rural	Female	Urban	Male	Urban	Female
## 50-54	11.7		8.7		15.4		8.4	
## 55-59	18.1		11.7		24.3		13.6	
## 60-64	26.9		20.3		37.0		19.3	
## 65-69	41.0		30.9		54.6		35.1	
## 70-74	66.0		54.3		71.1		50.0	

- Calcula la media para cada grupo de edad.

- **Result:**

```
## 50-54 55-59 60-64 65-69 70-74
## 11.050 16.925 25.875 40.400 60.350
```

- Calcula la media para cada grupo de población.

- **Resultado:**

```
##   Rural Male Rural Female   Urban Male Urban Female
##      32.74        25.18       40.48       25.28
```

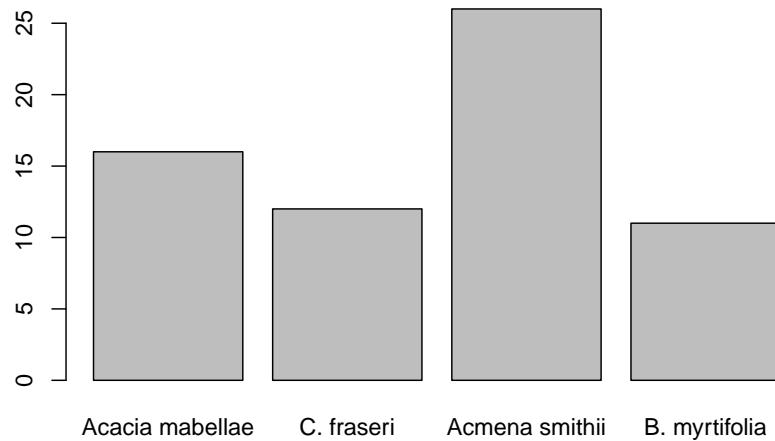
2. El `data.frame rainforest` contiene diferentes variables de `species`

```
library(DAAG)
rainforest
?rainforest
names(rainforest)
```

- Crear una tabla de conteos para cada `species` y realiza un gráfico descriptivo.

- **Resultado:**

```
##
## Acacia mabellae      C. fraseri  Acmena smithii    B. myrtifolia
##          16                  12                  26                  11
```

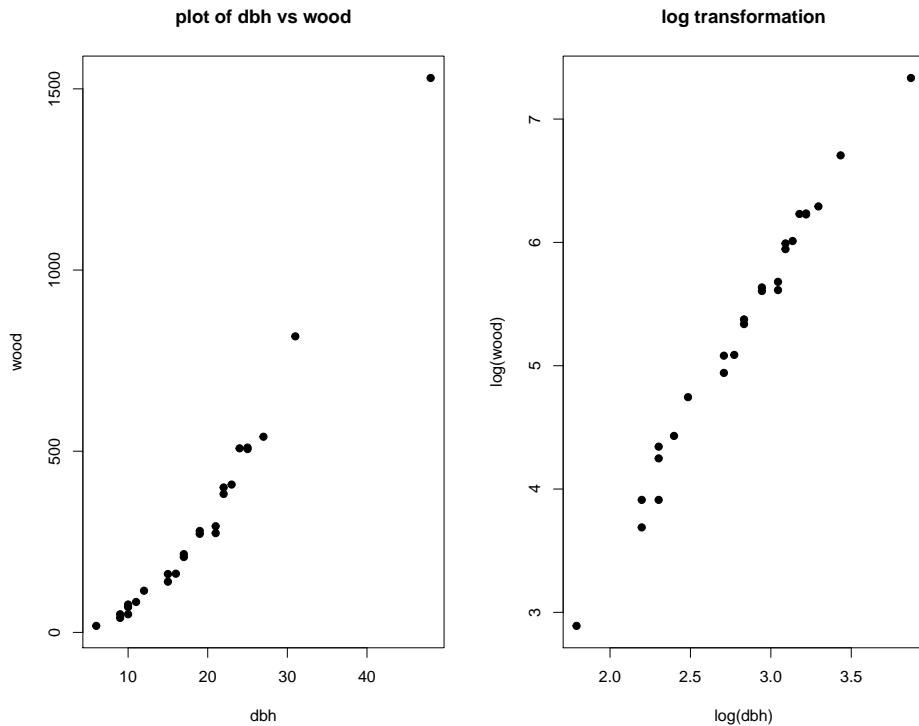


-1.bb

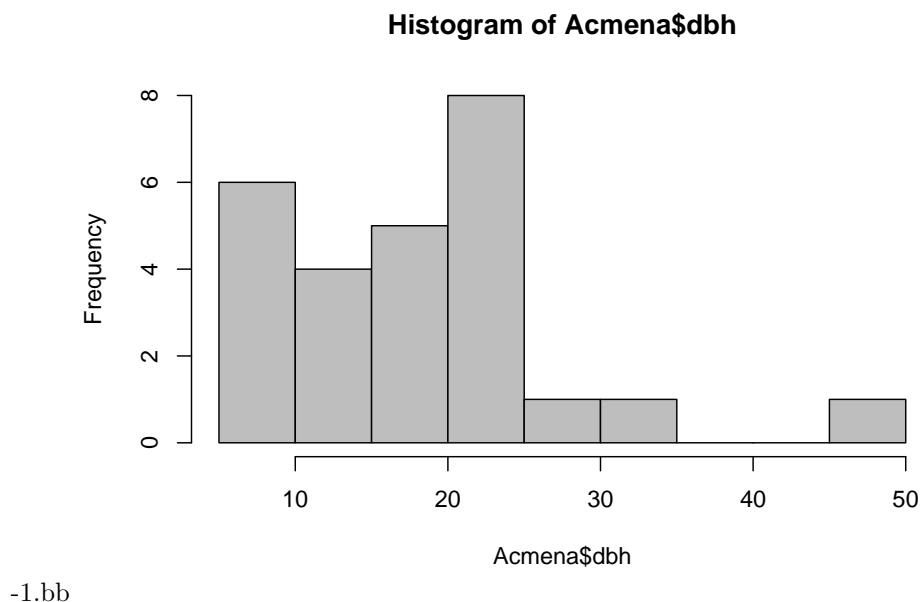
3. El `data.frame Acmena` est? creado a partir de `rainforest` mediante la funci?n `subset`.

- Realiza un gráfico que relacione la biomasa de la madera (`wood`) y el di?metro a la altura del pecho (`dbh`). Utiliza tambi?n la escala logar?tmica.

```
Acmena <- subset(rainforest, species == "Acmena smithii")
```



- Calcula un histograma de la variable dbh mediante la función `hist`



-1.bb

4. Crea un vector de números enteros positivos impares the longitud 100 y calcula los valores entre 60 y 80.

▪ **Result:**

```
## [1] 61 63 65 67 69 71 73 75 77 79
```

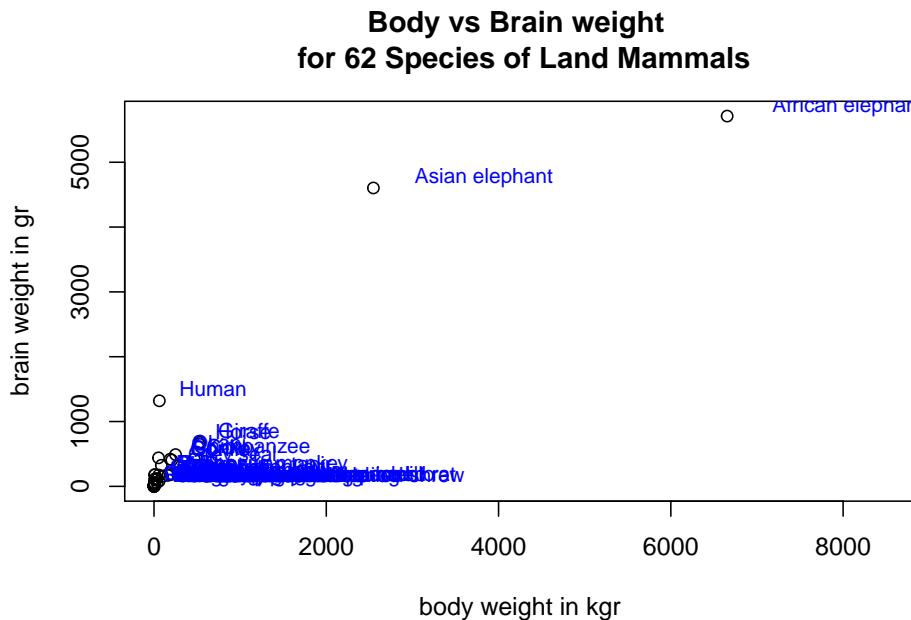
3.2 Scatterplots

```
library(MASS)
data("mammals")
?mammals
head(mammals)

##           body brain
## Arctic fox      3.385 44.5
## Owl monkey     0.480 15.5
## Mountain beaver 1.350  8.1
## Cow            465.000 423.0
## Grey wolf       36.330 119.5
## Goat            27.660 115.0

attach(mammals)
species <- row.names(mammals)
x <- body
y <- brain

library(calibrate)
# scatterplot
plot(x,y, xlab = "body weight in kgr", ylab = "brain weight in gr",
      main="Body vs Brain weight \n for 62 Species of Land Mammals", xlim=c(0,8500))
textxy(x,y,labs=species,col = "blue",cex=0.85)
```

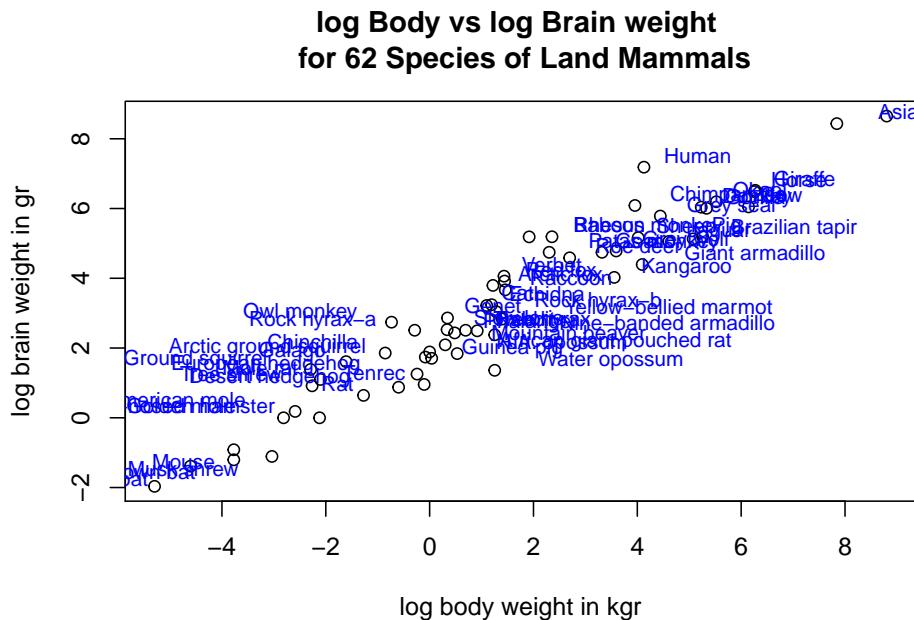


Identificar un punto en el scatterplot

```
identify(x,y,species)
```

En escala logarítmica

```
plot(log(x),log(y), xlab = "log body weight in kgr", ylab = "log brain weight in gr",
     main="log Body vs log Brain weight \n for 62 Species of Land Mammals")
textxy(log(x),log(y),labs=species,col = "blue",cex=0.85)
```



Identificar un punto en la escala logarítmica

```
identify(log(x),log(y),species)
```

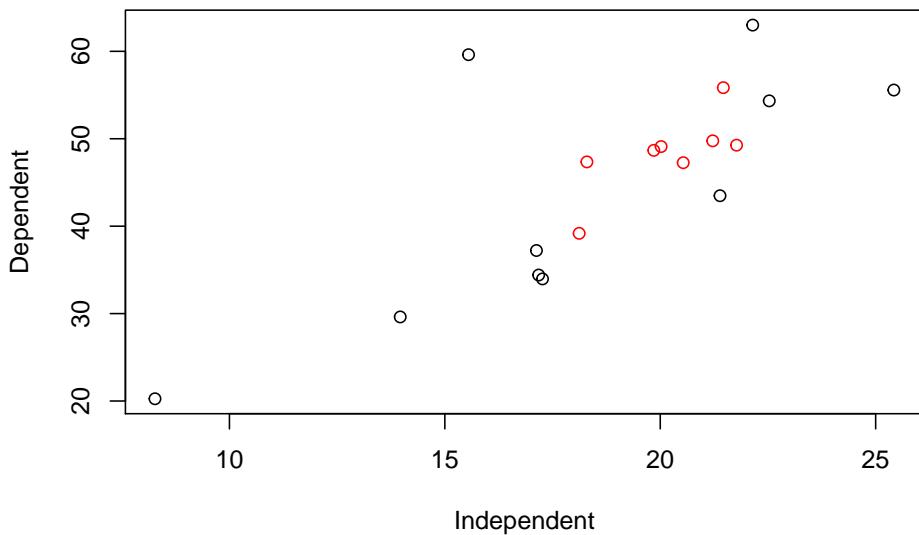
3.3. Más opciones gráficas

Varios conjuntos de datos en un sólo gráfico

Una vez realizado un `plot`, el comando `points` permite añadir nuevas observaciones.

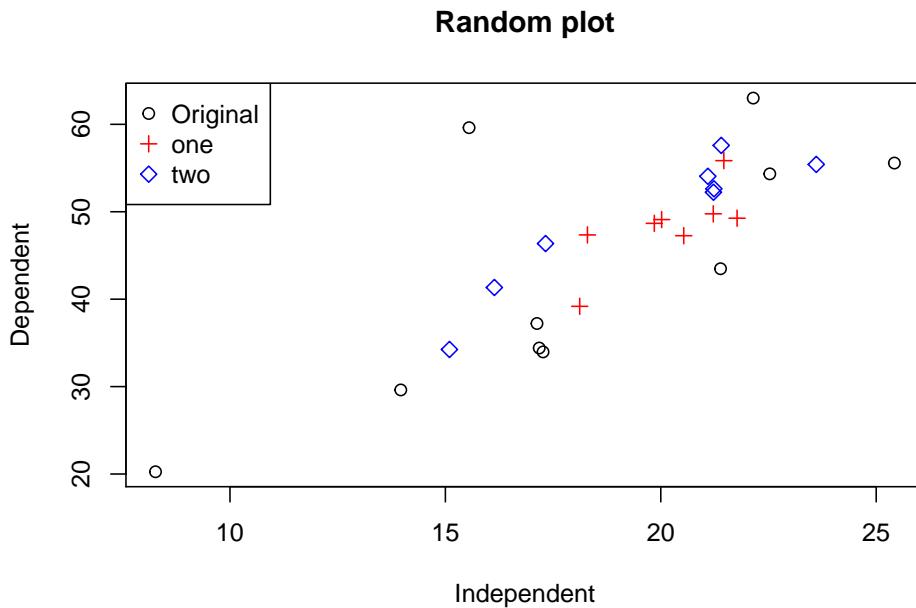
```
set.seed(1234)
x <- rnorm(10, sd=5, mean=20)
y <- 2.5*x - 1.0 + rnorm(10, sd=9, mean=0)
cor(x,y)

## [1] 0.7512194
plot(x,y,xlab="Independent",ylab="Dependent",main="Random plot")
x1 <- runif(8,15,25)
y1 <- 2.5*x1 - 1.0 + runif(8,-6,6)
points(x1,y1,col=2)
```

Random plot

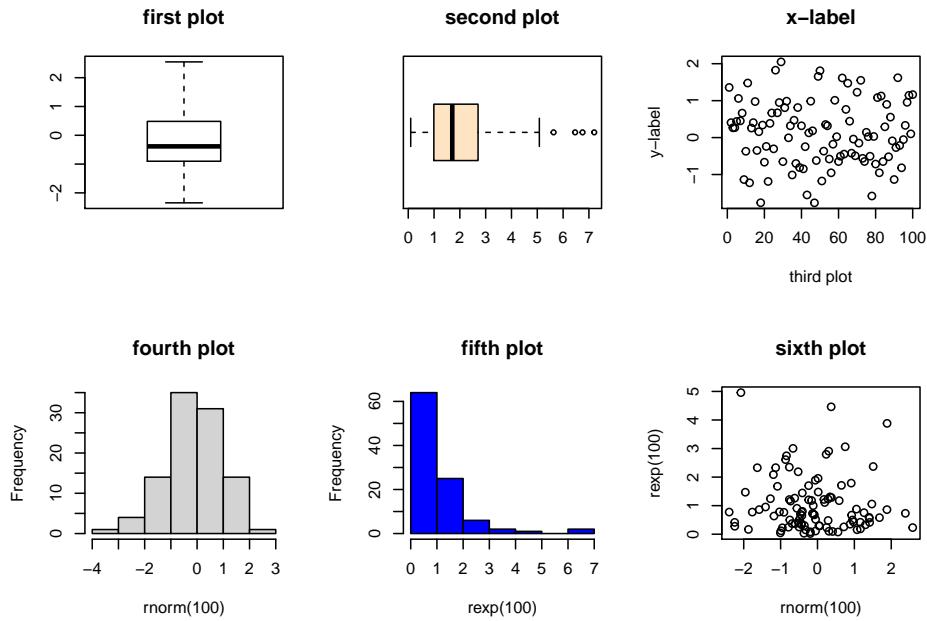
con la leyenda

```
set.seed(1234)
x2 <- runif(8,15,25)
y2 <- 2.5*x2 - 1.0 + runif(8,-6,6)
plot(x,y,xlab="Independent",ylab="Dependent",main="Random plot")
points(x1,y1,col=2,pch=3)
points(x2,y2,col=4,pch=5)
legend("topleft",c("Original","one","two"),col=c(1,2,4),pch=c(1,3,5))
```



Varios gráficos en un sola imagen

```
set.seed(1234)
par(mfrow=c(2,3))
boxplot(rnorm(100),main="first plot")
boxplot(rgamma(100,2),main="second plot", horizontal=TRUE,col="bisque")
plot(rnorm(100),xlab="third plot",
      ylab="y-label",main="x-label")
hist(rnorm(100),main="fourth plot",col="lightgrey")
hist(rexp(100),main="fifth plot",col="blue")
plot(rnorm(100),rexp(100),main="sixth plot")
```



Relaciones entre variables

```

uData <- rnorm(20)
vData <- rnorm(20,mean=5)
wData <- uData + 2*vData + rnorm(20,sd=0.5)
xData <- -2*uData+rnorm(20,sd=0.1)
yData <- 3*vData+rnorm(20,sd=2.5)
d <- data.frame(u=uData,v=vData,w=wData,x=xData,y=yData)
pairs(d)

```

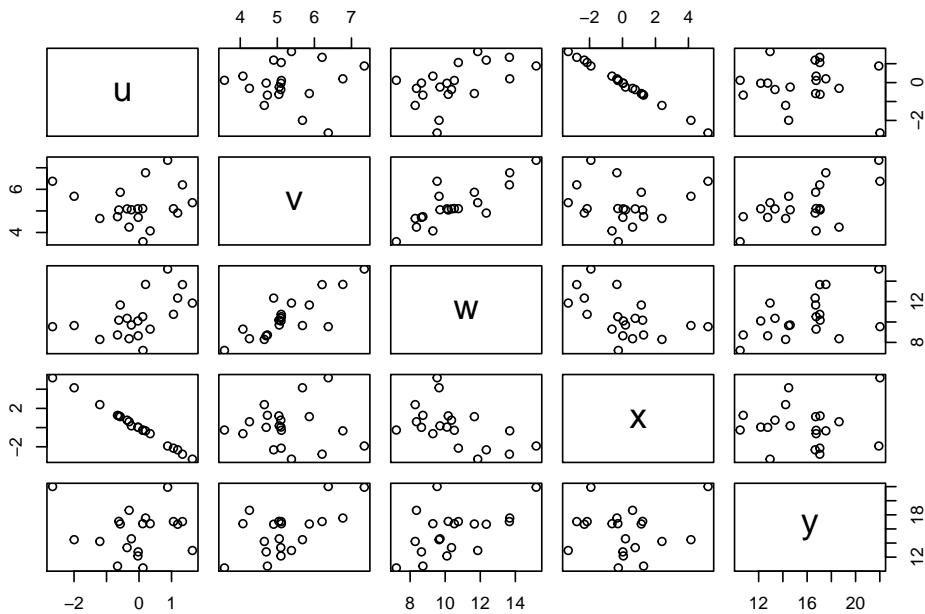
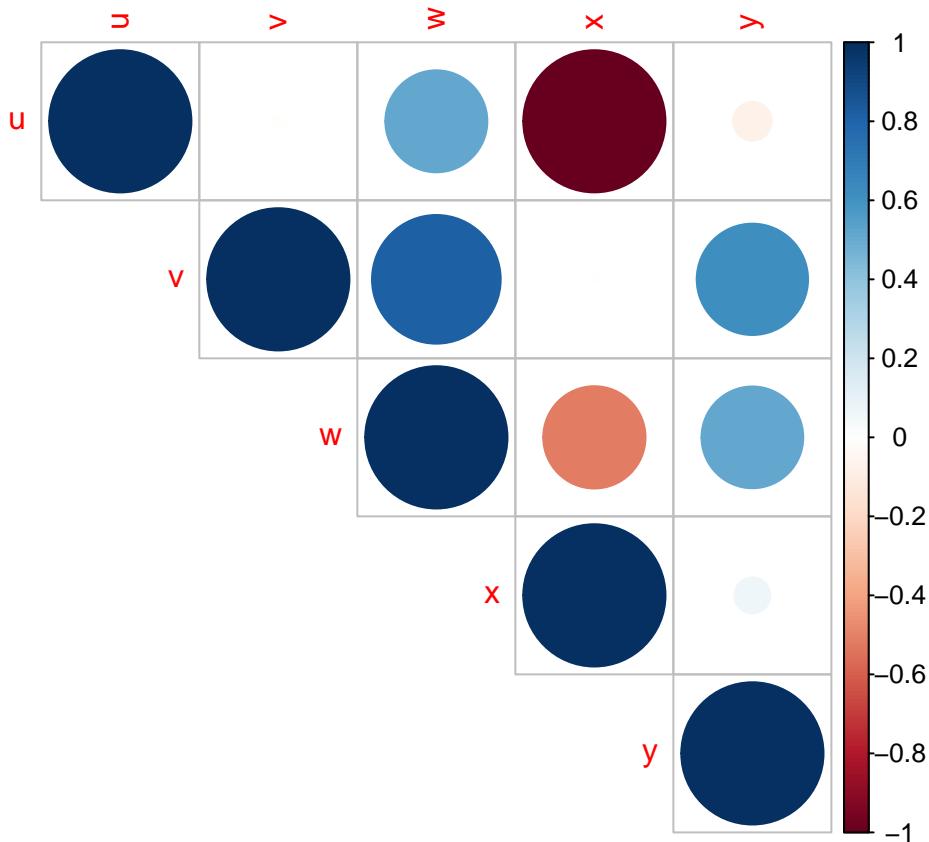


Gráfico de correlaciones

La función `corrplot` de la librería `corrplot` permite visualizar una matriz de correlaciones calculada mediante la función `cor`

```
library(corrplot)
M <- cor(d)
corrplot(M, method="circle", type="upper")
```



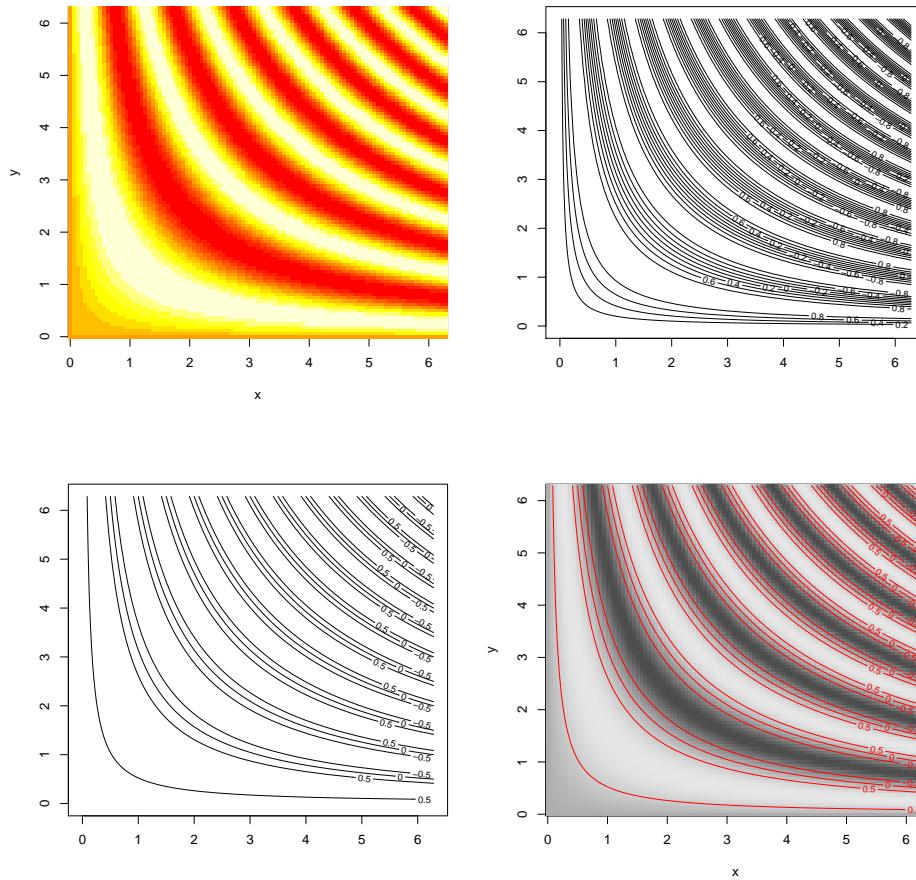
Gráficos de superficies: `image`, `contour` y `persp`

```

x <- seq(0,2*pi,by=pi/50)
y <- x
xg <- (x*0+1) %*% t(y)
yg <- (x) %*% t(y*0+1)
f <- sin(xg*yg)

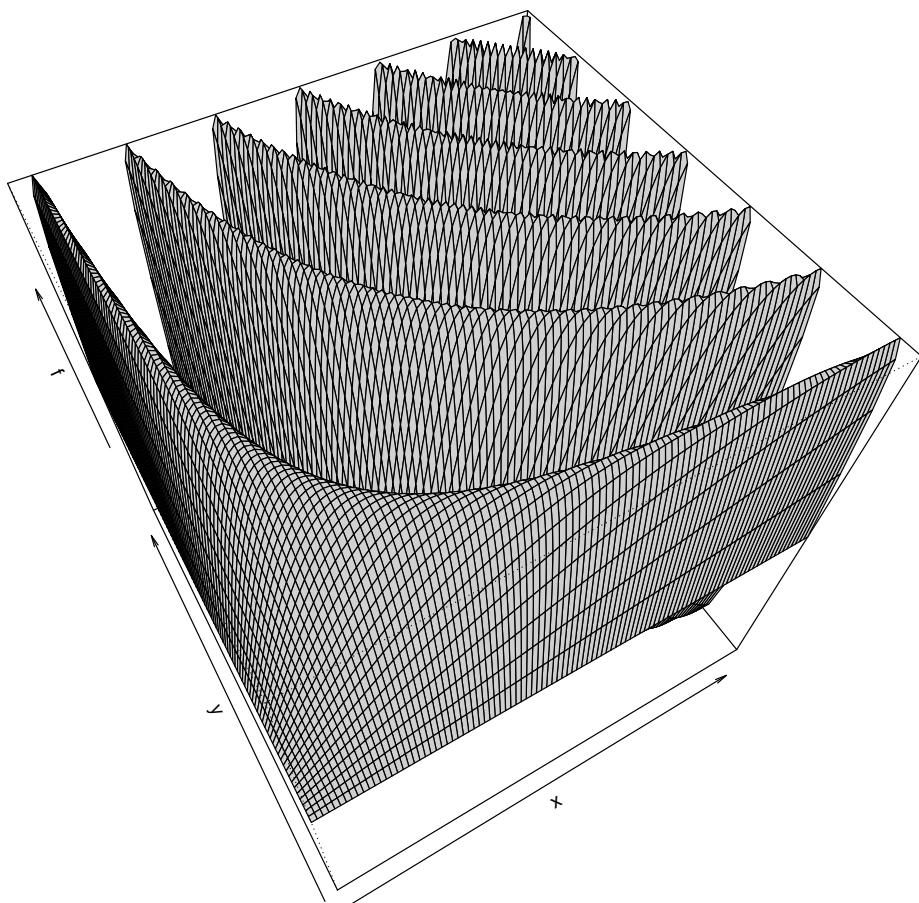
par(mfrow=c(2,2))
image(x,y,f)
contour(x,y,f)
contour(x,y,f,nlevels=4)
image(x,y,f,col=grey.colors(100))
contour(x,y,f,nlevels=4,add=TRUE,col="red")

```



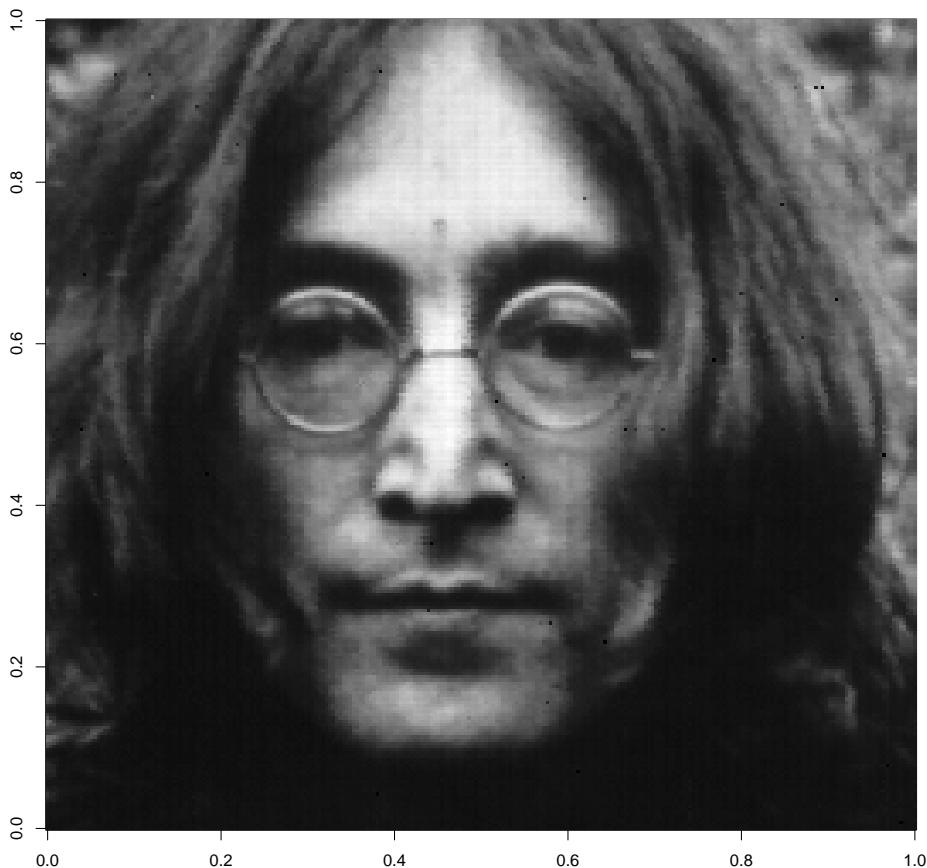
Podemos utilizar la función `persp`

```
persp(x,y,f,theta=-30,phi=55,col="lightgrey",shade=.01)
```



o representar imágenes

```
library(fields)
data(lennon)
image(lennon,col=grey(seq(0,1,l=256)))
```



3.4. Tablas de clasificación cruzada o de contingencia

```
library(MASS)
data(quine)
head(quine)

##   Eth Sex Age Lrn Days
## 1  A   M   F0  SL    2
## 2  A   M   F0  SL   11
## 3  A   M   F0  SL   14
## 4  A   M   F0  AL    5
## 5  A   M   F0  AL    5
## 6  A   M   F0  AL   13

attach(quine)
```

```

## The following objects are masked from quine (pos = 11):
##
##      Age, Days, Eth, Lrn, Sex

## The following objects are masked from quine (pos = 40):
##
##      Age, Days, Eth, Lrn, Sex






```

Cálculos sobre tablas de contingencia

```
tapply(Days,Age,mean)
```

```

##          F0          F1          F2          F3

```

```

## 14.85185 11.15217 21.05000 19.60606
tapply(Days,list(Sex,Age),mean)

##          F0          F1          F2          F3
## F 18.70000 12.96875 18.42105 14.00000
## M 12.58824  7.00000 23.42857 27.21429
tapply(Days,list(Sex,Age),function(x) sqrt(var(x)/length(x)))

##          F0          F1          F2          F3
## F 4.208589 2.329892 5.299959 2.940939
## M 3.768151 1.418093 3.766122 4.569582

```

3.5. Datos cualitativos

Supongamos unos datos cualquiera de las variables `treatment` y `improvement` de pacientes a una enfermedad determinada.

```

treatment <- factor(rep(c(1, 2), c(43, 41)), levels = c(1, 2),
                      labels = c("placebo", "treated"))
improved <- factor(rep(c(1, 2, 3, 1, 2, 3), c(29, 7, 7, 13, 7, 21)),
                     levels = c(1, 2, 3),
                     labels = c("none", "some", "marked"))

```

Tabla de contingencia

```
xtabs(~treatment+improved)
```

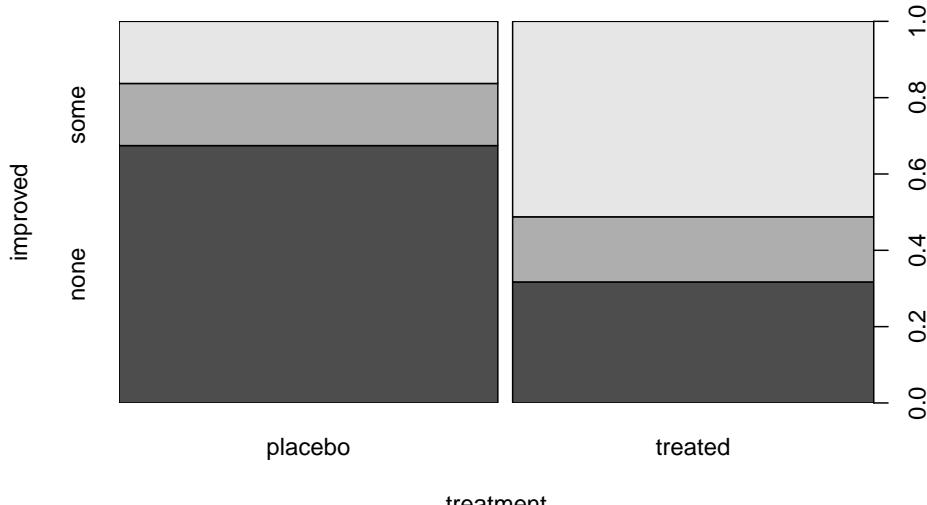
```

##           improved
## treatment none some marked
##   placebo    29     7     7
##   treated    13     7    21

```

De manera gráfica,

```
spineplot(improved ~ treatment)
```



El conjunto de datos de R, UCBAdmissions contiene los datos agregados de los solicitantes a universidad de Berkeley a los seis departamentos más grandes en 1973 clasificados por sexo y admisión.

```

data("UCBAdmissions")
?UCBAdmissions
apply(UCBAdmissions, c(2,1), sum)

##          Admit
## Gender   Admitted Rejected
##   Male      1198     1493
##   Female     557     1278
prop.table(apply(UCBAdmissions, c(2,1), sum))

##          Admit
## Gender   Admitted Rejected
##   Male    0.2646929 0.3298719
##   Female  0.1230667 0.2823685
ftable(UCBAdmissions)

##          Dept   A   B   C   D   E   F
## Admit   Gender
## Admitted Male      512 353 120 138  53  22
##           Female     89  17 202 131  94  24
## Rejected Male      313 207 205 279 138 351
##           Female     19   8 391 244 299 317

```

Con `ftable` podemos presentar la información con mayor claridad

```
ftable(round(prop.table(UCBAdmissions), 3),
      row.vars="Dept", col.vars = c("Gender", "Admit"))

##      Gender      Male      Female
##      Admit  Admitted Rejected Admitted Rejected
## Dept
## A          0.113    0.069    0.020    0.004
## B          0.078    0.046    0.004    0.002
## C          0.027    0.045    0.045    0.086
## D          0.030    0.062    0.029    0.054
## E          0.012    0.030    0.021    0.066
## F          0.005    0.078    0.005    0.070
```

Resulta más interesante mostrar la información por género `Gender` y `Dept` combinados (dimensiones 2 y 3 del array). Nótese que las tasas de admisión por `male` y `female` son más o menos similares en todos los departamentos, excepto en “A”, donde las tasas de las mujeres es mayor.

```
# prop.table(UCBAdmissions, c(2,3))
ftable(round(prop.table(UCBAdmissions, c(2,3)), 2),
      row.vars="Dept", col.vars = c("Gender", "Admit"))
```

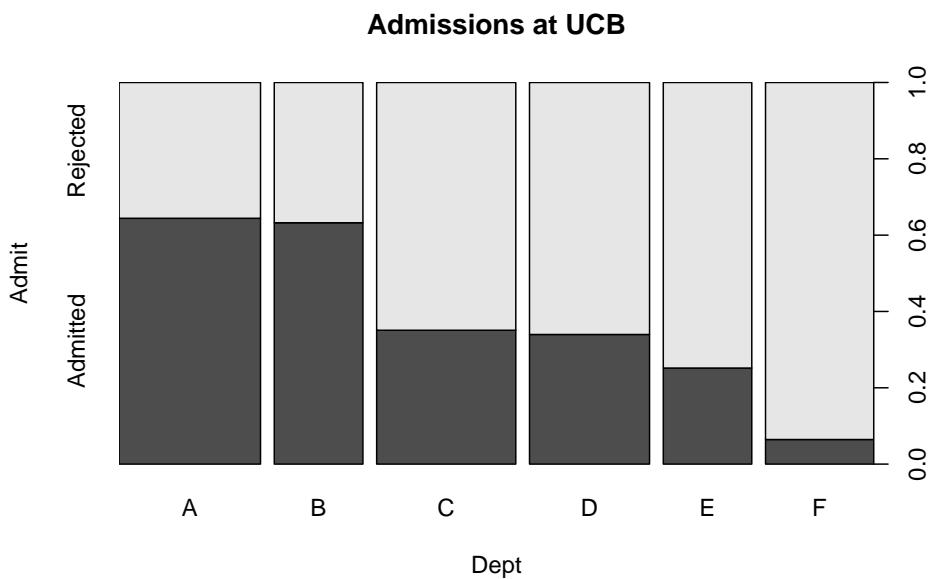
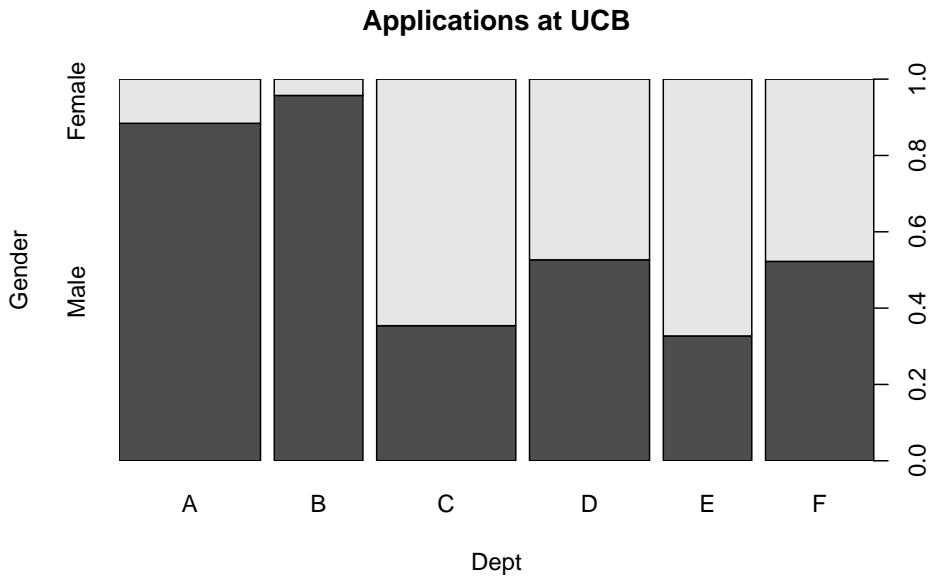
```
##      Gender      Male      Female
##      Admit  Admitted Rejected Admitted Rejected
## Dept
## A          0.62     0.38     0.82     0.18
## B          0.63     0.37     0.68     0.32
## C          0.37     0.63     0.34     0.66
## D          0.33     0.67     0.35     0.65
## E          0.28     0.72     0.24     0.76
## F          0.06     0.94     0.07     0.93

## Data aggregated over departments
apply(UCBAdmissions, c(1, 2), sum)
```

```
##           Gender
## Admit      Male Female
## Admitted 1198   557
## Rejected 1493  1278
```

Gráficamente

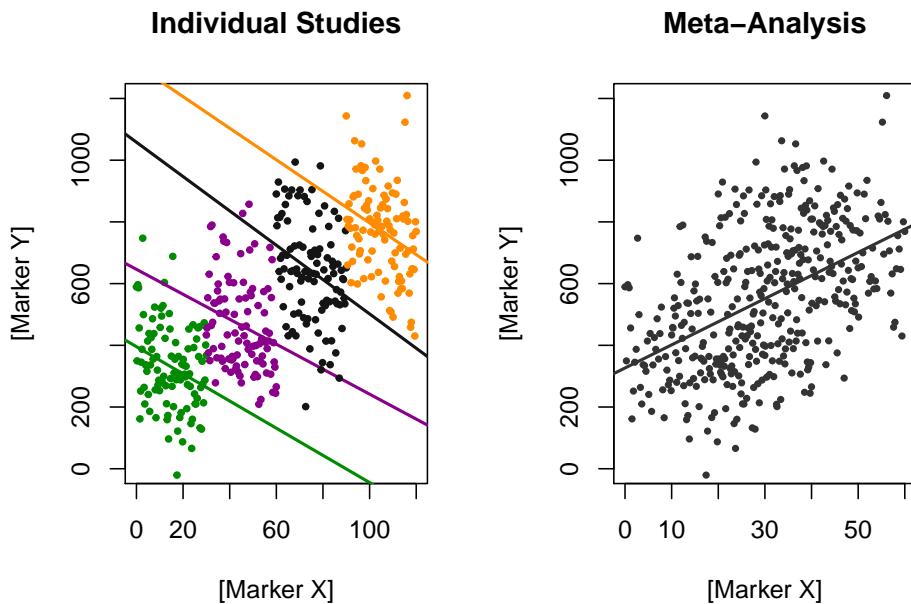
```
spineplot(margin.table(UCBAdmissions, c(3, 2)),
          main = "Applications at UCB")
```



Estos datos ilustran la denominada *paradoja de Simpson*. Este hecho ha sido analizado como un posible caso de discriminación por sexo en las tasas de admisión en Berkeley. De los 2691 hombres que solicitaron se admitidos, 1198 (44.5 %) fueron admitidos, comparado con las 1835 mujeres de las cuales tan sólo 557 (30.4 %) fueron admitidas. Se podrá por tanto concluir que los hombres

tienes tasas de admisión mayores que las mujeres. Wikipedia: Gender Bias UC Berkeley.

Otro ejemplo de la Paradoja de Simpson:



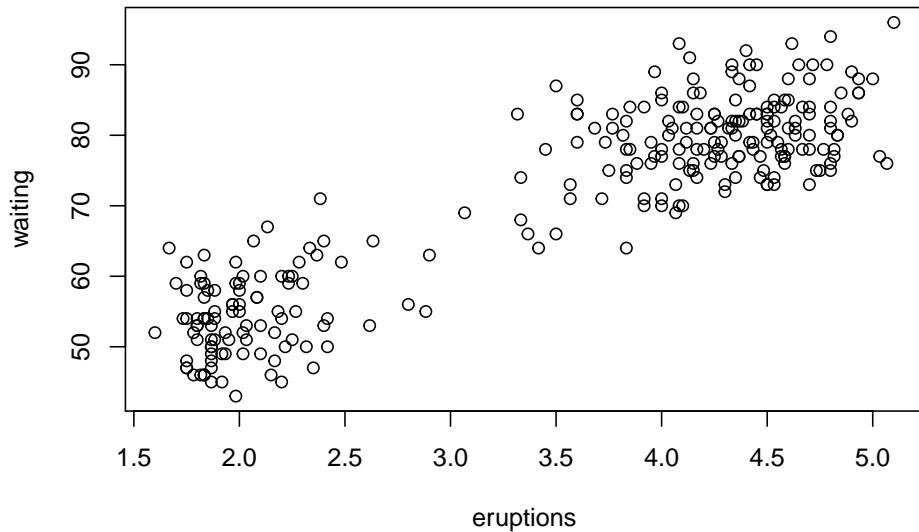
3.6. Datos cuantitativos

```
head(faithful)
```

```
##   eruptions waiting
## 1      3.600     79
## 2      1.800     54
## 3      3.333     74
## 4      2.283     62
## 5      4.533     85
## 6      2.883     55
```

Consideremos los datos del geyser Old Faithful en el parque nacional de Yellowstone, EEUU.

```
plot(faithful)
```



3.6.1. Distribuciones de frecuencias

Vamos a utilizar el conjunto de datos `faithful`, para ilustrar el concepto de distribución de frecuencias que consistiría en crear una serie de categorías o intervalos, en los que contaremos el número de observaciones en cada categoría.

```
duration <- faithful$eruptions
range(duration)
```

```
## [1] 1.6 5.1
```

Crearemos los sub-intervalos entre $[1.6, 5.1]$ y la secuencia $\{1.5, 2.0, 2.5, \dots\}$.

```
breaks <- seq(1.5,5.5,by=0.5)
breaks
```

```
## [1] 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5
```

La función `cut` nos permite dividir el rango en los intervalos que especificemos, con el argumento `right=FALSE`, consideraremos el intervalo cerrado por la derecha.

```
duration.cut = cut(duration, breaks, right=FALSE)
```

Con `table` generamos las frecuencias

```
duration.freq = table(duration.cut)
duration.freq
```

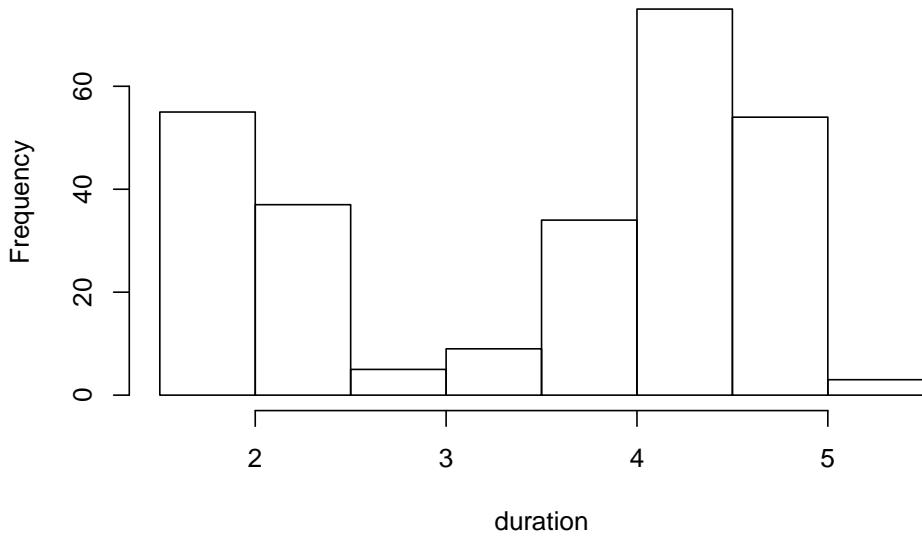
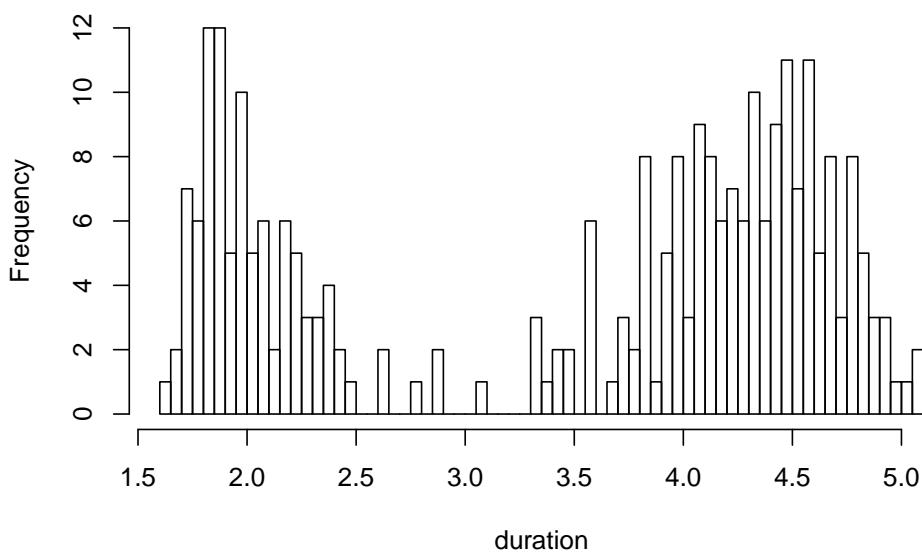
```
## duration.cut
## [1.5,2) [2,2.5) [2.5,3) [3,3.5) [3.5,4) [4,4.5) [4.5,5) [5,5.5)
##      51     41      5      7     30     73     61      4
```

Con `hist` podemos realizarlo de manera automática:

```
freq <- hist(duration)
freq
```

```
## $breaks
## [1] 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5
##
## $counts
## [1] 55 37 5 9 34 75 54 3
##
## $density
## [1] 0.40441176 0.27205882 0.03676471 0.06617647 0.25000000 0.55147059
## [7] 0.39705882 0.02205882
##
## $mids
## [1] 1.75 2.25 2.75 3.25 3.75 4.25 4.75 5.25
##
## $xname
## [1] "duration"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"

freq <- hist(duration, breaks = breaks)
```

Histogram of duration**Histogram of duration**

Estimación de densidad construye una estimación dada una distribución de probabilidad para una muestra dada.

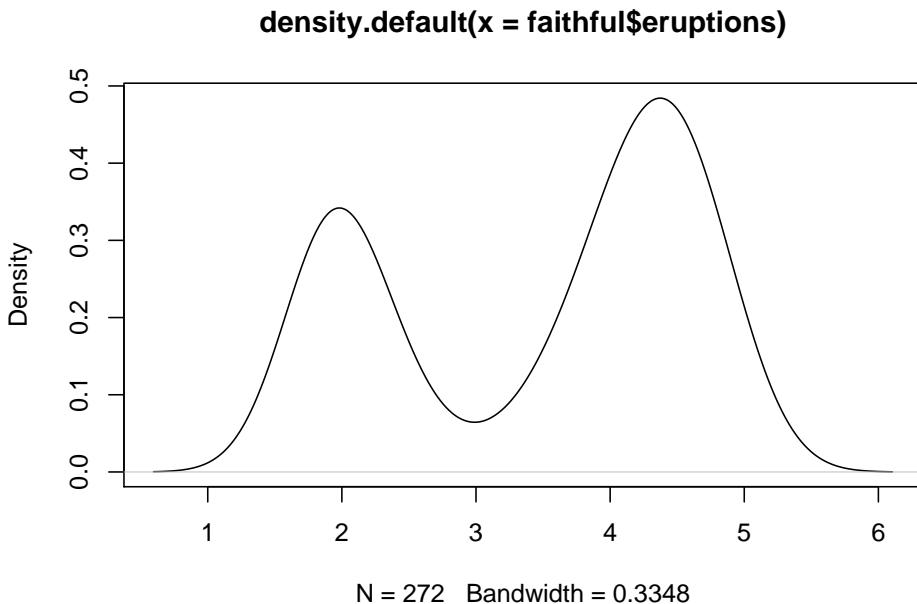
```

require(graphics)
d <- density(faithful$eruptions)
d

##
## Call:
## density.default(x = faithful$eruptions)
##
## Data: faithful$eruptions (272 obs.); Bandwidth 'bw' = 0.3348
##
##      x          y
## Min. :0.5957  Min. :0.0002262
## 1st Qu.:1.9728 1st Qu.:0.0514171
## Median :3.3500 Median :0.1447010
## Mean   :3.3500 Mean  :0.1813462
## 3rd Qu.:4.7272 3rd Qu.:0.3086071
## Max.  :6.1043  Max. :0.4842095

plot(d)

```

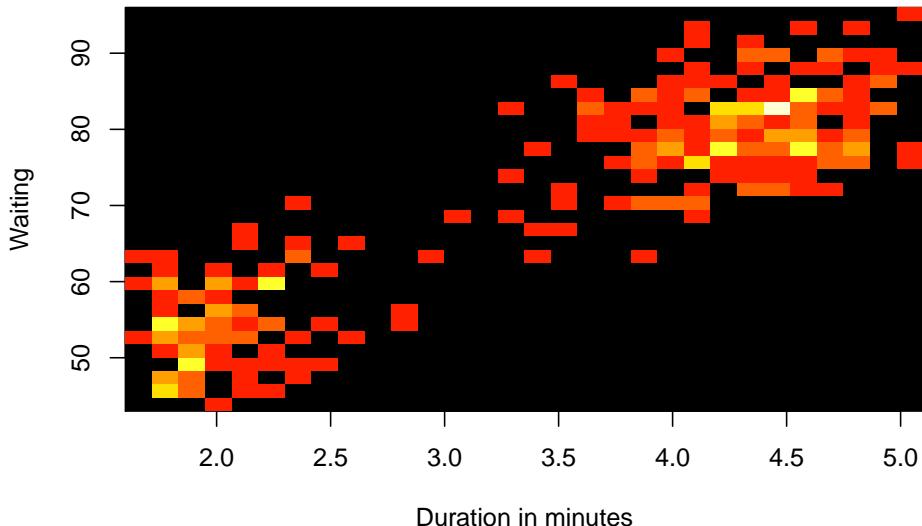


En dos dimensiones:

```

library(gplots)
h2 <- hist2d(faithful, nbins=30,xlab="Duration in minutes",ylab="Waiting")

```



```
h2
```

```
##  
## -----  
## 2-D Histogram Object  
## -----  
##  
## Call: hist2d(x = faithful, nbins = 30, xlab = "Duration in minutes",  
##      ylab = "Waiting")  
##  
## Number of data points: 272  
## Number of grid bins: 30 x 30  
## X range: ( 1.6 , 5.1 )  
## Y range: ( 43 , 96 )  
names(h2)
```

```
## [1] "counts"    "x.breaks"   "y.breaks"   "x"           "y"          "nobs"  
## [7] "call"
```

Frecuencias relativas

```
duration.relfreq <- duration.freq / nrow(faithful)  
tab <- cbind(duration.freq, duration.relfreq)  
apply(tab, 2, sum)
```

```
##      duration.freq duration.relfreq  
##                272                  1
```

Distribución de frecuencias acumuladas:

```
cumsum(duration.freq)

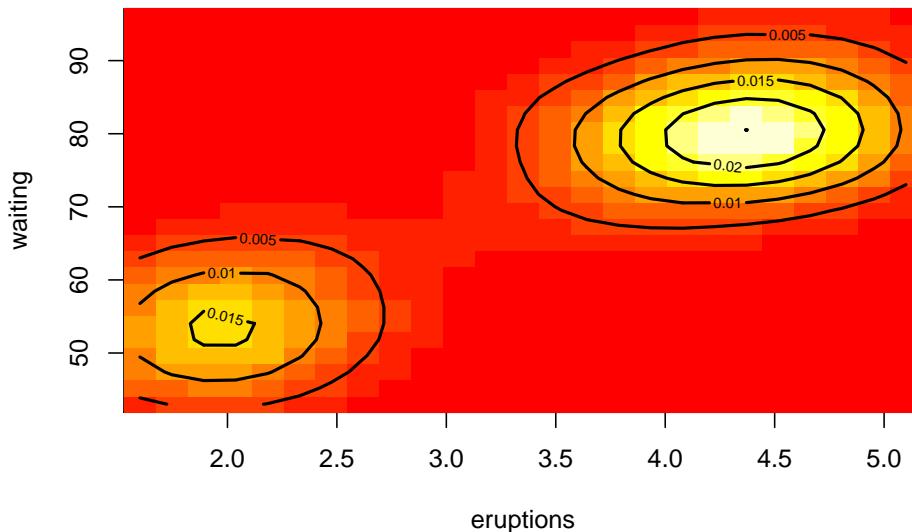
## [1.5,2) [2,2.5) [2.5,3) [3,3.5) [3.5,4) [4,4.5) [4.5,5) [5,5.5)
##      51      92      97     104     134     207     268     272

cumsum(duration.relfreq)

## [1.5,2) [2,2.5) [2.5,3) [3,3.5) [3.5,4) [4,4.5) [4.5,5) [5,5.5)
## 0.1875000 0.3382353 0.3566176 0.3823529 0.4926471 0.7610294 0.9852941
## [5,5.5)
## 1.0000000
```

Estimación bivariante tipo kernel

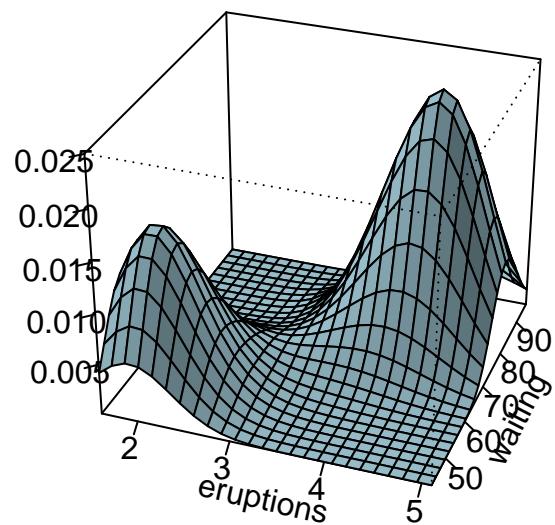
```
data("faithful")
attach(faithful)
Dens2d<-kde2d(eruptions,waiting)
image(Dens2d,xlab="eruptions",ylab="waiting")
contour(Dens2d,add=TRUE,col="black",lwd=2,nlevels=5)
```



```
detach("faithful")
```

Gráficos persp

```
persp(Dens2d,phi=30,theta=20,d=5,xlab="eruptions",ylab="waiting",zlab="",shade=.2,col="lightblue")
```



Capítulo 4

Introducción a la programación básica con R

4.1. Condicionales

Comparaciones

- equal: ==

```
"hola" == "hola"
```

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

```
"hola" == "Hola"
```

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

```
1 == 2-1
```

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

- not equal: !=

```
a <- c(1,2,4,5)
```

```
b <- c(1,2,3,5)
```

```
a == b
```

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

```
a != b
```

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

- mayor/menor que: > <

56 CAPÍTULO 4. INTRODUCCIÓN A LA PROGRAMACIÓN BÁSICA CON R

```
set.seed(1)
a <- rnorm(10)
b <- rnorm(10)
a<b

## [1] TRUE TRUE TRUE FALSE TRUE TRUE FALSE TRUE TRUE TRUE
  ▪ mayor/menor que o igual: >= <=
set.seed(2)
a <- rnorm(10)
b <- rnorm(10)
a >= b

## [1] FALSE FALSE TRUE FALSE FALSE TRUE FALSE FALSE TRUE FALSE
  ▪ which
set.seed(3)
which(a>b)

## [1] 3 6 9
LETTERS

## [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q"
## [18] "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
which(LETTERS=="R")

## [1] 18
  ▪ which.min o which.max
set.seed(4)
a <- rnorm(10)
a

## [1] 0.2167549 -0.5424926  0.8911446  0.5959806  1.6356180  0.6892754
## [7] -1.2812466 -0.2131445  1.8965399  1.7768632
which.min(a)

## [1] 7
which.max(a)

## [1] 9
  ▪ is.na
a[2] <- NA
is.na(a)
```

```
## [1] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
which(is.na(a))

## [1] 2
```

4.2. Operadores Lógicos

- and: &

```
z = 1:6
which(2 < z & z > 3)
```

```
## [1] 4 5 6
```

- or: |

```
z = 1:6
(z > 2) & (z < 5)
```

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

```
which((z > 2) & (z < 5))
```

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

- not: !

```
x <- c(TRUE, FALSE, 0, 6)
y <- c(FALSE, TRUE, FALSE, TRUE)
```

```
!x
```

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

Ejemplo:

- && vs &

```
x&y
```

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

```
x&&y
```

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

- || vs |

```
x||y
```

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

```
x|y
```

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

4.3. if

```
if(cond1=true) { cmd1 } else { cmd2 }
if(1==0) {
  print(1)
} else {
  print(2)
}

## [1] 2
```

4.4. ifelse

```
ifelse(test, true_value, false_value)
x <- 1:10 # Creates sample data
ifelse(x<5 | x>8, x, 0)

## [1] 1 2 3 4 0 0 0 0 9 10
```

4.5. Loops o Bucles

Los más empleados en R son `for`, `while` y `apply`. Los menos habituales `repeat`. La función `break` sirve para salir de un bucle loop.

4.5.1. for

Sintaxis

```
for(variable in sequence) {
  statements
}

for (j in 1:5)
{
  print(j^2)
}
```

```
## [1] 1
## [1] 4
## [1] 9
## [1] 16
## [1] 25
```

Repetir el bucle guardando los resultados en un vector `x`.

```
n = 5
x = NULL # creates a NULL object
for (j in 1:n)
{
  x[j] = j^2
}
x
```

```
## [1] 1 4 9 16 25
```

Generamos el lanzamiento de un dado

```
nsides = 6
ntrials = 1000
trials = NULL
for (j in 1:ntrials)
{
  trials[j] = sample(1:nsides,1) # We get one sample at a time
}
mean(trials^2)
```

```
## [1] 14.563
```

Ejemplo:

```
x <- 1:10
z <- NULL
for(i in seq(along=x)) {
  if (x[i]<5) {
    z <- c(z,x[i]-1)
  } else {
    stop("values need to be <5")
  }
}
## Error: values need to be <5
z
## [1] 0 1 2 3
```

4.5.2. while

Similar al bucle `for`, pero las iteraciones están controladas por una condición.

```
z <- 0
while(z < 5) {
  z <- z + 2
  print(z)
}

## [1] 2
```

60 CAPÍTULO 4. INTRODUCCIÓN A LA PROGRAMACIÓN BÁSICA CON R

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

Capítulo 5

Distribuciones de probabilidad en R

El paquete **stats** de R (que se instala por defecto al instalar R, y se carga en memoria siempre que iniciamos sesión) implementa numerosas funciones para la realización de cálculos asociados a distintas distribuciones de probabilidad.

Entre las utilizadas más comúnmente podemos citar:

Distribuciones discretas:

Distribución	Nombre en R
Binomial	<code>binom</code>
Poisson	<code>pois</code>
Geométrica	<code>geom</code>
Hipergeométrica	<code>hyper</code>

Distribuciones continuas:

Distribución	Nombre en R
Uniforme	<code>unif</code>
Normal	<code>norm</code>
t-Student	<code>t</code>
F-Fisher	<code>F</code>
Chi-cuadrado	<code>chisq</code>

- Examinamos algunas de las operaciones básicas asociadas con las distribuciones de probabilidad.
- Hay un gran número de distribuciones de probabilidad disponibles, pero sólo observamos unas pocas.
- Para obtener una lista completa de las distribuciones disponibles en R puede utilizar el siguiente comando:

```
help("Distributions")
```

- Para cada distribución hay cuatro comandos. Los comandos para cada distribución están precedidos de una letra para indicar la funcionalidad:
 - **d**: devuelve la función de densidad de probabilidad
 - **p**: devuelve la función de densidad acumulada
 - **q**: returns the inverse cumulative density function (quantiles)
 - **r**: devuelve los números generados aleatoriamente

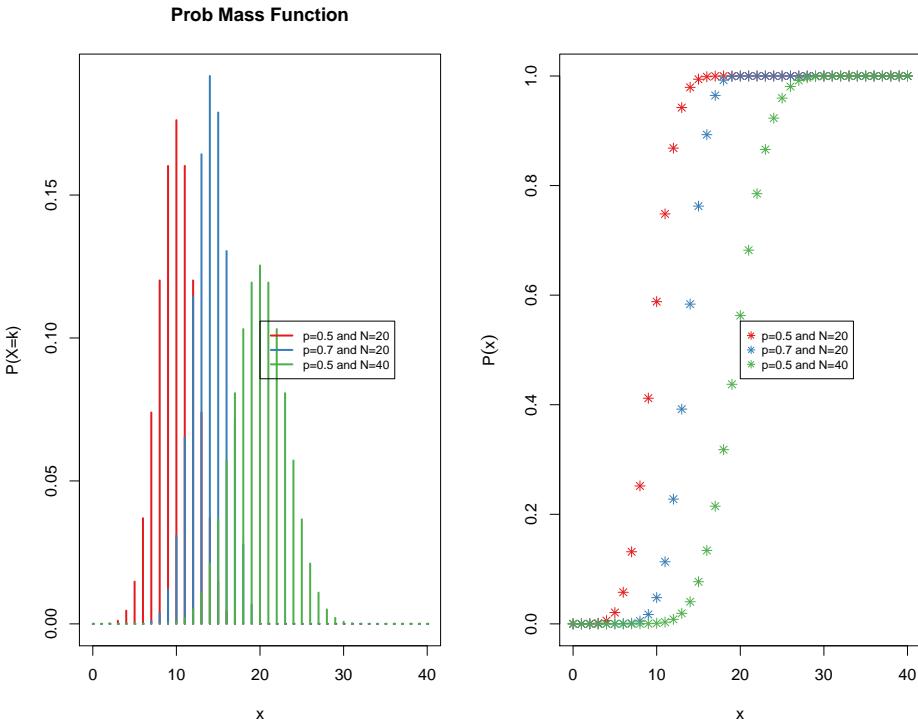
5.1. Distribución binomial $Bin(n, p)$

- La distribución binomial es una distribución de probabilidad discreta. Describe el resultado de ensayos independientes de n en un experimento. Se supone que cada ensayo tiene sólo dos resultados, ya sea éxito o fracaso. Si la probabilidad de un ensayo exitoso es de p , entonces la probabilidad de tener resultados exitosos de k en un experimento de ensayos independientes de n es dada por la **probabilidad de la función de masa**:

$$f(k; n, p) = \Pr(X = k) = \binom{n}{k} p^k (1 - p)^{n-k}, \quad k = 0, 1, 2, \dots, n$$

La **función de distribución acumulativa** puede expresarse como:

$$F(k; n, p) = \Pr(X \leq k) = \sum_{i=0}^k \binom{n}{i} p^i (1 - p)^{n-i}$$



with *mean* np and *variance* $np(1 - p)$.

Pregunta:

Suponga que hay doce preguntas de opción múltiple en un examen de matemáticas. Cada pregunta tiene cinco posibles respuestas, y sólo una de ellas es correcta. Encuentre la probabilidad de tener cuatro o menos respuestas correctas si un estudiante intenta responder a cada pregunta al azar.

Solución:

Dado que sólo una de cada cinco respuestas posibles es correcta, la probabilidad de responder correctamente una pregunta al azar es de $1/5 = 0.2$. Podemos encontrar la probabilidad de tener exactamente 4 respuestas correctas por intentos aleatorios de la siguiente manera.

```
p = 1/5
n = 12
k = 4
dbinom(k, size=n, prob=0.2)
```

```
## [1] 0.1328756
```

Para encontrar la probabilidad de tener cuatro o menos respuestas correctas mediante intentos aleatorios, aplicamos la función `dbinom` con k=0,1,2,4``.

```

prob <- NULL
for(k in 0:4){
  prob <- c(prob, dbinom(k,n,p))
}
prob

## [1] 0.06871948 0.20615843 0.28346784 0.23622320 0.13287555
sum(prob)

## [1] 0.9274445
# or simply
sum(dbinom(0:4,n,p))

## [1] 0.9274445

```

Alternativamente, podemos usar la función de probabilidad acumulada para la distribución binomial ‘*pbinom*’.

```
pbinom(4, size=n, prob=0.2)
```

```
## [1] 0.9274445
```

Solución: La probabilidad de que cuatro o menos preguntas sean contestadas correctamente al azar en un cuestionario de opción múltiple de doce preguntas es del 92,7 %.

¿Cuál es la probabilidad de que 2 ó 3 preguntas sean respondidas correctamente?

```
sum(dbinom(2:3,n,p))
```

```
## [1] 0.519691
```

Pregunta:

Supongamos que la empresa **A** fabricó un producto **B** con una probabilidad de 0,005 de ser defectuoso. Suponga que el producto **B** se envía en una caja que contiene 25 **B** artículos. ¿Cuál es la probabilidad de que una caja elegida al azar contenga exactamente un producto defectuoso?

Solución:

Pregunta reformulada: ¿Cuál es $P(X = 1)$ cuando X tiene la distribución $Bin(25, 0,005)$?

$$P(X = 1) = \binom{25}{1} 0,005^1 (1 - 0,005)^{(25-1)}$$

```
p=0.005
choose(25,1)*p^1*(1-p)^(25-1)

## [1] 0.1108317
# or
dbinom(1,25,0.005)

## [1] 0.1108317
```

¿Cuál es la probabilidad de que una caja elegida al azar no contenga más de un artículo defectuoso?

Solución:

$$P(X \leq 1) = P(X = 0) + P(X = 1)$$

```
sum(dbinom(0:1,25,p))

## [1] 0.9930519
# or
pbinom(1,25,0.005)

## [1] 0.9930519
```

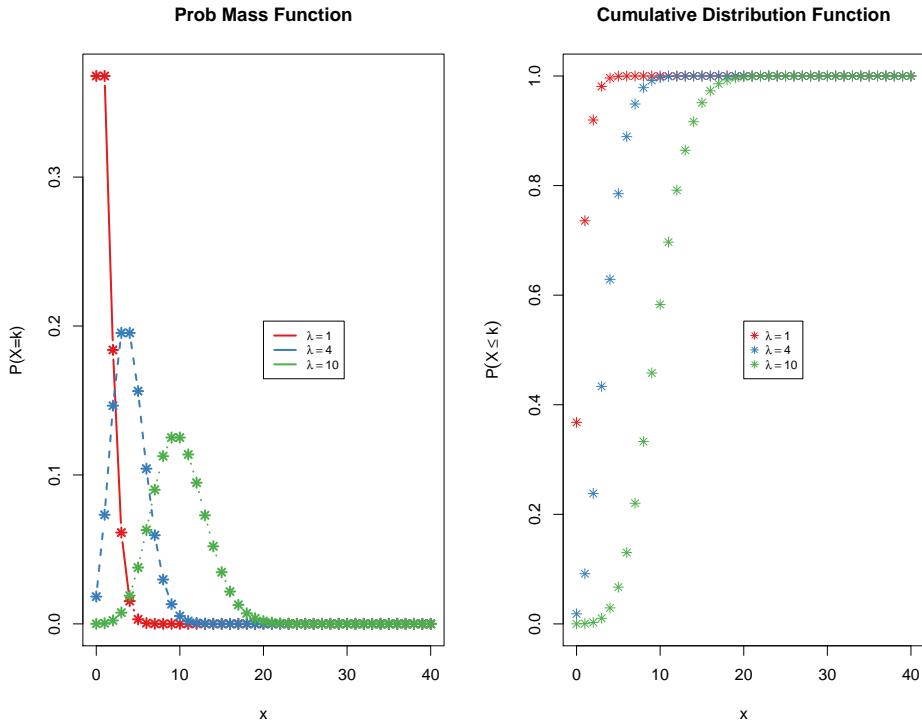
5.2. Distribución de Poisson $Pois(\lambda)$

La distribución de Poisson es la distribución de probabilidad de ocurrencias de eventos independientes en un intervalo. Si λ es la ocurrencia media por intervalo, entonces la probabilidad de tener ocurrencias k dentro de un intervalo dado es la función de masa de probabilidad dada por:

$$\Pr(k \text{ eventos en el intervalo}) = \frac{\lambda^k e^{-\lambda}}{k!}$$

La **función de densidad acumulada** para la función de probabilidad acumulativa de Poisson es

$$P(X \leq x \mid \lambda) = \frac{e^{-\lambda} \lambda^x}{x!} \quad \text{for } x = 0, 1, 2, \dots$$



Pregunta:

Supongamos que el número de plantas individuales de una especie dada que esperamos encontrar en un cuadrado de un metro cuadrado sigue la distribución de Poisson con una media de $\lambda = 10$. Encuentra la probabilidad de encontrar exactamente 12 dólares por persona.

Pregunta:

Si hay doce coches cruzando un puente por minuto en promedio, encuentre la probabilidad de tener diecisiete o más coches cruzando el puente en un minuto en particular.

5.2.1. Aproximación de Binomial como Poisson

Example

El cinco por ciento (5 %) de las bombillas del árbol de Navidad fabricadas por una compañía son defectuosas. El Gerente de Control de Calidad de la compañía está muy preocupado y por lo tanto toma muestras aleatorias de 100 bulbos que salen de la línea de montaje. Sea X el número en la muestra que está defectuoso. ¿Cuál es la probabilidad de que la muestra contenga como máximo tres bulbos defectuosos?

```
p = 0.05
k = 3
n = 100
pbinom(k,size=n,prob=p)
```

```
## [1] 0.2578387
```

Se puede demostrar que la distribución Binomial puede ser aproximada con la función de masa de probabilidad de Poisson cuando n es grande. Usando la distribución de Poisson, la media $\lambda = np$

```
lambda <- n*p
sum(dpois(0:3,lambda))
```

```
## [1] 0.2650259
```

Es importante tener en cuenta que la aproximación de Poisson a la distribución binomial funciona bien sólo cuando n es grande y p es pequeño. En general, la aproximación funciona bien si $n \geq 20$ y $p \leq 0,05$, o si $n \geq 100$ y $p \leq 0,10$.

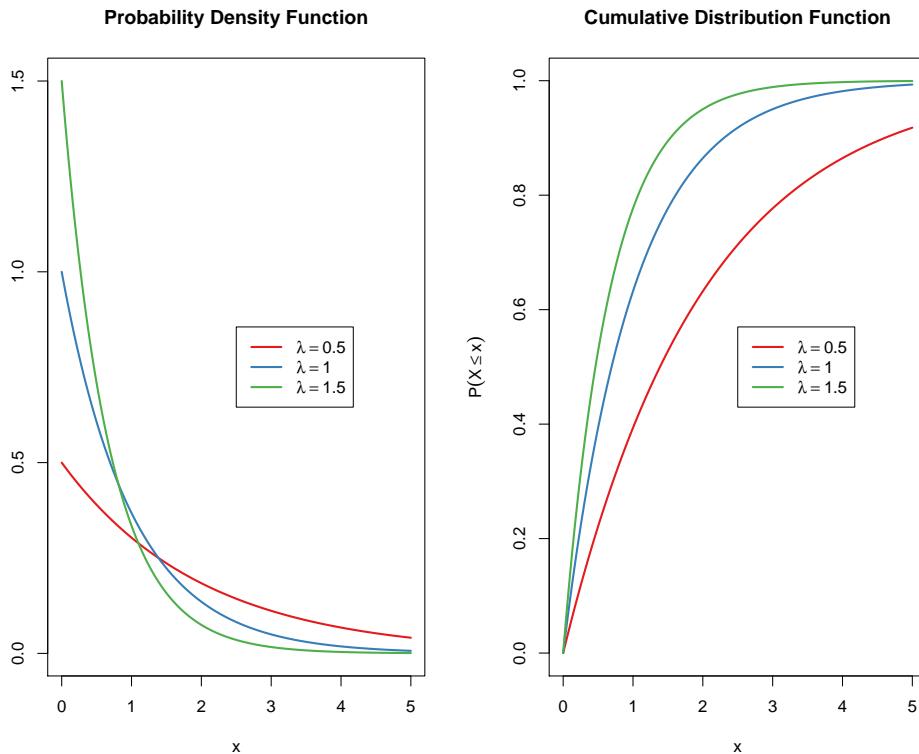
5.3. Distribution Exponencial $Exp(\lambda)$

- La distribución exponencial es la distribución de probabilidad que describe el tiempo entre eventos en un proceso de Poisson, es decir, un proceso en el que los eventos ocurren continua e independientemente a una tasa promedio constante.
- Es un caso particular de la distribución gamma. Es el continuo análogo de la distribución geométrica, y tiene la propiedad clave de no tener memoria. Además de ser utilizado para el análisis de los procesos de Poisson, se encuentra en varios otros contextos.
- La función de densidad de probabilidad (pdf) de una distribución exponencial como

$$f(x; \lambda) = \lambda \exp(-\lambda x)$$

donde $\lambda > 0$ es la tasa del evento (también conocida como parámetro de tasa, tasa de llegada, tasa de mortalidad, tasa de fracaso, tasa de transición). La variable exponencial $x \in [0,$

infy]



- La función de distribución acumulada de la distribución exponencial es

$$F(x) = \Pr(X \leq x) = \begin{cases} 1 - e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Con media $\mathbb{E}(X) = 1/\lambda$, and $\text{Var}(X) = 1/\lambda^2$.

Pregunta:

Supongamos que la cantidad de tiempo que uno pasa en un banco se distribuye exponencialmente con una media de 10 minutos, $\lambda = 1/10$.

- ¿Cuál es la probabilidad de que un cliente pase más de 15 minutos en el banco?
- ¿Cuál es la probabilidad de que un cliente pase más de 15 minutos en el banco si aún está en el banco? después de 10 minutos?

Soluciones:

a.

$$P(X > 15) = e^{-15\lambda} = e^{-3/2} = 0,2231$$

```
pexp(15,rate=1/10,lower.tail = FALSE) # or 1-pexp(15,rate=1/10)
```

```
## [1] 0.2231302
```

b.

$$P(X > 15 | X > 10) = P(X > 5) = e^{-1/2} = 0,606$$

```
pexp(5,rate=1/10,lower.tail = FALSE)
```

```
## [1] 0.6065307
```

5.4. Distribution Normal $\mathcal{N}(\mu, \sigma^2)$

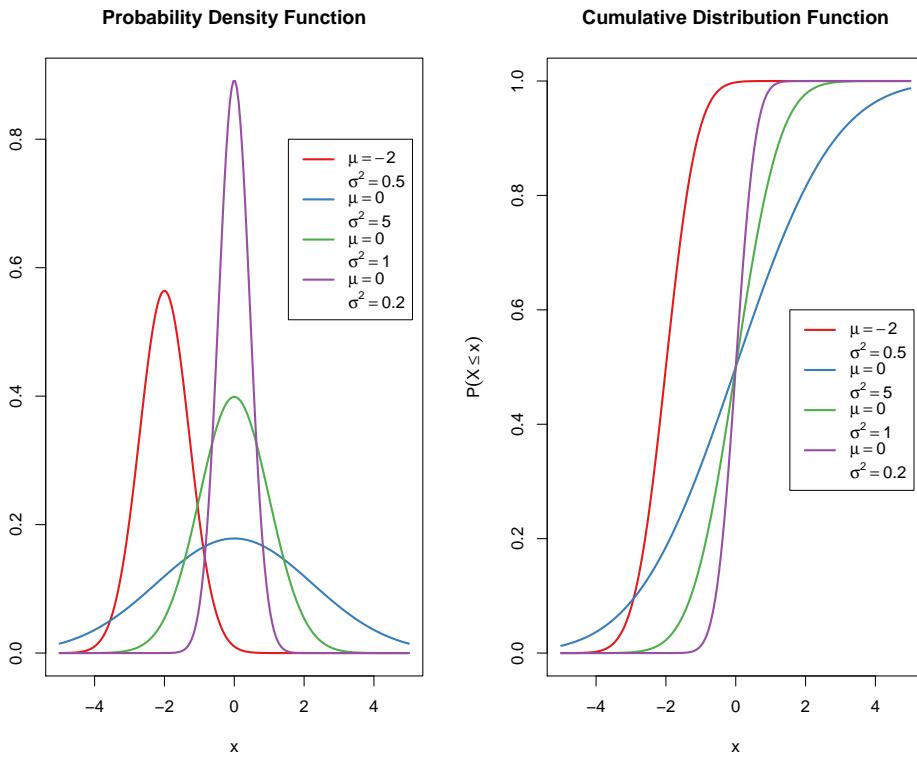
- La función de densidad de probabilidad de la distribución Normal es:

$$f(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\sigma^2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

dónde

- μ es la media de la distribución (también la mediana y el modo).
- σ es la desviación estándar ($\sigma > 0$).
- σ^2 es la variación.
- El proceso para estandarizar la distribución Normal consiste en transformar la variable Normal $N(\mu, \sigma)$ en $N(0, 1)$, es decir

$$Z = \frac{X - \mu}{\sigma} \sim N(0, 1)$$

**Pregunta:**

X es una variable normalmente distribuida con una media de $\mu = 30$ y una desviación estándar de $\sigma = 4$. Encontrar

- $P(x < 40)$
- $P(x > 21)$
- $P(30 < x < 35)$

Solucion:

- a) Para $x = 40$, la z estandarizada es $(40 - 30)/4 = 2.5$ y por tanto:

$$P(X < 40) = P(Z < 2.5) = 0.9938$$

```
pnorm(2.5) # or
## [1] 0.9937903
pnorm(40,mean=30,sd=4,lower.tail=TRUE)
## [1] 0.9937903
b)  $P(x > 21)$ 
```

```

pnorm(21,mean=30,sd=4,lower.tail = FALSE)

## [1] 0.9877755
c)  $P(30 < x < 35)$ 
pnorm(35,mean=30,sd=4,lower.tail = TRUE)-pnorm(30,mean=30,sd=4,lower.tail = TRUE)

## [1] 0.3943502

```

Pregunta:

El ingreso a una determinada universidad se determina mediante un examen nacional. Los resultados de esta prueba se distribuyen normalmente con una media de 500 y una desviación estándar de 100. Tom quiere ser admitido en esta universidad y sabe que debe obtener mejores resultados que al menos el 70% de los estudiantes que tomaron el examen. Tom toma el examen y saca 585 puntos. ¿Será admitido en esta universidad?

Solución:

```
{r hist-rnorm-03, fig.align='center'} N = 1000 hist(rnorm(N,500,100),20,col="grey")
abline(v=585,col=2)
```

Es $P(X < 585) > 70\%$?

```
pnorm(585,mean=500,sd=100)
```

```
## [1] 0.8023375
```

Tom obtuvo una puntuación mejor que el 80.23% de los estudiantes que tomaron el examen y será admitido en esta universidad.

5.5. Distribución Uniforme $U(a, b)$

- La distribución uniforme continua es la distribución de probabilidad de la selección de números aleatorios del intervalo continuo entre a y b . Su función de densidad está definida por lo siguiente.

$$f(x) = \begin{cases} \frac{1}{b-a} & a \leq x \leq b \\ 0 & \text{elsewhere} \end{cases}$$

- La función `runif()` puede ser usada para simular variables aleatorias uniformes independientes de n . Por ejemplo, podemos generar 5 números aleatorios uniformes en $[0, 1]$ como sigue:

```

set.seed(1234)
runif(5)

## [1] 0.1137034 0.6222994 0.6092747 0.6233794 0.8609154

```

- Para generar números uniformes en un intervalo del formulario $[a, b]$, usamos los argumentos `min=a`. y `max=b`. Por ejemplo:

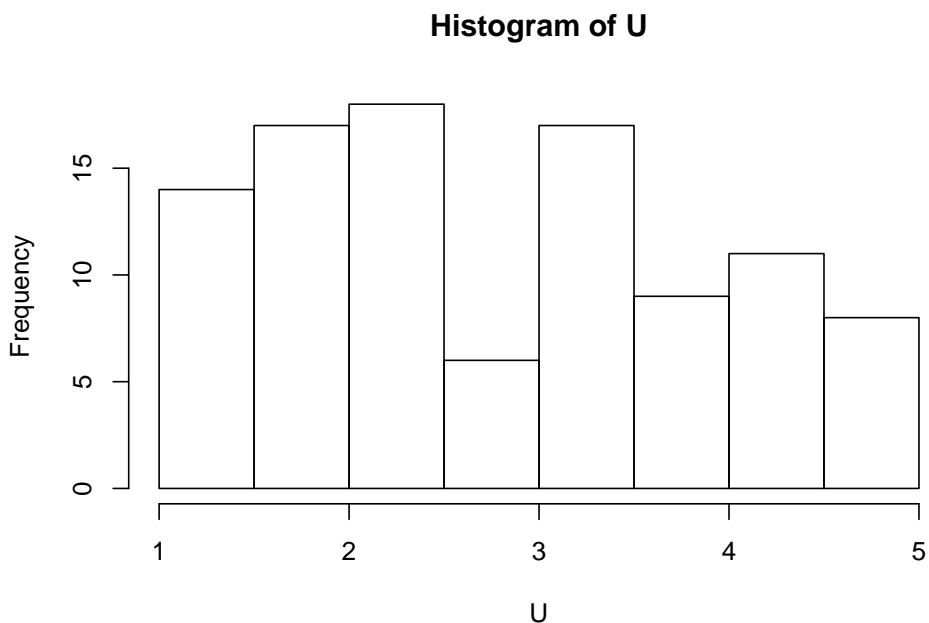
```
set.seed(1234)
runif(3, 1.2, 5.8)
```

```
## [1] 1.723036 4.062577 4.002664
```

da 3 números uniformes en $[1,2,5,8]$.

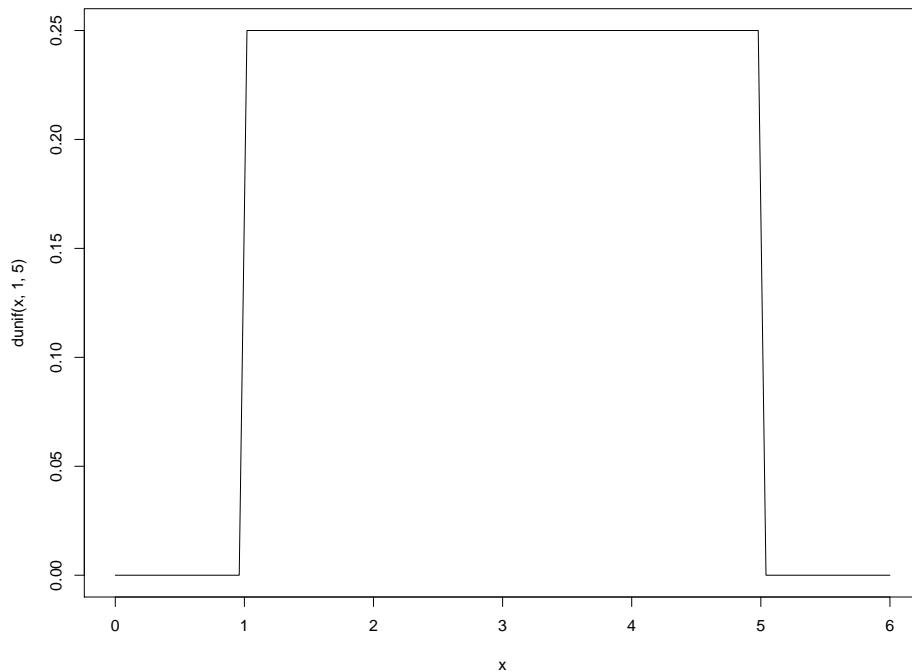
- En el siguiente ejemplo, asignaremos 100 números uniformes independientes en el intervalo $[1, 5]$ a un objeto vectorial llamado U:

```
set.seed(1234)
U <- runif(100, 1, 5)
hist(U)
```



- La función de densidad puede ser calculada usando `dunif()`

```
curve(dunif(x, 1, 5), from=0, to=6)
```



La media de U es $\mathbb{E}[U] = \frac{1}{2}(a + b)$, la mediana es $\mathbb{M}[U] = \frac{1}{2}(a + b)$ y la Varianza es $\text{Var}[U] = \frac{1}{12}(b - a)^2$.

Capítulo 6

Modelos lineales y análisis de la varianza

6.1. Principios de la modelización estadística

- Dado un conjunto de variables, cada una de las cuales es un vector de lecturas de un rasgo específico de las muestras en un experimento.
- **Problema:** ¿De qué manera una variable Y depende de otras variables X_1, \dots, X_n en el estudio?
- Un **modelo estadístico** define una relación matemática entre los X_i y Y . El modelo es una representación del real Y que pretende reemplazarlo en la medida de lo posible. Al menos el modelo debería capturar la dependencia de Y de los X_i .

6.1.1. Identificar y Caracterizar Variables

Este es el primer paso en el modelado:

- Qué variable es la variable de respuesta;
- Qué variables son las variables explicativas;
- ¿Son las variables explicativas continuas, categóricas o una mezcla de ambas?
- ¿Cuál es la naturaleza de la variable de respuesta? ¿Es una medición continua, un conteo, una proporción, una categoría o un tiempo hasta un evento?

6.1.2. Tipos de variables y tipo de modelo

- En función de las variables explicativas:

Las variables explicativas	Modelo
Todas las variables explicativas continuas	Regresión
Todas las variables explicativas categóricas	Análisis de varianza (ANOVA)
Variables explicativas tanto continuas como categóricas	Regresión, Análisis de Covarianza (ANCOVA)

- En función de la variable respuesta:

<i>La variable de respuesta</i>	<i>¿Qué tipo de datos es?</i>
Continuo	Regresión normal, Anova, Ancova
Proporción	Regresión logística
Conteos	Modelos log-lineales (también conocidos como regresión de Poisson)
Binario	Regresión logística binaria
Tiempo hasta evento	Análisis de supervivencia

6.2. El modelo lineal general

Los modelos lineales son una de las herramientas más importantes del análisis cuantitativo. Los utilizamos cuando queremos predecir –o explicar– una variable dependiente a partir de una o más variables independientes.

Se trata de un modelo para el **análisis de regresión**, que tiene como objetivo determinar una función matemática que describa el comportamiento de una variable dados los valores de otra u otras variables.

En el **Análisis de regresión simple**, se pretende estudiar y explicar el comportamiento de una variable que notamos y , y que llamaremos **variable respuesta**, **variable dependiente** o **variable de interés**, a partir de otra variable, que notamos x , y que llamamos **variable explicativa**, **variable independiente**, **covariante** o **regresor**. El principal objetivo de la regresión es encontrar

la función que mejor explique la relación entre la variable dependiente y las independientes.

- Una forma muy general para el modelo sería

$$y = f(x_1, x_2, \dots, x_p) + \epsilon,$$

donde f es una función desconocida y ϵ es el error en esta representación. Puesto que normalmente no tenemos suficientes datos para intentar estimar f directamente (*problema inverso*), normalmente tenemos que asumir que tiene alguna forma restringida.

- La forma más simple y común es el **modelo lineal (LM)**.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon,$$

donde β_i $i = 0, 1, 2$ son parámetros *desconocidos*. β_0 se llama el término de intercepción. Por lo tanto, el problema se reduce a la estimación de cuatro valores en lugar de los complicados e infinitos f dimensionales.

- Un modelo lineal simple con una sola variable exploratoria se define como:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

donde \hat{y} son los valores ajustados para $\hat{\beta}_0$ (intercepto) y $\hat{\beta}_1$ (pendiente). Luego por un x_i dado obtenemos un \hat{y}_i que se aproxima a y_i

Vamos a crear un ejemplo (con $p = 1$):

```
set.seed(1)
n <- 50

x <- seq(1,n)
beta0 <- 15
beta1 <- 0.5

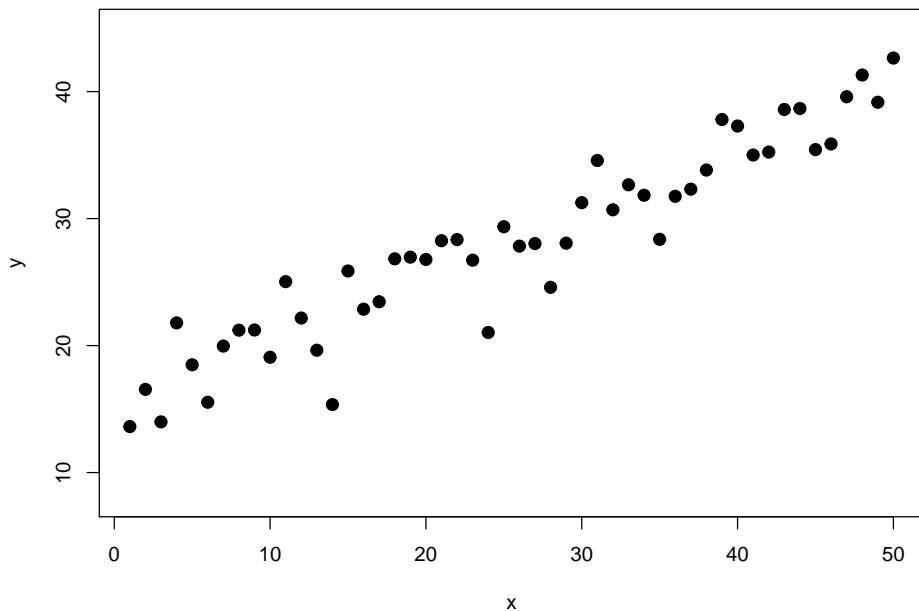
sigma <- 3 # desviación típica de los errores
eps <- rnorm(n,mean=0,sd=3) # generar errores aleatorios gaussianos

# Generar datos aleatorios
y <- beta0 + beta1*x + eps
```

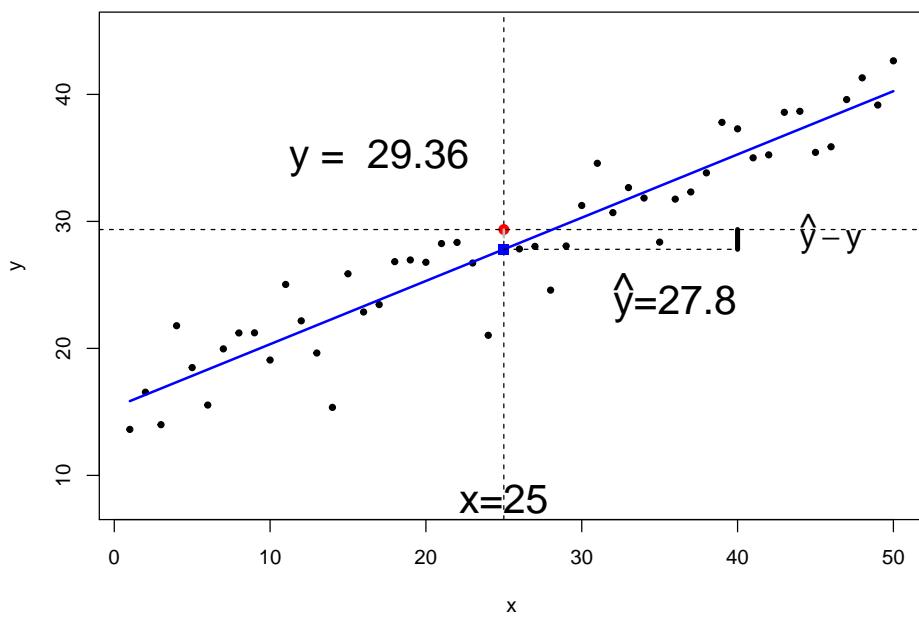
Representamos los datos

```
plot(x,y,ylim = c(8,45), cex=1.3, xlab = "x", ylab="y",pch=19)
```

78 CAPÍTULO 6. MODELOS LINEALES Y ANÁLISIS DE LA VARIANZA



Un procedimiento matemático para encontrar la curva que mejor se ajusta a un conjunto dado de puntos minimizando la suma de los cuadrados de los residuos de los puntos de la línea ajustada. Ilustración de los mínimos cuadrados de ajuste



Podemos calcular directamente las cantidades de interés, es decir, la solución ordinaria de mínimos cuadrados:

$$\min_{\beta_0, \beta_1} = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Donde

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n x_i y_i}{\sum_{i=1}^n x_i^2} \text{ y } \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

En forma matricial, con $X = [1 : x_1 : \dots : x_p]$

$$\hat{\beta} = (X'X)^{-1} X'y$$

donde $\hat{\beta} = (\hat{\beta}_0, \hat{\beta}_1)$

6.3. Definición de modelos en R

Para completar una regresión lineal usando R primero es necesario entender la sintaxis para definir los modelos.

- Un aspecto fundamental de los modelos es el uso de fórmulas modelo para especificar las variables involucradas en el modelo y las posibles interacciones entre las variables explicativas incluidas en el modelo.
- Una fórmula modelo se introduce en una función que realiza una regresión lineal o anova, por ejemplo.
- Mientras que una fórmula modelo se parece un poco a una fórmula matemática, los símbolos de la “ecuación” significan cosas diferentes a las del álgebra.

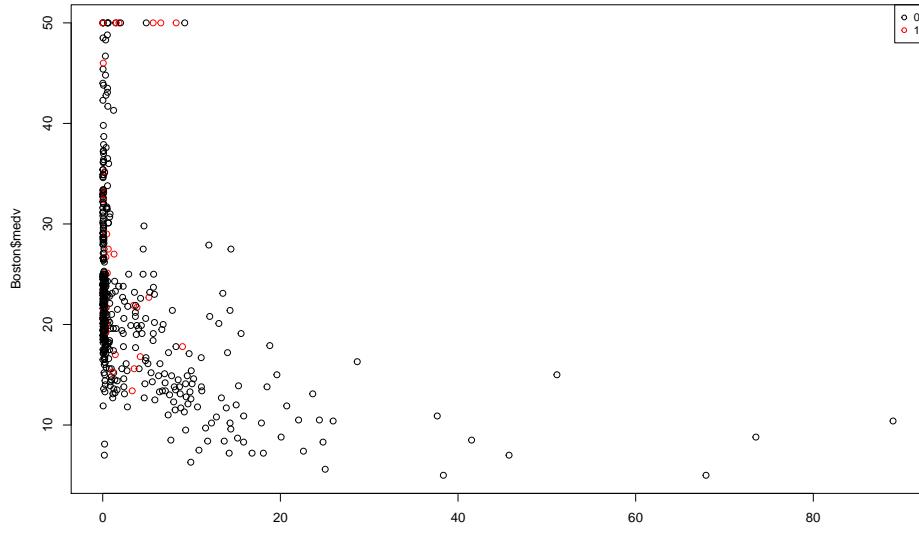
Sintaxis	Modelo	Observaciones
$y ~ x$	$y = \beta_0 + \beta_1 x$	Línea recta con intercepto implícita
$y ~ -1 + x$	$y = \beta_1 x$	Línea recta sin intercepción; es decir, un ajuste forzado (0,0)
$y ~ x + I(x^2)$	$y = \beta_0 + \beta_1 x + \beta_2 x^2$	Modelo Polinomial; I() permite símbolos matemáticos
$y ~ x + z$	$y = \beta_0 + \beta_1 x + \beta_2 z$	Regresión Lineal Múltiple
$y ~ x:z$	$y = \beta_0 + \beta_1 xz$	Modelo con interacción entre x y z
$y ~ x*z$	$y = \beta_0 + \beta_1 x + \beta_2 z + \beta_3 xz$	Equivalente a $y~x+z+x:z$

6.3.1. Ejemplo: Datos de precios de viviendas en Boston

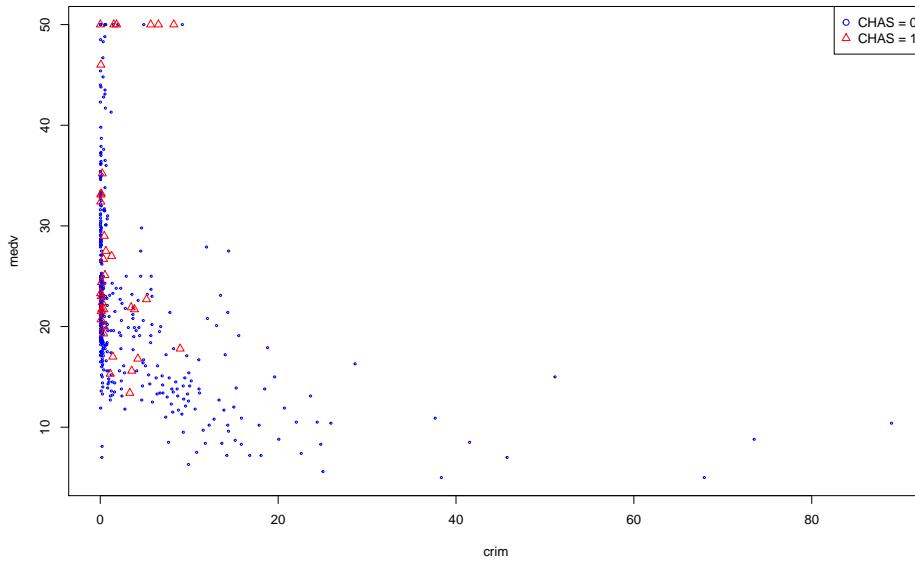
La librería MASS contiene el conjunto de datos de Boston, que registra `medv` (valor mediano de la casa) de 506 vecindarios alrededor de Boston. Trataremos de predecir el `medv` usando 13 predictores tales como `rm` (número promedio de habitaciones por casa), `age` (edad promedio de las casas), y `lstat` (porcentaje de hogares con bajo estatus socioeconómico).

```
library(MASS)
data("Boston")
?Boston

# Gráficos
plot(Boston$crim,Boston$medv,col=1+Boston$chas)
legend('topright', legend = levels(factor(Boston$chas)), col = 1:2, cex = 0.8, pch = 1)
```



```
plot(Boston[Boston$chas==0,c("crim","medv")],xlim=range(Boston$crim),ylim=range(Boston$medv))
points(Boston[Boston$chas==1,c("crim","medv")],col="red",pch=2)
legend("topright",c("CHAS = 0", "CHAS = 1"), col=c(4,2),pch=c(1,2))
```



```
attach(Boston)
```

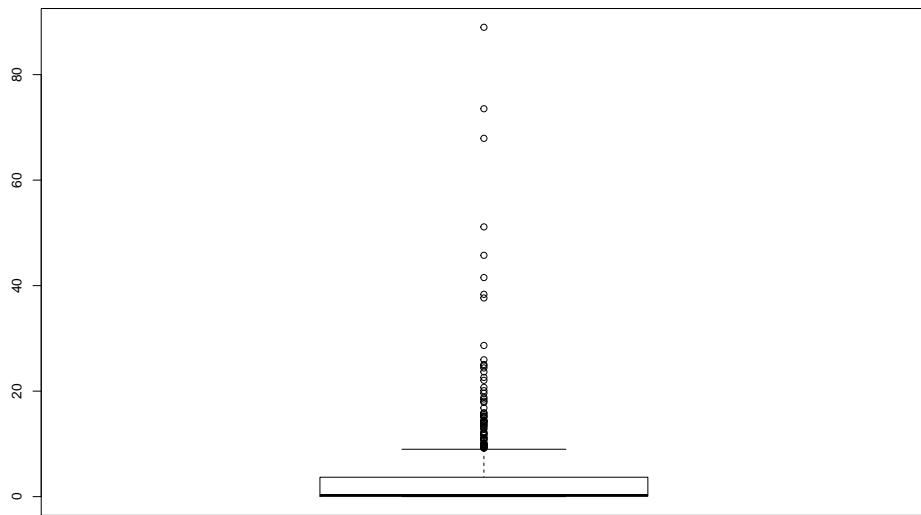
```
## The following object is masked from Wage (pos = 8):
##
##      age

## The following objects are masked from Boston (pos = 11):
##
##      age, black, chas, crim, dis, indus, lstat, medv, nox, ptratio,
##      rad, rm, tax, zn

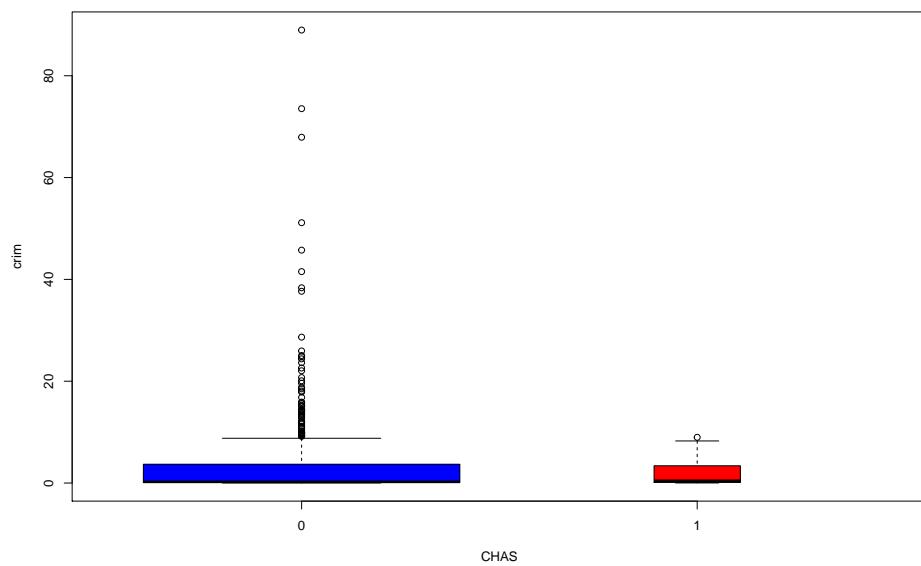
## The following object is masked from Wage (pos = 27):
##
##      age

## The following objects are masked from Boston (pos = 38):
##
##      age, black, chas, crim, dis, indus, lstat, medv, nox, ptratio,
##      rad, rm, tax, zn
boxplot(crim)
```

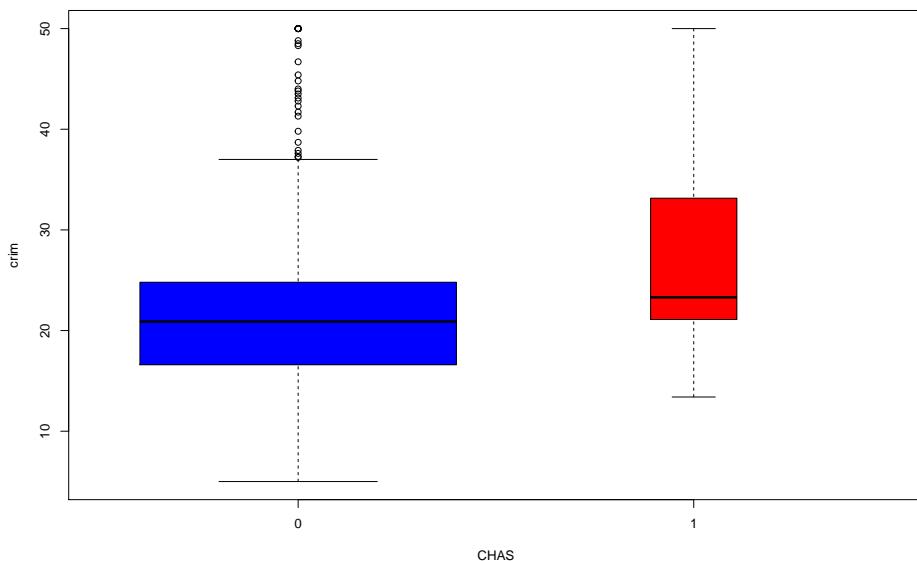
82 CAPÍTULO 6. MODELOS LINEALES Y ANÁLISIS DE LA VARIANZA



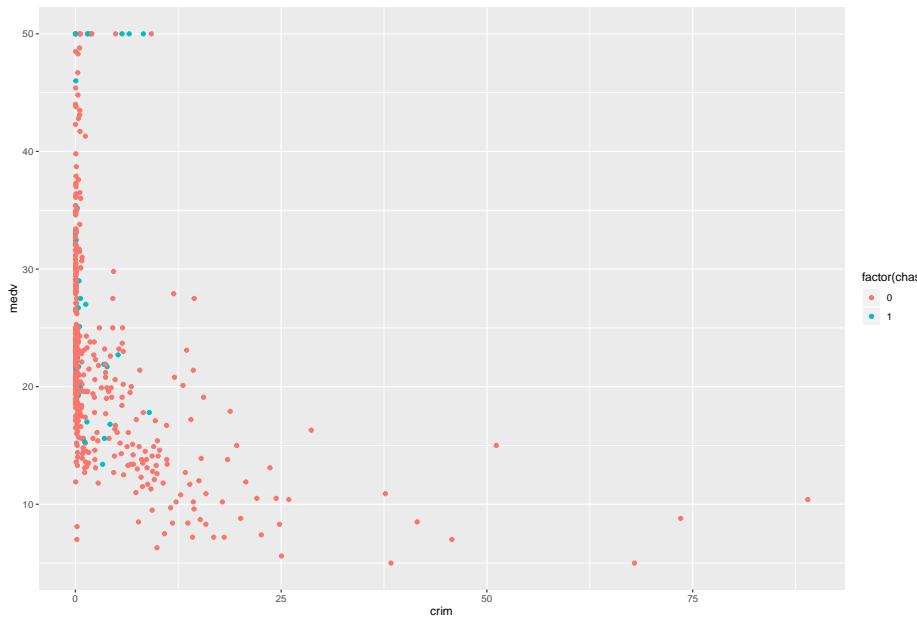
```
boxplot(crim ~ factor(chas), data = Boston,xlab="CHAS",ylab="crim",col=c(4,2),varwidth=
```



```
boxplot(medv ~ factor(chas), data = Boston,xlab="CHAS",ylab="crim",col=c(4,2),varwidth=
```

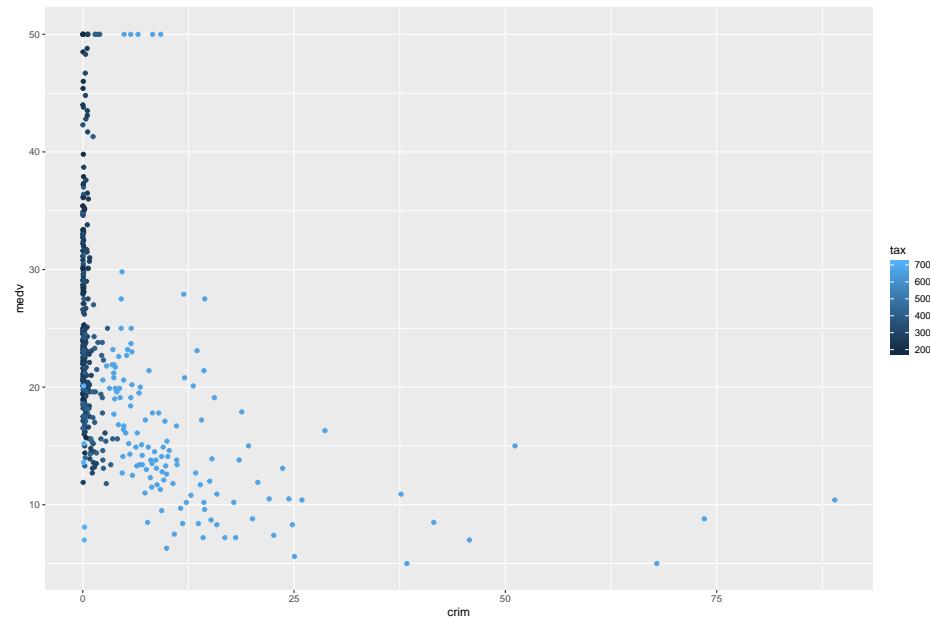


```
library(ggplot2)  
  
qplot(crim,medv,data=Boston, colour=factor(chas))
```

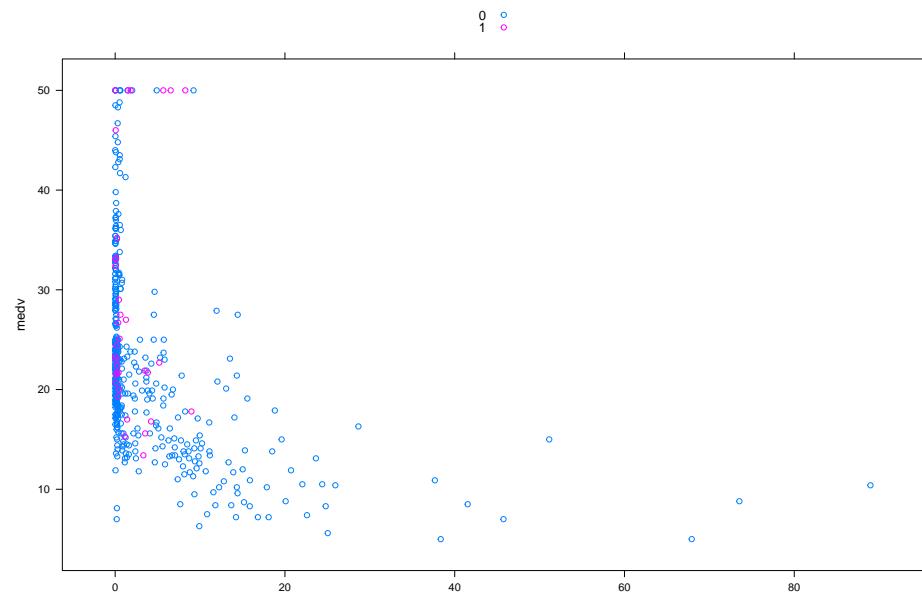


```
qplot(crim,medv,data=Boston, colour=tax)
```

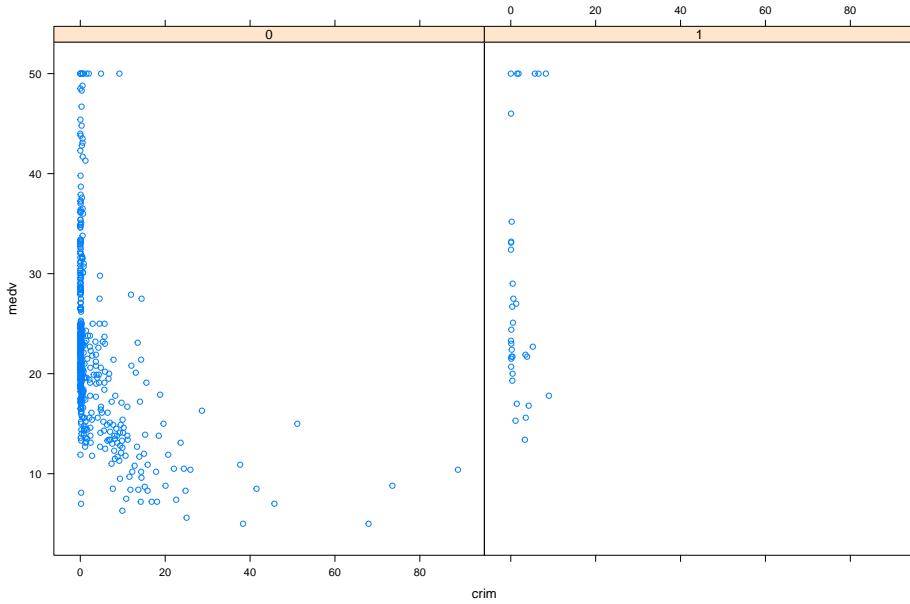
84 CAPÍTULO 6. MODELOS LINEALES Y ANÁLISIS DE LA VARIANZA



```
library(lattice)
xyplot(medv~crim,groups=factor(chas),auto.key = TRUE)
```



```
xyplot(medv~crim|factor(chas),auto.key = TRUE)
```



```

names(Boston)

## [1] "crim"      "zn"        "indus"     "chas"      "nox"      "rm"       "age"
## [8] "dis"       "rad"       "tax"       "ptratio"   "black"    "lstat"    "medv"

str(Boston)

## 'data.frame': 506 obs. of 14 variables:
## $ crim : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
## $ zn   : num  18 0 0 0 0 12.5 12.5 12.5 12.5 ...
## $ indus: num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
## $ chas : int  0 0 0 0 0 0 0 0 0 ...
## $ nox  : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
## $ rm   : num  6.58 6.42 7.18 7 7.15 ...
## $ age  : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
## $ dis  : num  4.09 4.97 4.97 6.06 6.06 ...
## $ rad  : int  1 2 2 3 3 3 5 5 5 5 ...
## $ tax  : num  296 242 242 222 222 311 311 311 311 ...
## $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
## $ black : num  397 397 393 395 397 ...
## $ lstat : num  4.98 9.14 4.03 2.94 5.33 ...
## $ medv  : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...

```

Comenzaremos usando la función `lm()` para encajar un modelo de regresión lineal simple, con `medv` como respuesta y `lstat` como predictor. La sintaxis básica de `lm()` es `lm(y~x,data)`, donde `y` es la respuesta, `x` es el predictor, y

86 CAPÍTULO 6. MODELOS LINEALES Y ANÁLISIS DE LA VARIANZA

los datos son el conjunto de datos en el que se guardan estas dos variables.

```
lm.fit <- lm(medv ~ lstat, data=Boston)
```

Si escribimos `lm.fit`, se obtiene información básica sobre el modelo. Para información más detallada, usamos `summary(lm.fit)`. Esto nos da valores de p y errores estándar para los coeficientes, así como la estadística R^2 y el estadístico para el modelo.

```
lm.fit

##
## Call:
## lm(formula = medv ~ lstat, data = Boston)
##
## Coefficients:
## (Intercept)      lstat
##           34.55      -0.95

summary(lm.fit)

##
## Call:
## lm(formula = medv ~ lstat, data = Boston)
##
## Residuals:
##    Min     1Q   Median     3Q    Max 
## -15.168 -3.990 -1.318  2.034 24.500 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 34.55384   0.56263   61.41   <2e-16 ***
## lstat       -0.95005   0.03873  -24.53   <2e-16 ***
## ---      
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.216 on 504 degrees of freedom
## Multiple R-squared:  0.5441, Adjusted R-squared:  0.5432 
## F-statistic: 601.6 on 1 and 504 DF,  p-value: < 2.2e-16
```

Podemos usar la función `names()` para averiguar qué otra información se almacena en `lm.fit`. Aunque podemos extraer estas cantidades por nombre - por ejemplo `lm.fit$coefficients` - es más seguro usar las funciones del extractor como `coef()` para acceder a ellas.

```

names(lm.fit)

## [1] "coefficients"   "residuals"      "effects"       "rank"
## [5] "fitted.values" "assign"        "qr"           "df.residual"
## [9] "xlevels"        "call"         "terms"        "model"

lm.fit$coefficients

## (Intercept)      lstat
## 34.5538409 -0.9500494

lm.fit[[1]]

## (Intercept)      lstat
## 34.5538409 -0.9500494

coef(lm.fit)

## (Intercept)      lstat
## 34.5538409 -0.9500494

```

Para obtener un intervalo de confianza para las estimaciones del coeficiente, podemos usar el comando `confint()`.

```

confint(lm.fit, level = 0.95)

##                 2.5 %     97.5 %
## (Intercept) 33.448457 35.6592247
## lstat       -1.026148 -0.8739505

```

Considere la posibilidad de construir un intervalo de confianza para β_1 utilizando la información proporcionada por el resumen de `lm.fit`:

```

summary(lm.fit)$coefficients

##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.5538409 0.56262735 61.41515 3.743081e-236
## lstat       -0.9500494 0.03873342 -24.52790 5.081103e-88

```

La función `predict()` puede ser usada para producir intervalos de confianza e intervalos de predicción para la predicción de `medv` para un valor dado de `lstat`.

```

CI <- predict(object = lm.fit, newdata = data.frame(lstat = c(5, 10, 15)),
               interval = "confidence")
CI

##      fit      lwr      upr

```

88 CAPÍTULO 6. MODELOS LINEALES Y ANÁLISIS DE LA VARIANZA

```
## 1 29.80359 29.00741 30.59978
## 2 25.05335 24.47413 25.63256
## 3 20.30310 19.73159 20.87461
PI <- predict(object = lm.fit, newdata = data.frame(lstat = c(5, 10, 15)),
              interval = "predict")
PI

##          fit      lwr      upr
## 1 29.80359 17.565675 42.04151
## 2 25.05335 12.827626 37.27907
## 3 20.30310  8.077742 32.52846
```

NOTA:

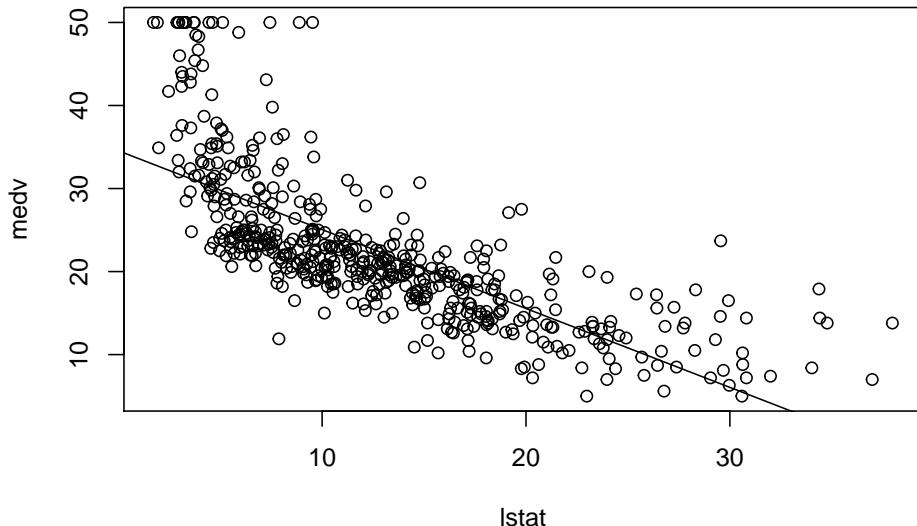
Un **intervalo de predicción** es un intervalo asociado con una variable aleatoria que aún no ha sido observada (predicción).

Un **intervalo de confianza** es un intervalo asociado a un parámetro y es un concepto de frecuentista.

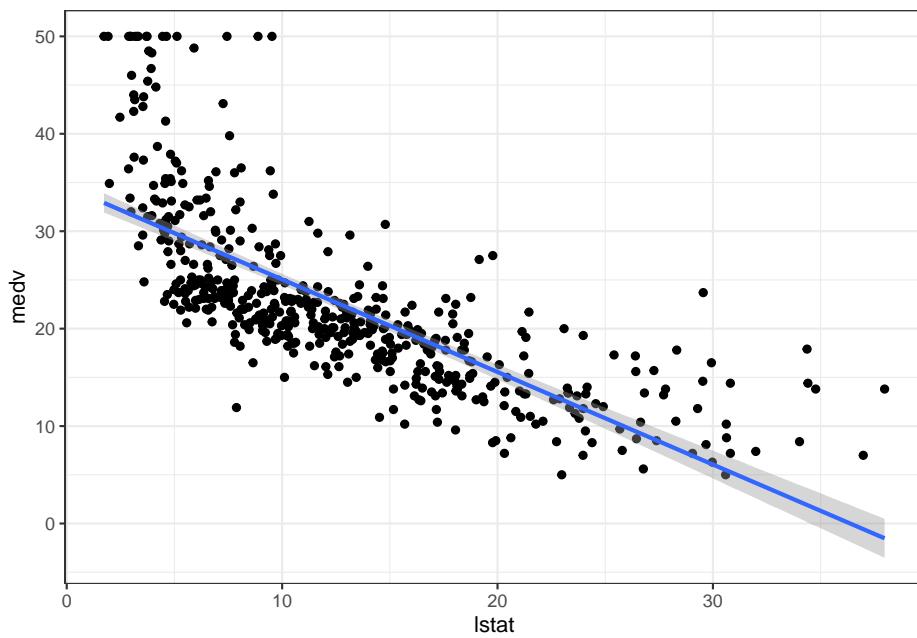
Por ejemplo, el intervalo de confianza del 95 % asociado con un valor “stat” de 10 es (24,474132,25,6325627) y el intervalo de predicción del 95 % es (12,8276263,37,2790683). Como se esperaba, los intervalos de confianza y predicción se centran en el mismo punto (un valor predicho de 25,0533473 para medv cuando lstat es igual a 10), pero estos últimos son sustancialmente más amplios.

Ahora trazaremos medv y lstat junto con la línea de regresión de los mínimos cuadrados usando las funciones `plot()` y `abline()`.

```
plot(medv ~ lstat, data = Boston)
abline(lm.fit)
```



```
# Or using ggplot2
library(ggplot2)
ggplot(data = Boston, aes(x = lstat, y = medv)) +
  geom_point() +
  geom_smooth(method = "lm") +
  theme_bw()
```

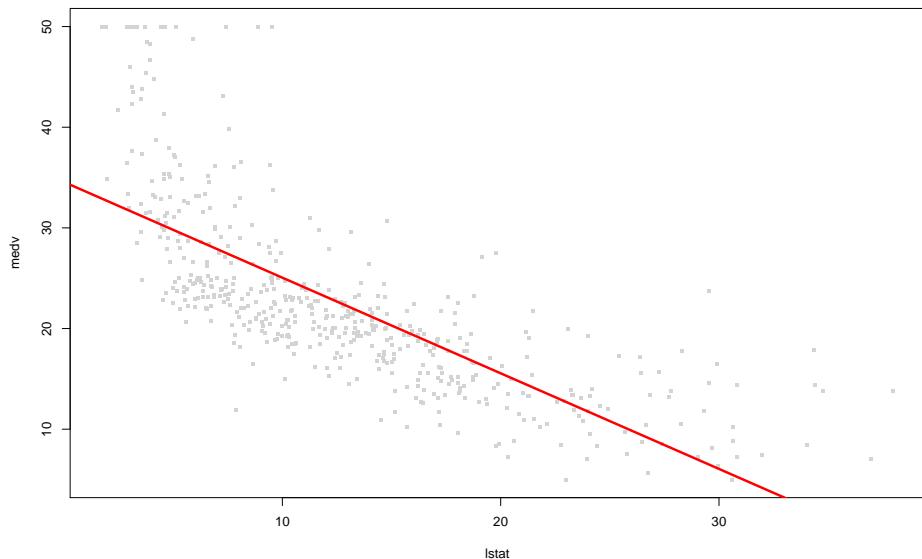


Hay alguna evidencia de no linealidad en la relación entre `lstat` y `medv`. Esta

90 CAPÍTULO 6. MODELOS LINEALES Y ANÁLISIS DE LA VARIANZA

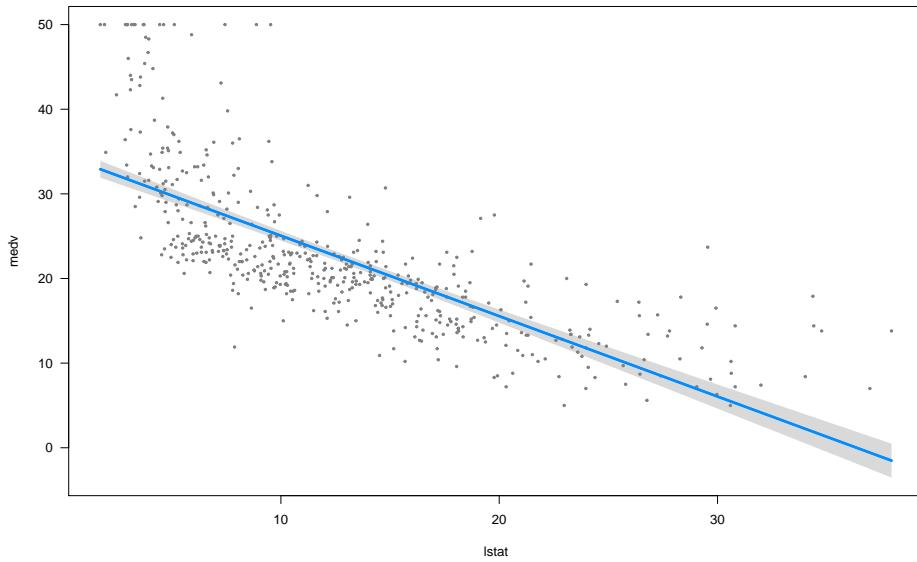
cuestión se discutirá más adelante.

```
plot(medv ~ lstat, data = Boston, pch=15, cex=.65, col="lightgrey")
abline(lm.fit, lwd = 3, col = "red")
```



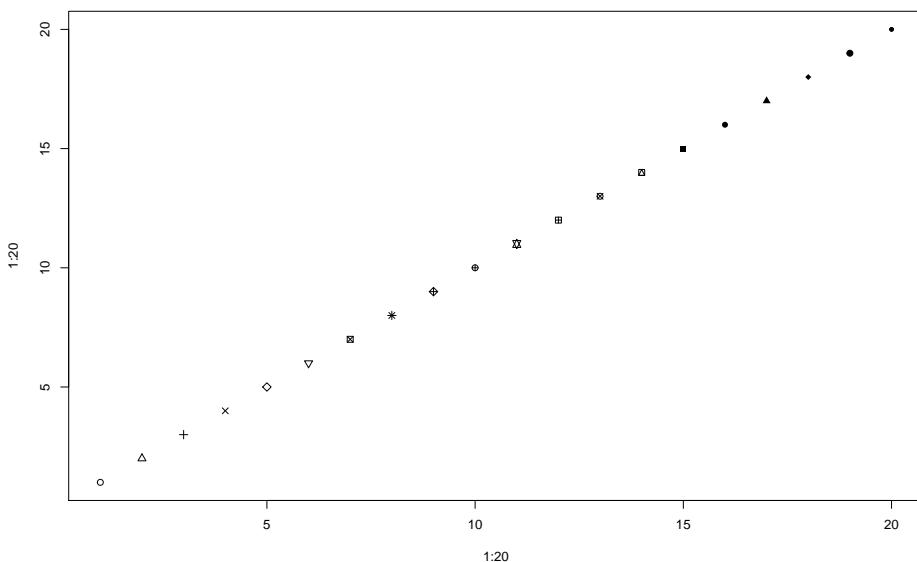
La librería(*visreg*) permite visualizar las funciones de regresión

```
# install.packages("visreg")
library(visreg)
visreg(lm.fit)
```



Opciones pch

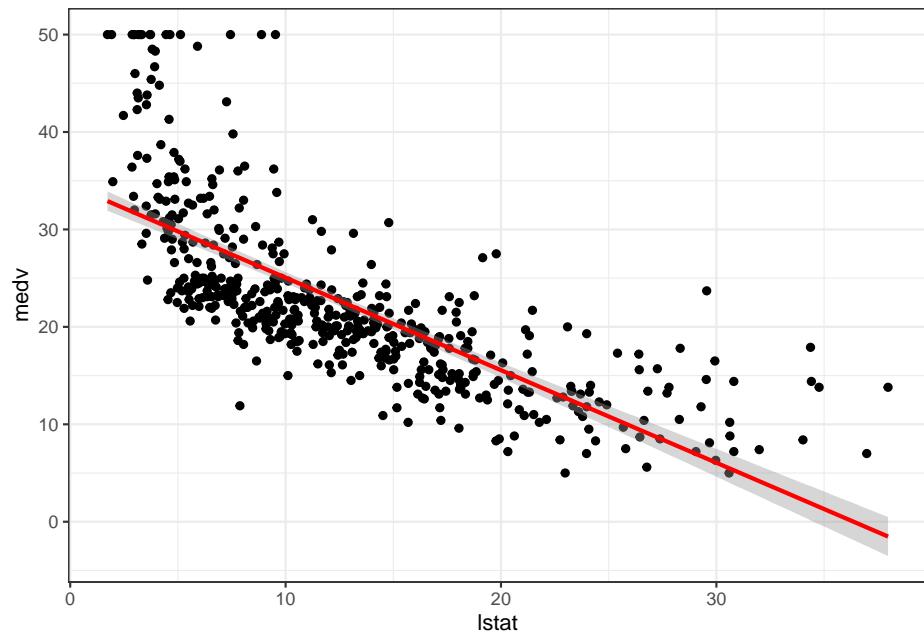
```
plot(1:20, 1:20, pch = 1:20)
```



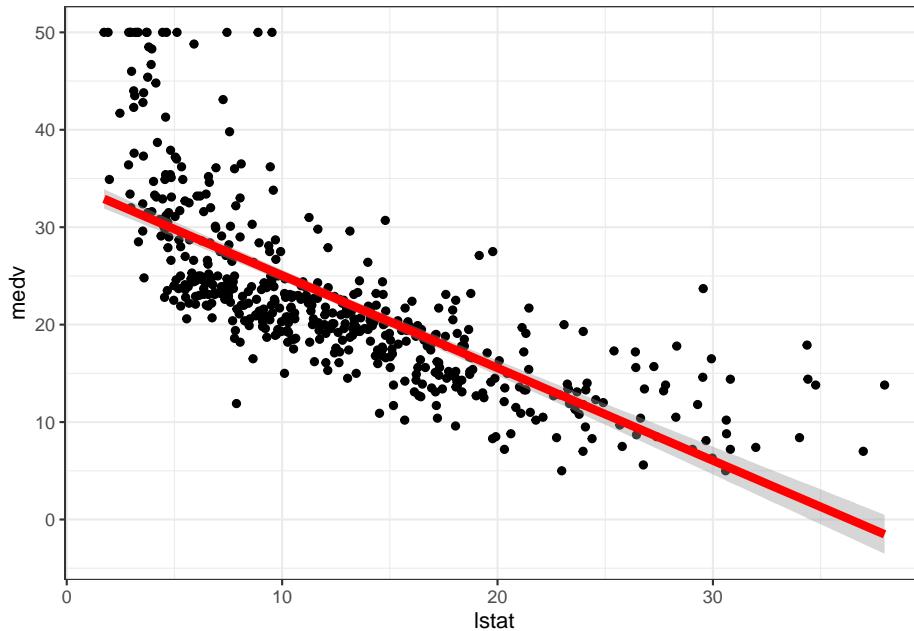
Librería ggplot2 con lm

```
ggplot(data = Boston, aes(x = lstat, y = medv)) +
  geom_point() +
  geom_smooth(method = "lm", color = "red") +
  theme_bw()
```

92 CAPÍTULO 6. MODELOS LINEALES Y ANÁLISIS DE LA VARIANZA



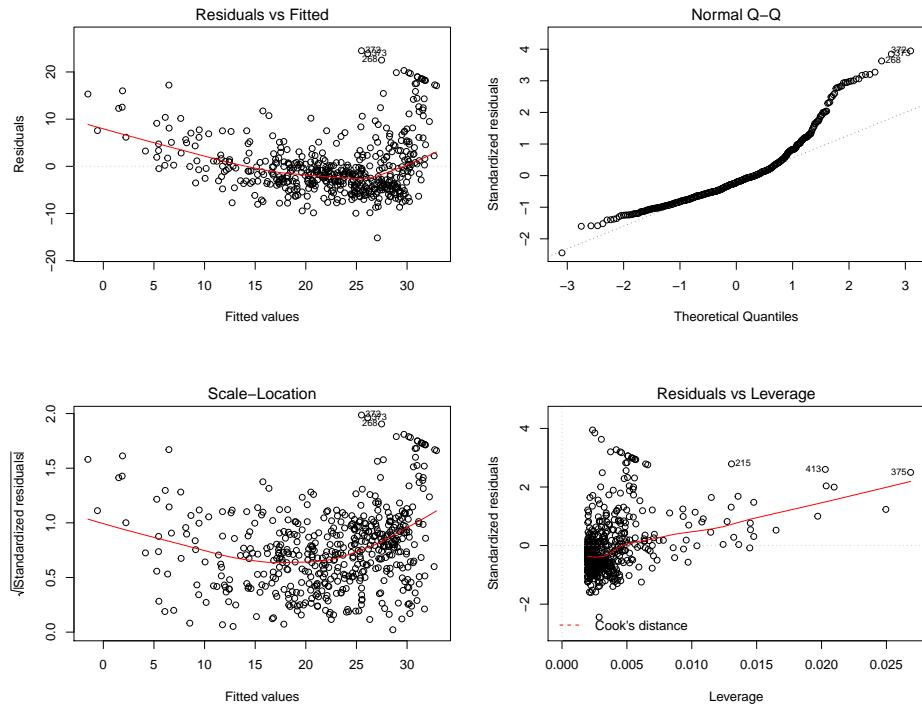
```
# thicker line
ggplot(data = Boston, aes(x = lstat, y = medv)) +
  geom_point() +
  geom_smooth(method = "lm", color = "red", size = 2) +
  theme_bw()
```



A continuación examinamos algunas gráficas de diagnóstico. Se producen automáticamente cuatro gráficas de diagnóstico aplicando la función `plot()` directamente a la salida de `lm()`. En general, este comando producirá un gráfico a la vez, y al presionar Enter se generará el siguiente gráfico. Sin embargo, a menudo es conveniente ver las cuatro parcelas juntas. Podemos lograr esto usando la función `par()`, que le dice a R que divida la pantalla en paneles separados para que se puedan ver múltiples gráficos simultáneamente. Por ejemplo, `par(mfrow = c(2, 2))` divide la región de trazado en una cuadrícula de paneles de 2\$ por 2\$.

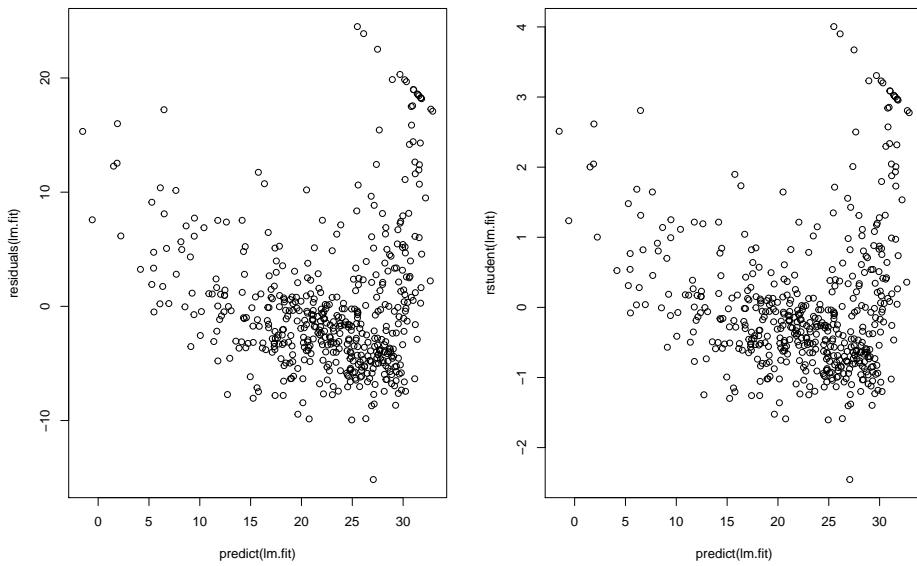
```
par(mfrow = c(2, 2))
plot(lm.fit)
```

94 CAPÍTULO 6. MODELOS LINEALES Y ANÁLISIS DE LA VARIANZA



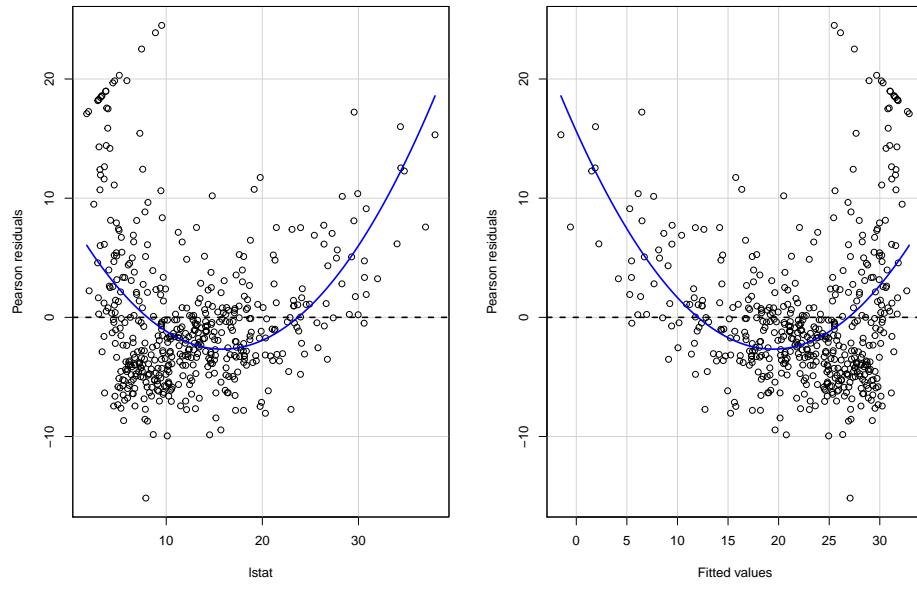
Alternatively, we can compute the residuals from a linear regression fit using the `residuals()` function. The function `rstudent()` will return the studentized residuals, and we can use this function to plot the residuals against the fitted values.

```
par(mfrow = c(1, 2))
plot(predict(lm.fit), residuals(lm.fit))
plot(predict(lm.fit), rstudent(lm.fit))
```



La biblioteca **car** tiene una función **residualPlots** para evaluar los residuos (calcula una prueba de curvatura para cada una de las parcelas añadiendo un término cuadrático y probando que la cuadrática sea cero). Ver **?residualPlots**.

```
library(car)
residualPlots(lm.fit)
```



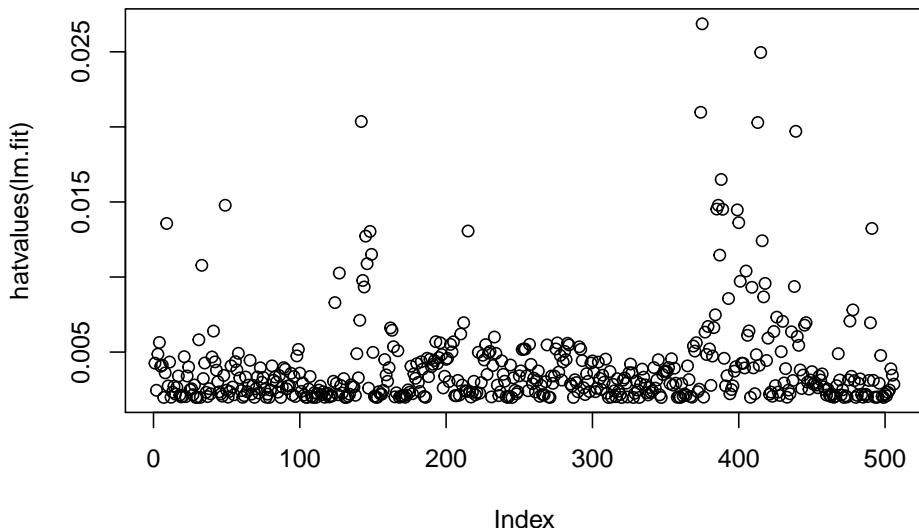
```
##          Test stat Pr(>|Test stat|)
```

96 CAPÍTULO 6. MODELOS LINEALES Y ANÁLISIS DE LA VARIANZA

```
## lstat          11.627      < 2.2e-16 ***
## Tukey test    11.627      < 2.2e-16 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Sobre la base de los gráficos de los residuos, hay alguna evidencia de no linealidad. Las estadísticas de apalancamiento pueden ser calculadas para cualquier número de predictores usando la función `hatvalues`. La función `influenceIndexPlot` del paquete `car` crea cuatro gráficos de diagnóstico que incluyen un gráfico de los valores de sombrero.

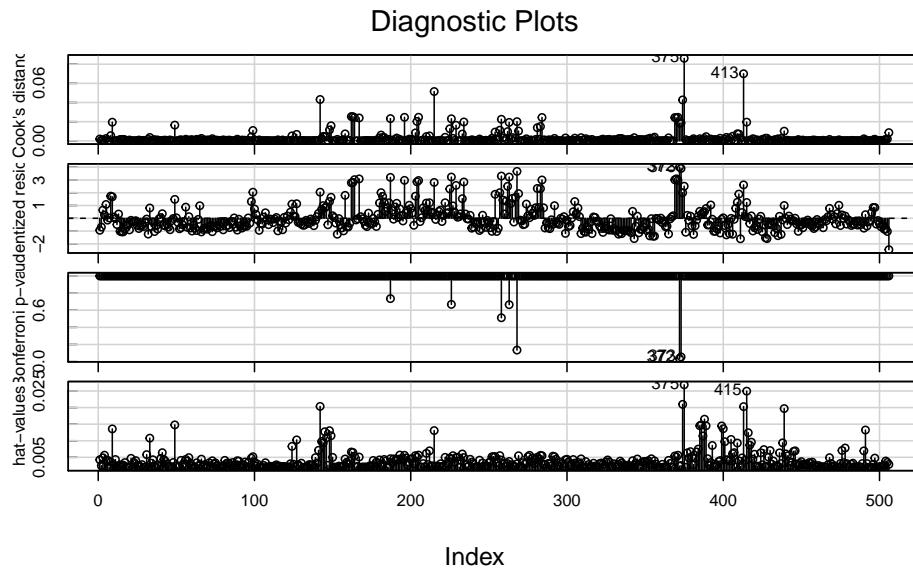
```
plot(hatvalues(lm.fit))
```



```
which.max(hatvalues(lm.fit))
```

```
## 375
## 375
influenceIndexPlot(lm.fit, id = 5)
```

```
## Warning in applyDefaults(id, defaults = list(method = "y", n = 2, cex =
## 1, : unnamed id arguments, will be ignored
```



6.3.2. Regresión lineal múltiple

Los modelos de correlación lineal múltiple requieren de las mismas condiciones que los modelos lineales simples más otras adicionales.

Para ajustar un modelo de regresión lineal múltiple usando mínimos cuadrados, utilizamos de nuevo la función `lm()`. La sintaxis `lm(y ~ x1 + x2 + x3)` se utiliza para ajustar un modelo con tres predictores, `x1`, `x2`, y `x3`. La función `summary()` produce ahora los coeficientes de regresión para todos los predictores.

```
ls.fit <- lm(medv ~ lstat + age, data = Boston)
summary(ls.fit)

##
## Call:
## lm(formula = medv ~ lstat + age, data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -15.981  -3.978  -1.283   1.968  23.158 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 33.22276   0.73085  45.458 < 2e-16 ***
## lstat        -1.03207   0.04819 -21.416 < 2e-16 ***
## age          0.03454   0.01223   2.826  0.00491 **  
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 6.173 on 503 degrees of freedom
## Multiple R-squared:  0.5513, Adjusted R-squared:  0.5495
## F-statistic: 309 on 2 and 503 DF, p-value: < 2.2e-16
```

El conjunto de datos Boston contiene 13 variables, por lo que sería engorroso tener que escribirlas todas para poder realizar una regresión utilizando todos los predictores. En su lugar, podemos utilizar la siguiente abreviatura:

```
ls.fit <- lm(medv ~ ., data = Boston)
summary(ls.fit)

##
## Call:
## lm(formula = medv ~ ., data = Boston)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -15.595 -2.730 -0.518  1.777 26.199
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.646e+01 5.103e+00 7.144 3.28e-12 ***
## crim        -1.080e-01 3.286e-02 -3.287 0.001087 **
## zn          4.642e-02 1.373e-02  3.382 0.000778 ***
## indus       2.056e-02 6.150e-02  0.334 0.738288
## chas        2.687e+00 8.616e-01  3.118 0.001925 **
## nox         -1.777e+01 3.820e+00 -4.651 4.25e-06 ***
## rm          3.810e+00 4.179e-01  9.116 < 2e-16 ***
## age         6.922e-04 1.321e-02  0.052 0.958229
## dis         -1.476e+00 1.995e-01 -7.398 6.01e-13 ***
## rad         3.060e-01 6.635e-02  4.613 5.07e-06 ***
## tax         -1.233e-02 3.760e-03 -3.280 0.001112 **
## ptratio     -9.527e-01 1.308e-01 -7.283 1.31e-12 ***
## black       9.312e-03 2.686e-03  3.467 0.000573 ***
## lstat      -5.248e-01 5.072e-02 -10.347 < 2e-16 ***
##
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.745 on 492 degrees of freedom
## Multiple R-squared:  0.7406, Adjusted R-squared:  0.7338
## F-statistic: 108.1 on 13 and 492 DF, p-value: < 2.2e-16
```

Podemos acceder a los componentes individuales de un objeto de resumen

por nombre (escriba `?summary.lm` para ver qué hay disponible). Por lo tanto `summary(lm.fit)$r.sq` nos da los R^2 , y `summary(lm.fit)$sigma` nos da hatsigma .

Si queremos realizar una regresión usando todas las variables pero excepto una, podemos eliminarla usando `-`. Por ejemplo, en la salida de regresión anterior, `age` tiene un alto valor p. Así que tal vez queramos hacer una regresión excluyendo este predictor. La siguiente sintaxis resulta en una regresión usando todos los predictores excepto `age`.

```
ls.fit1 <- lm(medv ~ . - age, data = Boston)
summary(ls.fit1)
```

```
##
## Call:
## lm(formula = medv ~ . - age, data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -15.6054 -2.7313 -0.5188  1.7601 26.2243 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 36.436927  5.080119  7.172 2.72e-12 ***
## crim        -0.108006  0.032832 -3.290 0.001075 **  
## zn          0.046334  0.013613  3.404 0.000719 ***  
## indus       0.020562  0.061433  0.335 0.737989    
## chas        2.689026  0.859598  3.128 0.001863 **  
## nox        -17.713540 3.679308 -4.814 1.97e-06 *** 
## rm          3.814394  0.408480  9.338 < 2e-16 ***
## dis         -1.478612  0.190611 -7.757 5.03e-14 *** 
## rad         0.305786  0.066089  4.627 4.75e-06 *** 
## tax         -0.012329  0.003755 -3.283 0.001099 **  
## ptratio     -0.952211  0.130294 -7.308 1.10e-12 *** 
## black       0.009321  0.002678  3.481 0.000544 *** 
## lstat      -0.523852  0.047625 -10.999 < 2e-16 ***
## ---        
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.74 on 493 degrees of freedom
## Multiple R-squared:  0.7406, Adjusted R-squared:  0.7343 
## F-statistic: 117.3 on 12 and 493 DF,  p-value: < 2.2e-16
```

6.3.2.1. Interacciones

Es fácil incluir términos de interacción en un modelo lineal usando la función `lm()`. La sintaxis `lstat:black` indica a R que incluya un término de interacción entre `lstat` y `black`. La sintaxis `lstat*age` incluye simultáneamente `lstat, age`, y el término de interacción `lstat × age` como predictores; es una abreviatura de `lstat + age + lstat:age`.

```
summary(lm(medv ~ lstat*age, data = Boston))

##
## Call:
## lm(formula = medv ~ lstat * age, data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -15.806  -4.045  -1.333   2.085  27.552 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 36.0885359  1.4698355 24.553 < 2e-16 ***
## lstat        -1.3921168  0.1674555 -8.313 8.78e-16 ***
## age          -0.0007209  0.0198792 -0.036  0.9711  
## lstat:age     0.0041560  0.0018518  2.244  0.0252 *  
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.149 on 502 degrees of freedom
## Multiple R-squared:  0.5557, Adjusted R-squared:  0.5531 
## F-statistic: 209.3 on 3 and 502 DF,  p-value: < 2.2e-16
```

6.3.2.2. Transformaciones no lineales para los predictores

La función `lm()` también puede acomodar transformaciones no lineales de los predictores. Por ejemplo, dado un predictor X podemos crear un predictor X^2 usando `I(X^2)`. La función `I()` es necesaria ya que `^` tiene un significado especial en una fórmula; envolviendo como lo hacemos permite el uso estándar en R, que es `I()` para elevar X a la potencia 2. Ahora realizamos una regresión de `medv` sobre `lstat` y `lstat2`.

```
lm.fit2 <- lm(medv ~ lstat + I(lstat^2), data = Boston)
summary(lm.fit2)

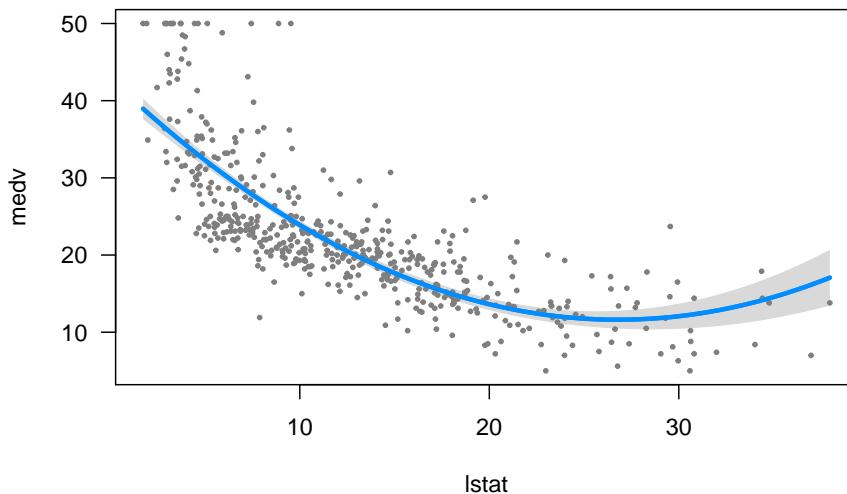
##
## Call:
## lm(formula = medv ~ lstat + I(lstat^2), data = Boston)
##
## Residuals:
```

```

##      Min      1Q Median      3Q      Max
## -15.2834 -3.8313 -0.5295  2.3095 25.4148
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 42.862007  0.872084  49.15 <2e-16 ***
## lstat      -2.332821  0.123803 -18.84 <2e-16 ***
## I(lstat^2)  0.043547  0.003745  11.63 <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.524 on 503 degrees of freedom
## Multiple R-squared:  0.6407, Adjusted R-squared:  0.6393
## F-statistic: 448.5 on 2 and 503 DF,  p-value: < 2.2e-16
# plot
visreg(lm.fit2)

```

-1.bb



El valor p cercano a cero asociado con el término cuadrático sugiere que conduce a un modelo mejorado. Usamos la función `anova()` para cuantificar aún más hasta qué punto el ajuste cuadrático es superior al ajuste lineal.

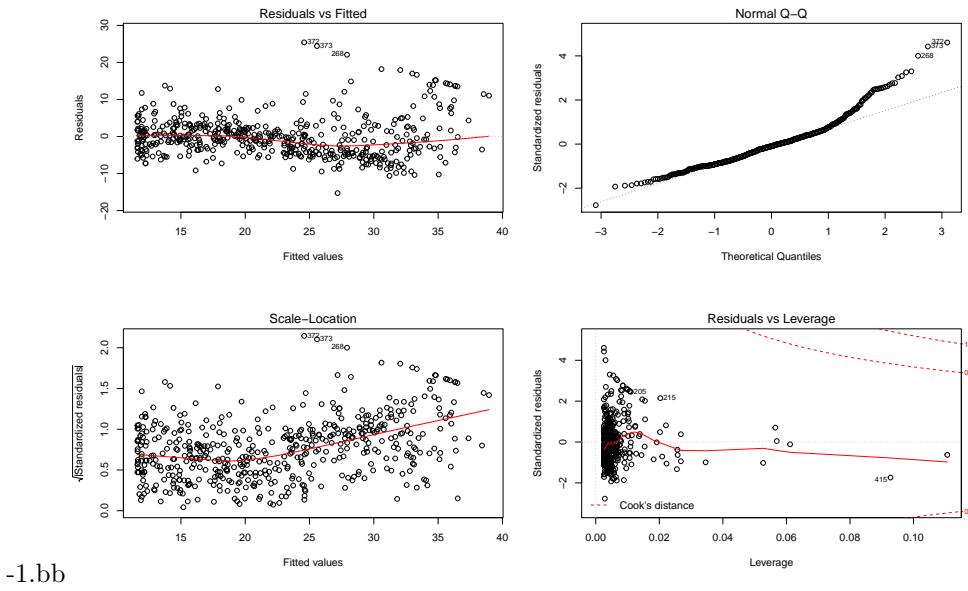
```
anova(lm.fit, lm.fit2)
```

102 CAPÍTULO 6. MODELOS LINEALES Y ANÁLISIS DE LA VARIANZA

```
## Analysis of Variance Table
##
## Model 1: medv ~ lstat
## Model 2: medv ~ lstat + I(lstat^2)
##   Res.Df   RSS Df Sum of Sq    F    Pr(>F)
## 1     504 19472
## 2     503 15347  1    4125.1 135.2 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Aquí el Modelo 1 (`lm.fit`) representa el submodelo lineal que contiene sólo un predictor, `lstat`, mientras que el Modelo 2 (`lm.fit2`) corresponde al modelo cuadrático más grande que tiene dos predictores, `lstat` y `lstat2`. La función `anova()` realiza una prueba de hipótesis comparando los dos modelos. La hipótesis nula es que los dos modelos se ajustan a los datos igualmente bien, y la hipótesis alternativa es que el modelo completo es superior. Aquí la estadística F es 135.1998221 y el valor p asociado es virtualmente cero. Esto proporciona una evidencia muy clara de que el modelo que contiene los predictores `lstat` y `lstat2` es muy superior al modelo que sólo contiene el predictor `lstat`. Esto no es sorprendente, ya que antes vimos evidencia de no linealidad en la relación entre `medv` y `lstat`. Si escribimos

```
par(mfrow = c(2,2))
plot(lm.fit2)
```



-1.bb

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

entonces vemos que cuando el término `lstat^2` se incluye en el modelo, hay poco patrón discernible en los residuos.

Para crear un ajuste cúbico, podemos incluir un predictor de la forma `I(X^3)`. Sin embargo, este enfoque puede empezar a ser engoroso para los polinomios de orden superior. Un mejor enfoque implica usar la función `poly()` para crear el polinomio dentro de `lm()`. Por ejemplo, el siguiente comando produce un ajuste polinómico de quinto orden:

```
lm.fit5 <- lm(medv ~ poly(lstat, 5), data = Boston)
summary(lm.fit5)

##
## Call:
## lm(formula = medv ~ poly(lstat, 5), data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -13.5433 -3.1039 -0.7052  2.0844 27.1153 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 22.5328    0.2318 97.197 < 2e-16 ***
## poly(lstat, 5)1 -152.4595    5.2148 -29.236 < 2e-16 ***
```

104 CAPÍTULO 6. MODELOS LINEALES Y ANÁLISIS DE LA VARIANZA

```

## poly(lstat, 5)2   64.2272    5.2148  12.316 < 2e-16 ***
## poly(lstat, 5)3 -27.0511    5.2148 -5.187 3.10e-07 ***
## poly(lstat, 5)4  25.4517    5.2148  4.881 1.42e-06 ***
## poly(lstat, 5)5 -19.2524    5.2148 -3.692 0.000247 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.215 on 500 degrees of freedom
## Multiple R-squared:  0.6817, Adjusted R-squared:  0.6785
## F-statistic: 214.2 on 5 and 500 DF,  p-value: < 2.2e-16

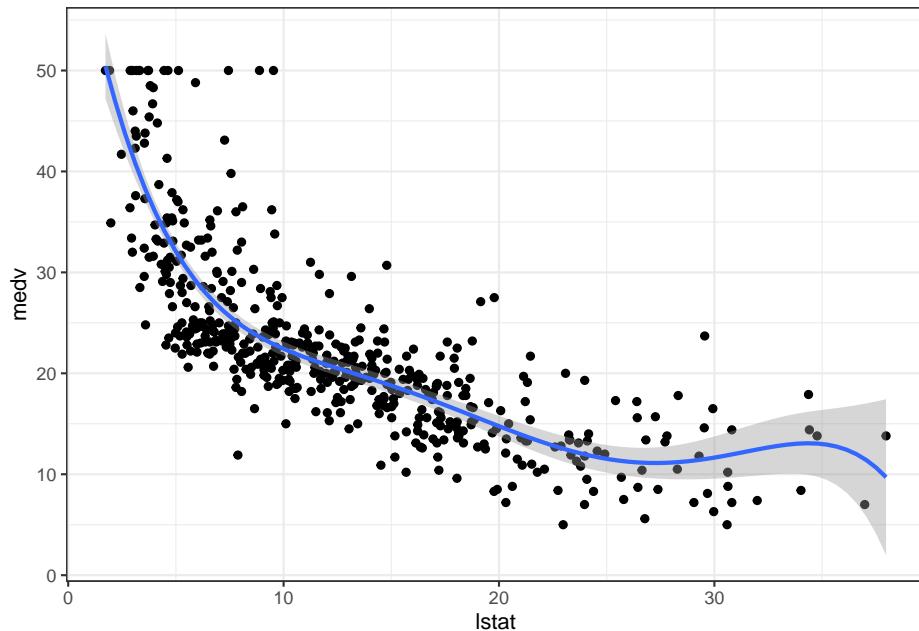
```

Esto sugiere que incluir términos polinómicos adicionales, hasta el quinto orden, conduce a una mejora en el ajuste del modelo. Sin embargo, una investigación adicional de los datos revela que ningún término polinómico más allá del quinto orden tiene valores p significativos en un ajuste de regresión.

```

library(ggplot2)
ggplot(data = Boston, aes(x = lstat, y = medv)) +
  geom_point() +
  theme_bw() +
  stat_smooth(method = "lm", formula = y ~ poly(x, 5))

```

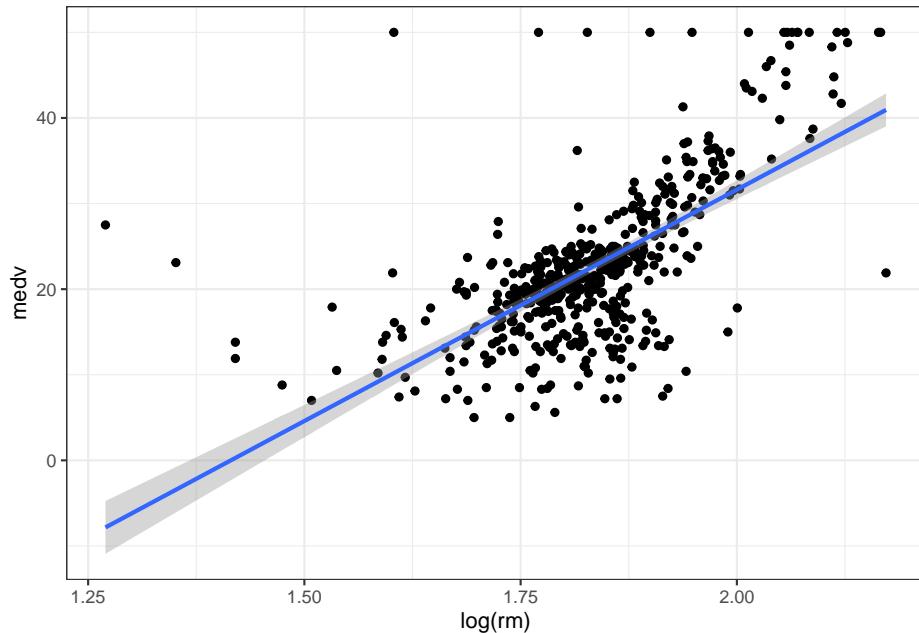


Por supuesto, no estamos restringidos a usar transformaciones polinómicas de

los predictores. Aquí probamos una transformación logarítmica.

```
summary(lm(medv ~ log(rm), data = Boston))
```

```
##  
## Call:  
## lm(formula = medv ~ log(rm), data = Boston)  
##  
## Residuals:  
##      Min       1Q   Median       3Q      Max  
## -19.487  -2.875  -0.104   2.837  39.816  
##  
## Coefficients:  
##              Estimate Std. Error t value Pr(>|t|)  
## (Intercept) -76.488     5.028 -15.21 <2e-16 ***  
## log(rm)      54.055     2.739  19.73 <2e-16 ***  
## ---  
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1  
##  
## Residual standard error: 6.915 on 504 degrees of freedom  
## Multiple R-squared:  0.4358, Adjusted R-squared:  0.4347  
## F-statistic: 389.3 on 1 and 504 DF,  p-value: < 2.2e-16  
ggplot(data = Boston, aes(x = log(rm), y = medv)) +  
  geom_point() +  
  theme_bw() +  
  stat_smooth(method = "lm")
```



6.3.3. Modelos con interacciones

Supongamos que el departamento de ventas de una empresa quiere estudiar la influencia que tiene la publicidad a través de distintos canales sobre el número de ventas de un producto. Se dispone de un conjunto de datos que contiene los ingresos (en millones) conseguido por ventas en 200 regiones, así como la cantidad de presupuesto, también en millones, destinado a anuncios por radio, TV y periódicos en cada una de ellas.

```
tv <- c(230.1, 44.5, 17.2, 151.5, 180.8, 8.7, 57.5, 120.2, 8.6, 199.8, 66.1,
214.7, 23.8, 97.5, 204.1, 195.4, 67.8, 281.4, 69.2, 147.3, 218.4, 237.4,
13.2, 228.3, 62.3, 262.9, 142.9, 240.1, 248.8, 70.6, 292.9, 112.9, 97.2,
265.6, 95.7, 290.7, 266.9, 74.7, 43.1, 228, 202.5, 177, 293.6, 206.9, 25.1,
175.1, 89.7, 239.9, 227.2, 66.9, 199.8, 100.4, 216.4, 182.6, 262.7, 198.9,
7.3, 136.2, 210.8, 210.7, 53.5, 261.3, 239.3, 102.7, 131.1, 69, 31.5, 139.3,
237.4, 216.8, 199.1, 109.8, 26.8, 129.4, 213.4, 16.9, 27.5, 120.5, 5.4,
116, 76.4, 239.8, 75.3, 68.4, 213.5, 193.2, 76.3, 110.7, 88.3, 109.8, 134.3,
28.6, 217.7, 250.9, 107.4, 163.3, 197.6, 184.9, 289.7, 135.2, 222.4, 296.4,
280.2, 187.9, 238.2, 137.9, 25, 90.4, 13.1, 255.4, 225.8, 241.7, 175.7,
209.6, 78.2, 75.1, 139.2, 76.4, 125.7, 19.4, 141.3, 18.8, 224, 123.1, 229.5,
87.2, 7.8, 80.2, 220.3, 59.6, 0.7, 265.2, 8.4, 219.8, 36.9, 48.3, 25.6,
273.7, 43, 184.9, 73.4, 193.7, 220.5, 104.6, 96.2, 140.3, 240.1, 243.2,
38, 44.7, 280.7, 121, 197.6, 171.3, 187.8, 4.1, 93.9, 149.8, 11.7, 131.7,
172.5, 85.7, 188.4, 163.5, 117.2, 234.5, 17.9, 206.8, 215.4, 284.3, 50,
164.5, 19.6, 168.4, 222.4, 276.9, 248.4, 170.2, 276.7, 165.6, 156.6, 218.5,
```

```

56.2, 287.6, 253.8, 205, 139.5, 191.1, 286, 18.7, 39.5, 75.5, 17.2, 166.8,
149.7, 38.2, 94.2, 177, 283.6, 232.1)
radio <- c(37.8, 39.3, 45.9, 41.3, 10.8, 48.9, 32.8, 19.6, 2.1, 2.6, 5.8, 24,
35.1, 7.6, 32.9, 47.7, 36.6, 39.6, 20.5, 23.9, 27.7, 5.1, 15.9, 16.9, 12.6,
3.5, 29.3, 16.7, 27.1, 16, 28.3, 17.4, 1.5, 20, 1.4, 4.1, 43.8, 49.4, 26.7,
37.7, 22.3, 33.4, 27.7, 8.4, 25.7, 22.5, 9.9, 41.5, 15.8, 11.7, 3.1, 9.6,
41.7, 46.2, 28.8, 49.4, 28.1, 19.2, 49.6, 29.5, 2, 42.7, 15.5, 29.6, 42.8,
9.3, 24.6, 14.5, 27.5, 43.9, 30.6, 14.3, 33, 5.7, 24.6, 43.7, 1.6, 28.5,
29.9, 7.7, 26.7, 4.1, 20.3, 44.5, 43, 18.4, 27.5, 40.6, 25.5, 47.8, 4.9,
1.5, 33.5, 36.5, 14, 31.6, 3.5, 21, 42.3, 41.7, 4.3, 36.3, 10.1, 17.2, 34.3,
46.4, 11, 0.3, 0.4, 26.9, 8.2, 38, 15.4, 20.6, 46.8, 35, 14.3, 0.8, 36.9,
16, 26.8, 21.7, 2.4, 34.6, 32.3, 11.8, 38.9, 0, 49, 12, 39.6, 2.9, 27.2,
33.5, 38.6, 47, 39, 28.9, 25.9, 43.9, 17, 35.4, 33.2, 5.7, 14.8, 1.9, 7.3,
49, 40.3, 25.8, 13.9, 8.4, 23.3, 39.7, 21.1, 11.6, 43.5, 1.3, 36.9, 18.4,
18.1, 35.8, 18.1, 36.8, 14.7, 3.4, 37.6, 5.2, 23.6, 10.6, 11.6, 20.9, 20.1,
7.1, 3.4, 48.9, 30.2, 7.8, 2.3, 10, 2.6, 5.4, 5.7, 43, 21.3, 45.1, 2.1,
28.7, 13.9, 12.1, 41.1, 10.8, 4.1, 42, 35.6, 3.7, 4.9, 9.3, 42, 8.6)
periodico <- c(69.2, 45.1, 69.3, 58.5, 58.4, 75, 23.5, 11.6, 1, 21.2, 24.2,
4, 65.9, 7.2, 46, 52.9, 114, 55.8, 18.3, 19.1, 53.4, 23.5, 49.6, 26.2, 18.3,
19.5, 12.6, 22.9, 22.9, 40.8, 43.2, 38.6, 30, 0.3, 7.4, 8.5, 5, 45.7, 35.1,
32, 31.6, 38.7, 1.8, 26.4, 43.3, 31.5, 35.7, 18.5, 49.9, 36.8, 34.6, 3.6,
39.6, 58.7, 15.9, 60, 41.4, 16.6, 37.7, 9.3, 21.4, 54.7, 27.3, 8.4, 28.9,
0.9, 2.2, 10.2, 11, 27.2, 38.7, 31.7, 19.3, 31.3, 13.1, 89.4, 20.7, 14.2,
9.4, 23.1, 22.3, 36.9, 32.5, 35.6, 33.8, 65.7, 16, 63.2, 73.4, 51.4, 9.3,
33, 59, 72.3, 10.9, 52.9, 5.9, 22, 51.2, 45.9, 49.8, 100.9, 21.4, 17.9,
5.3, 59, 29.7, 23.2, 25.6, 5.5, 56.5, 23.2, 2.4, 10.7, 34.5, 52.7, 25.6,
14.8, 79.2, 22.3, 46.2, 50.4, 15.6, 12.4, 74.2, 25.9, 50.6, 9.2, 3.2, 43.1,
8.7, 43, 2.1, 45.1, 65.6, 8.5, 9.3, 59.7, 20.5, 1.7, 12.9, 75.6, 37.9, 34.4,
38.9, 9, 8.7, 44.3, 11.9, 20.6, 37, 48.7, 14.2, 37.7, 9.5, 5.7, 50.5, 24.3,
45.2, 34.6, 30.7, 49.3, 25.6, 7.4, 5.4, 84.8, 21.6, 19.4, 57.6, 6.4, 18.4,
47.4, 17, 12.8, 13.1, 41.8, 20.3, 35.2, 23.7, 17.6, 8.3, 27.4, 29.7, 71.8,
30, 19.6, 26.6, 18.2, 3.7, 23.4, 5.8, 6, 31.6, 3.6, 6, 13.8, 8.1, 6.4, 66.2,
8.7)
ventas <- c(22.1, 10.4, 9.3, 18.5, 12.9, 7.2, 11.8, 13.2, 4.8, 10.6, 8.6, 17.4,
9.2, 9.7, 19, 22.4, 12.5, 24.4, 11.3, 14.6, 18, 12.5, 5.6, 15.5, 9.7, 12,
15, 15.9, 18.9, 10.5, 21.4, 11.9, 9.6, 17.4, 9.5, 12.8, 25.4, 14.7, 10.1,
21.5, 16.6, 17.1, 20.7, 12.9, 8.5, 14.9, 10.6, 23.2, 14.8, 9.7, 11.4, 10.7,
22.6, 21.2, 20.2, 23.7, 5.5, 13.2, 23.8, 18.4, 8.1, 24.2, 15.7, 14, 18,
9.3, 9.5, 13.4, 18.9, 22.3, 18.3, 12.4, 8.8, 11, 17, 8.7, 6.9, 14.2, 5.3,
11, 11.8, 12.3, 11.3, 13.6, 21.7, 15.2, 12, 16, 12.9, 16.7, 11.2, 7.3, 19.4,
22.2, 11.5, 16.9, 11.7, 15.5, 25.4, 17.2, 11.7, 23.8, 14.8, 14.7, 20.7,
19.2, 7.2, 8.7, 5.3, 19.8, 13.4, 21.8, 14.1, 15.9, 14.6, 12.6, 12.2, 9.4,
15.9, 6.6, 15.5, 7, 11.6, 15.2, 19.7, 10.6, 6.6, 8.8, 24.7, 9.7, 1.6, 12.7,
5.7, 19.6, 10.8, 11.6, 9.5, 20.8, 9.6, 20.7, 10.9, 19.2, 20.1, 10.4, 11.4,
10.3, 13.2, 25.4, 10.9, 10.1, 16.1, 11.6, 16.6, 19, 15.6, 3.2, 15.3, 10.1,
```

108 CAPÍTULO 6. MODELOS LINEALES Y ANÁLISIS DE LA VARIANZA

```
7.3, 12.9, 14.4, 13.3, 14.9, 18, 11.9, 11.9, 8, 12.2, 17.1, 15, 8.4, 14.5,
7.6, 11.7, 11.5, 27, 20.2, 11.7, 11.8, 12.6, 10.5, 12.2, 8.7, 26.2, 17.6,
22.6, 10.3, 17.3, 15.9, 6.7, 10.8, 9.9, 5.9, 19.6, 17.3, 7.6, 9.7, 12.8,
25.5, 13.4)
```

```
datos <- data.frame(tv, radio, periodico, ventas)
```

El modelo lineal múltiple que se obtiene empleando las variables `tv`, `radio` y `periodico` como predictores de ventas es el siguiente:

```
modelo <- lm(ventas ~ tv + radio + periodico, data = datos)
summary(modelo)
```

```
##
## Call:
## lm(formula = ventas ~ tv + radio + periodico, data = datos)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -8.8277 -0.8908  0.2418  1.1893  2.8292
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2.938889  0.311908  9.422   <2e-16 ***
## tv          0.045765  0.001395 32.809   <2e-16 ***
## radio        0.188530  0.008611 21.893   <2e-16 ***
## periodico   -0.001037  0.005871 -0.177    0.86
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.686 on 196 degrees of freedom
## Multiple R-squared:  0.8972, Adjusted R-squared:  0.8956
## F-statistic: 570.3 on 3 and 196 DF,  p-value: < 2.2e-16
```

De acuerdo al p-value obtenido para el coeficiente parcial de regresión de `periodico`, esta variable no contribuye de forma significativa al modelo. Como resultado de este análisis se concluye que las variables `tv` y `radio` están asociadas con la cantidad de ventas.

Con el comando `update()`, podemos actualizar el `modelo`. Por ejemplo:

```
modelo <- update(modelo, .~. -periodico)
summary(modelo)
```

```
##
## Call:
## lm(formula = ventas ~ tv + radio, data = datos)
##
```

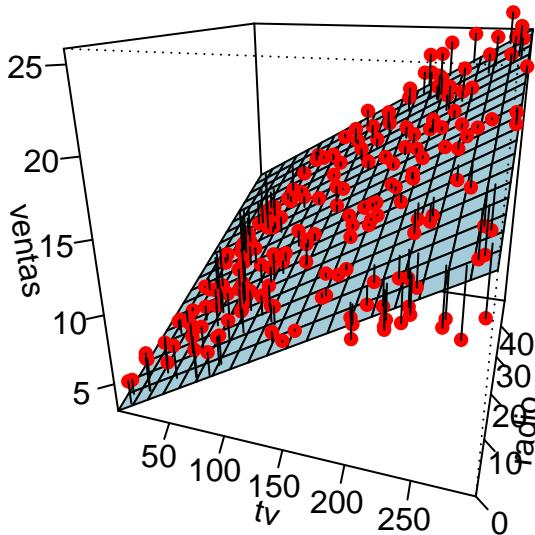
```

## Residuals:
##      Min    1Q Median    3Q   Max
## -8.7977 -0.8752  0.2422  1.1708 2.8328
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 2.92110   0.29449   9.919 <2e-16 ***
## tv          0.04575   0.00139  32.909 <2e-16 ***
## radio       0.18799   0.00804  23.382 <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.681 on 197 degrees of freedom
## Multiple R-squared:  0.8972, Adjusted R-squared:  0.8962 
## F-statistic: 859.6 on 2 and 197 DF,  p-value: < 2.2e-16

```

Al ser un modelo con dos predictores continuos se puede representar en 3D

Predicción ventas ~ TV y Radio



El modelo lineal a partir del cual se han obtenido las conclusiones asume que el efecto sobre las ventas debido a un incremento en el presupuesto de uno de los medios de comunicación es independiente del presupuesto gastado en los otros. Por ejemplo, el modelo lineal considera que el efecto promedio sobre las ventas debido a aumentar en una unidad el presupuesto de anuncios en TV es siempre de 0.04575, independientemente de la cantidad invertida en anuncios por radio. Sin embargo, la representación gráfica muestra que el modelo tiende a sobrevalorar las ventas cuando el presupuesto es muy alto en uno de los medios pero

110 CAPÍTULO 6. MODELOS LINEALES Y ANÁLISIS DE LA VARIANZA

muy bajo en el otro. Por contra, los valores de ventas predichos por el modelo están por debajo de las ventas reales cuando el presupuesto está repartido de forma equitativa entre ambos medios. Este comportamiento sugiere que existe interacción entre los predictores, por lo que el efecto de cada uno de ellos sobre la variable respuesta depende en cierta medida del valor que tome el otro predictor.

Tal y como se ha definido previamente, un modelo lineal con dos predictores sigue la ecuación:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$$

De acuerdo a esta definición, el incremento de una unidad en el predictor X_1 produce un incremento promedio de la variable Y de β_1 . Modificaciones en el predictor X_2 no alteran este hecho, y lo mismo ocurre con X_2 respecto a X_1 . Para que el modelo pueda contemplar la interacción se introduce un tercer predictor, que se construye con el producto de los predictores X_1 y X_2 .

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2 + \epsilon$$

La reorganización de los términos resulta en:

$$Y = \beta_0 + (\beta_1 + \beta_3 X_2) X_1 + \beta_2 X_2 + \epsilon$$

El efecto de X_1 sobre Y ya no es constante, sino que depende del valor que tome X_2 .

En R se puede introducir interacción entre predictores de dos formas, indicando los predictores individuales y entre cuales se quiere evaluar la interacción, o bien de forma directa.

```
modelo_interaccion <- lm(formula = ventas ~ tv + radio + tv:radio, data = datos)
summary(modelo_interaccion)

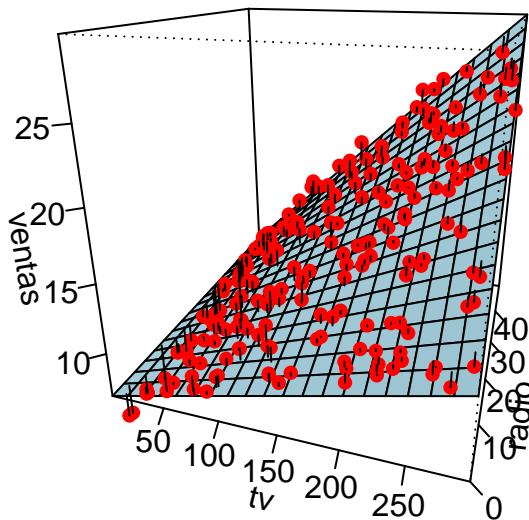
##
## Call:
## lm(formula = ventas ~ tv + radio + tv:radio, data = datos)
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -6.3366 -0.4028  0.1831  0.5948  1.5246 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 6.750e+00  2.479e-01 27.233   <2e-16 ***
## tv          1.910e-02  1.504e-03 12.699   <2e-16 ***
## radio        2.886e-02  8.905e-03  3.241   0.0014 **
```

```

## tv:radio    1.086e-03  5.242e-05  20.727    <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9435 on 196 degrees of freedom
## Multiple R-squared:  0.9678, Adjusted R-squared:  0.9673
## F-statistic:  1963 on 3 and 196 DF,  p-value: < 2.2e-16
#
# O tambien
# lm(formula = ventas ~ tv * radio, data = datos) es equivalente.

```

Predicción ventas ~ TV y Radio



Los resultados muestran una evidencia clara de que la interacción `tv:radio` es significativa y de que el modelo que incorpora la interacción (Adjusted R-squared = 0.9673) es superior al modelo que solo contemplaba el efecto de los predictores por separado (Adjusted R-squared = 0.8956).

Se puede emplear un ANOVA para realizar un test de hipótesis y obtener un p-value que evalúe la hipótesis nula de que ambos modelos se ajustan a los datos igual de bien.

```
anova(modelo, modelo_interaccion)
```

```

## Analysis of Variance Table
##
## Model 1: ventas ~ tv + radio
## Model 2: ventas ~ tv + radio + tv:radio
##   Res.Df   RSS Df Sum of Sq    F    Pr(>F)

```

112 CAPÍTULO 6. MODELOS LINEALES Y ANÁLISIS DE LA VARIANZA

```
## 1    197 556.91
## 2    196 174.48  1    382.43 429.59 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

En los modelos de regresión lineal múltiple que incorporan interacciones entre predictores hay que tener en cuenta que si se incorpora al modelo una interacción entre predictores, se deben incluir siempre los predictores individuales que participan en la interacción, independientemente de que su p-value sea significativo o no.

La interacción entre predictores no está limitada a predictores cuantitativos, también puede crearse interacción entre predictor cuantitativo y cualitativo.

Capítulo 7

Regresión logística

Una regresión logística se utiliza típicamente cuando hay una variable de resultado dicotómica (como ganar o perder), y una variable predictiva continua que está relacionada con la probabilidad o “odds” de la variable de resultado. También se puede utilizar con predictores categóricos y con múltiples predictores.

Si usamos una regresión lineal para modelar una variable dicotómica (como Y), es posible que el modelo resultante no restrinja los Y pronosticados dentro de 0 y 1. Además, otros supuestos de regresión lineal como la normalidad del error pueden ser violados. Así que en vez de eso, modelamos las probabilidades del evento de $\$log$ (logit), donde, p es la probabilidad del evento.

$$z_i = \ln\left(\frac{p_i}{1 - p_i}\right) = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p$$

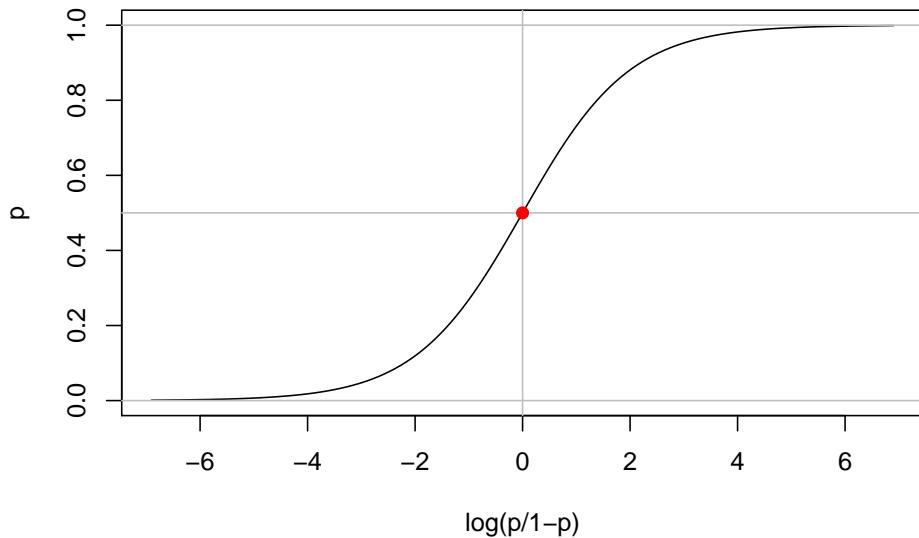
La ecuación anterior puede ser modelada usando `glm()` mediante el argumento `family="binomial`. Pero estamos más interesados en la probabilidad del evento que en las probabilidades logarítmicas del evento. Por lo tanto, los valores pronosticados del modelo anterior, es decir, las probabilidades logarítmicas del evento, pueden convertirse a probabilidad de evento de la siguiente manera:

$$p_i = 1 - \frac{1}{1 + \exp(z_i)}$$

A esto se le llama la *logit-inversa*, `?plogis`.

El siguiente gráfico relaciona `p` y `logit(p)`.

```
p <- seq(0,1,l=1000)
logitp <- log(p/(1-p))
plot(logitp,p,t='l',xlab="log(p/1-p)")
abline(h=c(0,0.5,1),v=0,col="grey")
points(0,0.5,pch=19,col="red")
```



7.1. Ejemplo: Datos de crédito en Alemania (*Credit scoring*)

Cuando un banco recibe una solicitud de préstamo, basada en el perfil del solicitante, el banco tiene que tomar una decisión sobre si procede o no con la aprobación del préstamo. Hay dos tipos de riesgos asociados a la decisión del banco:

- Si el solicitante tiene un buen riesgo de crédito, es decir, es probable que pague el préstamo, entonces no aprobar el préstamo a la persona resulta en una pérdida de negocio para el banco.
- Si el solicitante tiene un riesgo de crédito malo, es decir, no es probable que pague el préstamo, entonces la aprobación del préstamo a la persona resulta en una pérdida financiera para el banco.

El objetivo de este tipo de análisis es **minimizar el riesgo y maximizar el beneficio del banco**.

Para minimizar las pérdidas desde la perspectiva del banco, el banco necesita

7.1. EJEMPLO: DATOS DE CRÉDITO EN ALEMANIA (CREDIT SCORING)115

una regla de decisión sobre a quién dar la aprobación del préstamo y a quién no. Los perfiles demográficos y socioeconómicos de un solicitante son considerados por los administradores de préstamos antes de que se tome una decisión sobre su solicitud de préstamo.

Los datos German Credit Data contiene datos sobre 20 variables y la clasificación de si un solicitante es considerado un *Bueno* o un *Malo* riesgo de crédito para 1000 solicitantes de préstamos.

Se espera que un *modelo predictivo* elaborado a partir de estos datos sirva de guía al gerente del banco para tomar una decisión sobre la aprobación de un préstamo a un posible solicitante en función de su perfil.

Ver descripción (en inglés)

```
# German Credit Data
gcreditdata <- read.table("https://archive.ics.uci.edu/ml/machine-learning-databases/statlog/germancredit.data", header=TRUE)

colnames(gcreditdata)<-c("account.status","months",
                         "credit.history","purpose","credit.amount",
                         "savings","employment","installment.rate","personal.status",
                         "guarantors","residence","property","age","other.installments",
                         "housing","credit.cards","job","dependents","phone","foreign.worker","credit.ratio")

head(gcreditdata)

##   account.status months credit.history purpose credit.amount savings
## 1             A11      6           A34     A43        1169     A65
## 2             A12     48           A32     A43        5951     A61
## 3             A14     12           A34     A46        2096     A61
## 4             A11     42           A32     A42        7882     A61
## 5             A11     24           A33     A40        4870     A61
## 6             A14     36           A32     A46        9055     A65
##   employment installment.rate personal.status guarantors residence
## 1             A75            4          A93     A101         4
## 2             A73            2          A92     A101         2
## 3             A74            2          A93     A101         3
## 4             A74            2          A93     A103         4
## 5             A73            3          A93     A101         4
## 6             A73            2          A93     A101         4
##   property age other.installments housing credit.cards job dependents
## 1       A121  67           A143     A152        2 A173       1
## 2       A121  22           A143     A152        1 A173       1
## 3       A121  49           A143     A152        1 A172       2
## 4       A122  45           A143     A153        1 A173       2
```

```

## 5     A124 53          A143    A153      2 A173      2
## 6     A124 35          A143    A153      1 A172      2
##   phone foreign.worker credit.rating
## 1   A192           A201      1
## 2   A191           A201      2
## 3   A191           A201      1
## 4   A191           A201      1
## 5   A191           A201      2
## 6   A192           A201      1

```

Cambie el nombre de algunos niveles de factor:

```



```

Dividamos los datos en dos grupos (entrenamiento y prueba), 70 %/30 %.

```

n<- dim(gcreditdata)[1]

set.seed(1234) # select a random sample with
train <- sample(1:n , 0.7*n)

gcreditdata.test <- gcreditdata[-train,]
gcreditdata.train <- gcreditdata[train,]

ytrain <- gcreditdata$credit.rating[train]
ytest <- gcreditdata$credit.rating[-train]

```

Un modelo logístico puede ser ajustado con la función `glm`.

7.1. EJEMPLO: DATOS DE CRÉDITO EN ALEMANIA (CREDIT SCORING)117

```
m1 <- glm(credit.rating ~ . , family = binomial, data= gcreditdata.train)
summary(m1)

##
## Call:
## glm(formula = credit.rating ~ . , family = binomial, data = gcreditdata.train)
##
## Deviance Residuals:
##      Min        1Q     Median        3Q       Max
## -2.8353   -0.6178    0.3415    0.6729    2.0514
##
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)               5.259e-01  1.371e+00  0.384 0.701257
## account.status<200DM      3.532e-01  2.719e-01  1.299 0.193876
## account.status>200DM      8.198e-01  4.626e-01  1.772 0.076353 .
## account.statusNoStatus    1.487e+00  2.782e-01  5.344 9.08e-08 ***
## months                     -3.687e-02 1.156e-02 -3.189 0.001428 **
## credit.historyAllpaid      -1.347e-01 6.539e-01 -0.206 0.836779
## credit.historyAllpaidtillnow 5.296e-01 4.975e-01  1.065 0.287038
## credit.historyDelayinpaying 7.156e-01 5.575e-01  1.284 0.199266
## credit.historyCritical     1.439e+00  5.073e-01  2.837 0.004559 **
## purposecar(used)           1.838e+00  4.856e-01  3.786 0.000153 ***
## purposefurniture/equipment 1.095e+00  9.231e-01  1.186 0.235728
## purposeradio/television    8.879e-01  3.289e-01  2.700 0.006944 **
## purposedomesticappliances 9.928e-01  3.107e-01  3.196 0.001395 **
## purposerepairs              5.399e-01  9.006e-01  0.599 0.548881
## purposeeducation            -4.889e-01 6.587e-01 -0.742 0.457933
## purposevacation-doesnotexist? -1.800e-01 4.648e-01 -0.387 0.698616
## purposeretraining           2.093e+00  1.266e+00  1.653 0.098417 .
## purposebusiness              5.772e-01  3.985e-01  1.448 0.147489
## credit.amount                 -1.107e-04 5.456e-05 -2.028 0.042526 *
## savingsA62                   5.219e-01  3.548e-01  1.471 0.141326
## savingsA63                   8.722e-01  5.710e-01  1.528 0.126623
## savingsA64                   1.746e+00  6.533e-01  2.673 0.007514 **
## savingsA65                   1.354e+00  3.350e-01  4.042 5.30e-05 ***
## employmentA72                1.729e-01  5.327e-01  0.324 0.745566
## employmentA73                2.462e-01  5.116e-01  0.481 0.630356
## employmentA74                1.182e+00  5.660e-01  2.089 0.036736 *
## employmentA75                4.924e-02  5.076e-01  0.097 0.922712
## installment.rate              -4.157e-01 1.098e-01 -3.787 0.000152 ***
## personal.statusA92            5.279e-02  4.718e-01  0.112 0.910902
## personal.statusA93            6.791e-01  4.577e-01  1.484 0.137906
## personal.statusA94            2.017e-01  5.550e-01  0.363 0.716253
## guarantorsA102               1.880e-01  5.068e-01  0.371 0.710613
```

```

## guarantorsA103          1.024e+00  5.814e-01  1.761  0.078259 .
## residence                5.357e-02  1.047e-01  0.512  0.608926
## propertyA122            -4.618e-01 3.142e-01 -1.470  0.141584
## propertyA123            -4.679e-01 2.906e-01 -1.610  0.107366
## propertyA124            -7.642e-01 5.658e-01 -1.351  0.176834
## age                      1.872e-02  1.172e-02  1.598  0.110121
## other.installmentsA142   -1.259e-01 5.197e-01 -0.242  0.808648
## other.installmentsA143   3.400e-01  2.892e-01  1.176  0.239669
## housingA152              5.768e-01  2.935e-01  1.965  0.049384 *
## housingA153              7.825e-01  6.261e-01  1.250  0.211387
## credit.cards             -3.720e-01 2.456e-01 -1.515  0.129882
## jobA172                  -8.402e-01 8.307e-01 -1.011  0.311822
## jobA173                  -9.497e-01 7.970e-01 -1.192  0.233388
## jobA174                  -9.513e-01 8.262e-01 -1.152  0.249525
## dependents               -2.108e-01 3.080e-01 -0.685  0.493620
## phoneA192                 4.876e-01  2.556e-01  1.908  0.056400 .
## foreign.workerA202        9.540e-01  7.550e-01  1.264  0.206386
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 839.40  on 699  degrees of freedom
## Residual deviance: 596.11  on 651  degrees of freedom
## AIC: 694.11
##
## Number of Fisher Scoring iterations: 5

```

Variables significativas:

```

sig.var<- summary(m1)$coeff[-1,4] <0.01
names(sig.var)[sig.var == TRUE]

## [1] "account.statusNoStatus"      "months"
## [3] "credit.historyCritical"     "purposelcar(used)"
## [5] "purposeradio/television"    "purposedomesticappliances"
## [7] "savingsA64"                  "savingsA65"
## [9] "installment.rate"

```

Con `predict` podemos predecir con el modelo logístico el conjunto de test.

```

pred1<- predict.glm(m1,newdata = gcreditdata.test, type="response")
result1<- table(ytest, floor(pred1+0.5))
result1

```

```

## 
## ytest 0 1
## 0 48 51
## 1 25 176
error1<- sum(result1[1,2], result1[2,1])/sum(result1)
error1

## [1] 0.2533333

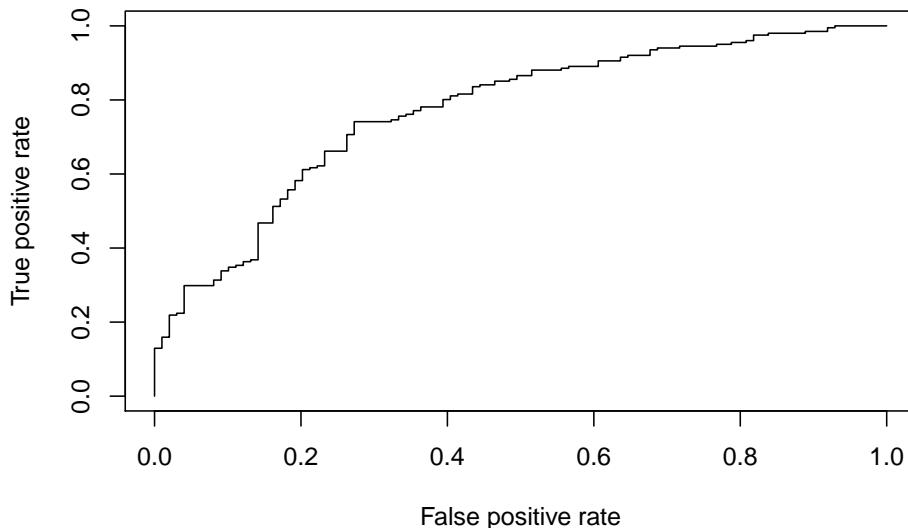
```

Curva ROC con la librería ROCR:

```

library(ROCR)
pred = ROCR::prediction(pred1,ytest)
perf <- performance(pred, "tpr", "fpr")
plot(perf)

```



```

AUCLog1=performance(pred, measure = "auc")@y.values[[1]]
cat("AUC: ",AUCLog1,"n")

## AUC: 0.7699382 n

```

7.2. Ejemplo: Predecir el salario de los trabajadores

Consideremos los datos `adult.csv`. Intentaremos predecir la variable de respuesta `ABOVE50k` (Salario >50k) mediante una regresión logística basada en variables demográficas explicativas.

```
inputData <- read.csv("http://idaejin.github.io/courses/R/data/adult.csv")
head(inputData)

##   AGE      WORKCLASS FNLWGT EDUCATION EDUCATIONNUM    MARITALSTATUS
## 1 39      State-gov  77516  Bachelors          13 Never-married
## 2 50  Self-emp-not-inc 83311  Bachelors          13 Married-civ-spouse
## 3 38        Private 215646 HS-grad            9 Divorced
## 4 53        Private 234721   11th            7 Married-civ-spouse
## 5 28        Private 338409 Bachelors          13 Married-civ-spouse
## 6 37        Private 284582  Masters          14 Married-civ-spouse
##             OCCUPATION RELATIONSHIP RACE    SEX CAPITALGAIN CAPITALLOSS
## 1      Adm-clerical Not-in-family White  Male    2174           0
## 2 Exec-managerial       Husband  White  Male           0           0
## 3 Handlers-cleaners Not-in-family White  Male           0           0
## 4 Handlers-cleaners       Husband Black  Male           0           0
## 5 Prof-specialty        Wife Black Female           0           0
## 6 Exec-managerial        Wife  White Female           0           0
##   HOURSPERWEEK NATIVECOUNTRY ABOVE50K
## 1          40 United-States     0
## 2          13 United-States     0
## 3          40 United-States     0
## 4          40 United-States     0
## 5          40      Cuba         0
## 6          40 United-States     0
```

Verificar sesgo de clase

Idealmente, la proporción de eventos y no eventos en la variable Y debería ser aproximadamente la misma. Por lo tanto, primero verifiquemos la proporción de clases en la variable dependiente ABOVE50K.

```
table(inputData$ABOVE50K)
```

```
##
##      0      1
## 24720 7841
```

Claramente, existe un sesgo de clase, una condición observada cuando la proporción de eventos es mucho menor que la proporción de no eventos. Por lo tanto, debemos muestrear las observaciones en proporciones aproximadamente iguales para obtener mejores modelos.

Crear Muestras de Entrenamiento y Pruebas

Una manera de abordar el problema del sesgo de clase es muestrear los 0 y 1 para los `trainingData` (muestra de entrenamiento) en proporciones iguales. Al hacerlo, pondremos el resto de los `inputData` no incluidos para la formación en `testData` (muestra de validación). Como resultado, el tamaño de la muestra de

entrenamiento será menor que el de la validación, lo que está bien, porque hay un gran número de observaciones (>10K).

```
# Create Training Data
input_ones <- inputData[which(inputData$ABOVE50K == 1), ] # all 1's
input_zeros <- inputData[which(inputData$ABOVE50K == 0), ] # all 0's

set.seed(100) # for repeatability of samples

input_ones_training_rows <- sample(1:nrow(input_ones), 0.7*nrow(input_ones)) # 1's for training
input_zeros_training_rows <- sample(1:nrow(input_zeros), 0.7*nrow(input_ones)) # 0's for training

# Pick as many 0's as 1's
training_ones <- input_ones[input_ones_training_rows, ]
training_zeros <- input_zeros[input_zeros_training_rows, ]
trainingData <- rbind(training_ones, training_zeros) # row bind the 1's and 0's

# Create Test Data
test_ones <- input_ones[-input_ones_training_rows, ]
test_zeros <- input_zeros[-input_zeros_training_rows, ]

testData <- rbind(test_ones, test_zeros) # row bind the 1's and 0's
```

Construir modelos de Logit y predecir

```
logitMod <- glm(ABOVE50K ~ RELATIONSHIP + AGE + CAPITALGAIN + OCCUPATION + EDUCATIONNUM, data=testData)

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
predicted <- plogis(predict(logitMod, testData)) # predicted scores
# or
predicted <- predict(logitMod, testData, type="response") # predicted scores
```

Cuando usamos la función de predicción en este modelo, predecirá las probabilidades de la variable Y . Para convertirlo en una probabilidad de predicción que esté entre 0 y 1, usamos el `plogis()`.

Decidir la probabilidad de corte de predicción óptima para el modelo.

El puntaje de probabilidad de la predicción de corte por defecto es de 0,5 o la proporción de 1's y 0's en los datos de entrenamiento. Pero a veces, afinar el corte de probabilidad puede mejorar la precisión tanto en las muestras de desarrollo como en las de validación. La función `InformationValue::optimalCutoff` proporciona formas de encontrar el punto de corte óptimo para mejorar la predicción de 1, 0, 1 y 0 y reducir el error de clasificación. Permite calcular la puntuación óptima que minimiza el error de clasificación para el modelo anterior.

```
library(InformationValue)
optCutOff <- optimalCutoff(testData$ABOVE50K, predicted)[1]
optCutOff
```

```
## [1] 0.89
```

Error de clasificación

El error de clasificación errónea es el desajuste porcentual de los valores predefinidos frente a los reales, independientemente de que sean 1 o 0. Cuanto menor sea el error de clasificación, mejor será su modelo.

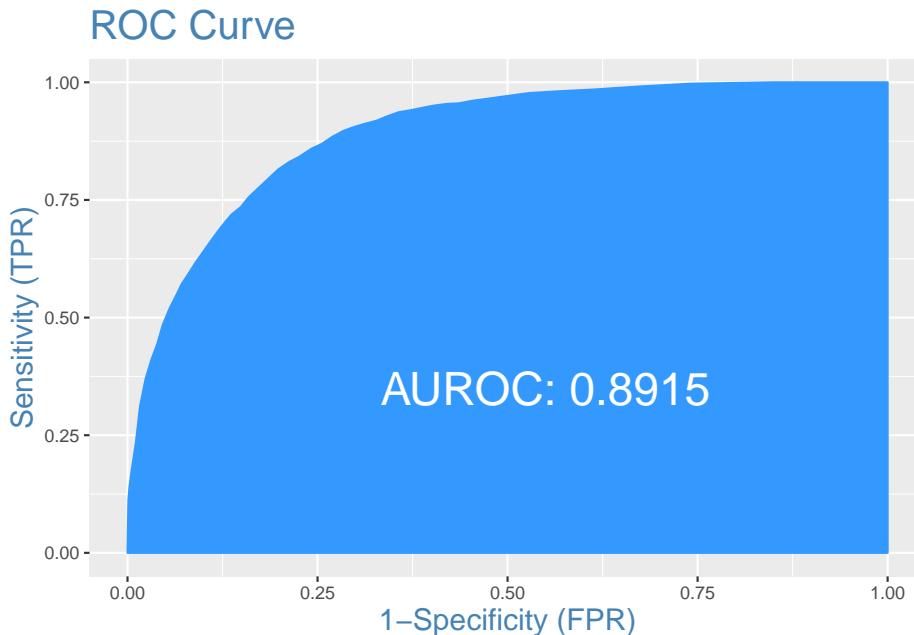
```
misClassError(testData$ABOVE50K, predicted, threshold = optCutOff)
```

```
## [1] 0.0892
```

ROC

Receiver Operating Characteristics a curva traza el porcentaje de verdaderos positivos pronosticados con precisión por un modelo logit dado a medida que la probabilidad de corte de la predicción se reduce de 1 a 0. Para un buen modelo, a medida que se reduce el corte, debería marcar más de 1 real como positivo y menos de 0 real como 1. Por lo tanto, para un buen modelo, la curva debería subir bruscamente, indicando que el TPR (eje Y) aumenta más rápido que el FPR (eje X) a medida que disminuye la puntuación de corte. Cuanto mayor sea el área bajo la curva ROC, mejor será la capacidad de predicción del modelo.

```
plotROC(testData$ABOVE50K, predicted)
```



Especificidad y sensibilidad

La **Sensibilidad** (o Tasa Verdaderos Positivos) es el porcentaje de 1's (reales) correctamente predichos por el modelo, mientras que, **especificidad** es el porcentaje de 0's (reales) correctamente predicho. La **especificidad** también puede calcularse como 1-Tasa Falsos Positivos.

$$\text{Sensitivity} = \frac{\#\text{Actual 1's and Predicted as 1's}}{\#\text{of Actual 1's}}$$

$$\text{Specificity} = \frac{\#\text{Actual 0's and Predicted as 0's}}{\#\text{of Actual 0's}}$$

```
sensitivity(testData$ ABOVE50K, predicted, threshold = optCutOff)
## [1] 0.3442414
specificity(testData$ ABOVE50K, predicted, threshold = optCutOff)
## [1] 0.9800853
```

Los números anteriores se calculan a partir de la muestra de validación que no se utilizó para la formación del modelo. Así que, una tasa de detección de la verdad de 34.42 % en los datos de prueba es bueno.

Matriz de Confusión Las columnas son reales, mientras que las filas son predicciones.

```
confusionMatrix(testData$ABOVE50K, predicted, threshold = optCutOff)

##      0     1
## 0 18849 1543
## 1   383  810
```

7.3. Ejemplo: Datos de los supervivientes del Titanic

El conjunto de datos es una colección de datos sobre algunos de los pasajeros, y el objetivo es predecir la supervivencia (1 si el pasajero sobrevivió o 0 si no lo hizo) basándose en algunas características como la clase de servicio, el sexo, la edad, etc. Como puede ver, vamos a utilizar variables categóricas y continuas.

VARIABLE DESCRIPTIONS:

pclass	Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
survival	Survival (0 = No; 1 = Yes)
name	Name
sex	Sex
age	Age
sibsp	Number of Siblings/Spouses Aboard
parch	Number of Parents/Children Aboard
ticket	Ticket Number
fare	Passenger Fare
cabin	Cabin
embarked	Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)
boat	Lifeboat
body	Body Identification Number
home_dest	Home/Destination

Descripción completa (Aquí)

Descarga los datos aquí

Leemos los datos de entreamiento `train` y `test`:

```
train <- read.csv('http://idaejin.github.io/courses/R/data/titanic_train.csv', header=TRUE)
test <- read.csv('http://idaejin.github.io/courses/R/data/titanic_test.csv', header=TRUE)
```

Questions:

- Ajustar un modelo logístico con `pclass` como variable explicativa. ¿Cuál es la interpretación del modelo ajustado?

- Encontrar el mejor modelo de regresión logística posible basado en todas las variables disponibles.

```

model <- glm(Survived ~ ., family=binomial(link='logit'), data=train)
summary(model)

## 
## Call:
## glm(formula = Survived ~ ., family = binomial(link = "logit"),
##      data = train)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.6064 -0.5954 -0.4254  0.6220  2.4165
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 5.137627  0.594998  8.635 < 2e-16 ***
## Pclass      -1.087156  0.151168 -7.192 6.40e-13 ***
## Sexmale     -2.756819  0.212026 -13.002 < 2e-16 ***
## Age         -0.037267  0.008195 -4.547 5.43e-06 ***
## SibSp       -0.292920  0.114642 -2.555  0.0106 *
## Parch       -0.116576  0.128127 -0.910  0.3629
## Fare        0.001528  0.002353  0.649  0.5160
## EmbarkedQ   -0.002656  0.400882 -0.007  0.9947
## EmbarkedS   -0.318786  0.252960 -1.260  0.2076
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 1065.39 on 799 degrees of freedom
## Residual deviance: 709.39 on 791 degrees of freedom
## AIC: 727.39
##
## Number of Fisher Scoring iterations: 5
anova(model, test="Chisq")

## Analysis of Deviance Table
## 
## Model: binomial, link: logit
## 
## Response: Survived
## 
## Terms added sequentially (first to last)
## 
```

```

##          Df Deviance Resid. Df Resid. Dev Pr(>Chi)
## NULL             799  1065.39
## Pclass      1   83.607    798   981.79 < 2.2e-16 ***
## Sex         1  240.014    797   741.77 < 2.2e-16 ***
## Age         1   17.495    796   724.28 2.881e-05 ***
## SibSp       1   10.842    795   713.43  0.000992 ***
## Parch       1    0.863    794   712.57  0.352873
## Fare        1    0.994    793   711.58  0.318717
## Embarked    2    2.187    791   709.39  0.334990
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
mod1 <- glm(Survived ~ as.factor(Pclass), family=binomial, data=train)
summary(mod1)

## 
## Call:
## glm(formula = Survived ~ as.factor(Pclass), family = binomial,
##      data = train)
##
## Deviance Residuals:
##      Min      1Q      Median      3Q      Max
## -1.3787 -0.7515 -0.7515  0.9887  1.6747
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.4616    0.1474   3.131  0.00174 **
## as.factor(Pclass)2 -0.5455    0.2138  -2.551  0.01074 *
## as.factor(Pclass)3 -1.5816    0.1844  -8.575 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 1065.39 on 799 degrees of freedom
## Residual deviance: 979.94 on 797 degrees of freedom
## AIC: 985.94
##
## Number of Fisher Scoring iterations: 4
anova(mod1,test="Chisq")

## Analysis of Deviance Table
##
## Model: binomial, link: logit
##

```

```

## Response: Survived
##
## Terms added sequentially (first to last)
##
##
##          Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL             799    1065.39
## as.factor(Pclass) 2     85.452      797    979.94 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

efecto de interacción entre la clase de pasajero y el sexo, ya que la clase de pasajero mostró una diferencia mucho mayor en la tasa de supervivencia entre las mujeres en comparación con los hombres (es decir, las mujeres de clase superior tenían muchas más probabilidades de sobrevivir que las mujeres de clase inferior, mientras que los hombres de primera clase tenían más probabilidades de sobrevivir que los hombres de segunda o tercera clase, pero no por el mismo margen que las mujeres).

```
mod2 <- glm(Survived ~ Pclass + Sex + Age + SibSp, family = binomial(logit), data = train)
summary(mod2)
```

```

##
## Call:
## glm(formula = Survived ~ Pclass + Sex + Age + SibSp, family = binomial(logit),
##      data = train)
##
## Deviance Residuals:
##      Min        1Q        Median         3Q        Max
## -2.6595   -0.6125   -0.4247    0.6149    2.4302
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  5.05604   0.50130 10.086 < 2e-16 ***
## Pclass       -1.14391   0.12585 -9.089 < 2e-16 ***
## Sexmale     -2.75564   0.20471 -13.461 < 2e-16 ***
## Age         -0.03725   0.00812 -4.588 4.48e-06 ***
## SibSp        -0.33075   0.10892 -3.037  0.00239 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 1065.39 on 799 degrees of freedom
## Residual deviance: 713.43 on 795 degrees of freedom
## AIC: 723.43
##
```

```

## Number of Fisher Scoring iterations: 5
anova(mod2,test="Chisq")

## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: Survived
##
## Terms added sequentially (first to last)
##
##
##          Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL           799    1065.39
## Pclass      1   83.607     798    981.79 < 2.2e-16 ***
## Sex         1   240.014     797    741.77 < 2.2e-16 ***
## Age         1    17.495     796    724.28 2.881e-05 ***
## SibSp       1    10.842     795    713.43  0.000992 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

mod3 <- glm(Survived ~ Pclass + Sex + Pclass:Sex + Age + SibSp, family = binomial(logit))
summary(mod3)

##
## Call:
## glm(formula = Survived ~ Pclass + Sex + Pclass:Sex + Age + SibSp,
##      family = binomial(logit), data = train)
##
## Deviance Residuals:
##      Min        1Q     Median        3Q       Max
## -3.1993  -0.6265  -0.4770   0.4485   2.3093
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) 7.606411  0.960804  7.917 2.44e-15 ***
## Pclass      -2.108360  0.316024 -6.672 2.53e-11 ***
## Sexmale     -5.887480  0.920417 -6.397 1.59e-10 ***
## Age        -0.038063  0.008498 -4.479 7.50e-06 ***
## SibSp      -0.310269  0.109370 -2.837 0.004556 **
## Pclass:Sexmale 1.254202  0.338241  3.708 0.000209 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##

```

```

##      Null deviance: 1065.39 on 799 degrees of freedom
## Residual deviance:  695.66 on 794 degrees of freedom
## AIC: 707.66
##
## Number of Fisher Scoring iterations: 6
anova(mod3,test="Chisq")

## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: Survived
##
## Terms added sequentially (first to last)
##
##
##          Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL            799    1065.39
## Pclass          1     83.607    798    981.79 < 2.2e-16 ***
## Sex             1    240.014    797    741.77 < 2.2e-16 ***
## Age             1    17.495    796    724.28 2.881e-05 ***
## SibSp           1    10.842    795    713.43  0.000992 ***
## Pclass:Sex      1    17.779    794    695.66 2.481e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

En los pasos anteriores, evaluamos brevemente el ajuste del modelo, ahora nos gustaría ver cómo lo está haciendo el modelo al predecir y en un nuevo conjunto de datos. Estableciendo el parámetro `type='response'`, R producirá probabilidades en la forma de $P(y = 1|X)$. Nuestro límite de decisión será de 0,5.

Si $P(y = 1|X) > 0,5$ entonces $y = 1$ en caso contrario $y = 0$. Tener en cuenta que para algunas aplicaciones diferentes límites de decisión podría ser una mejor opción.

```

fitted.results <- predict(mod3,newdata=test,type='response')
fitted.results <- ifelse(fitted.results > 0.5,1,0)

misClasificError <- mean(fitted.results != test$Survived)
print(paste('Accuracy',1-misClasificError))

## [1] "Accuracy 0.8075"

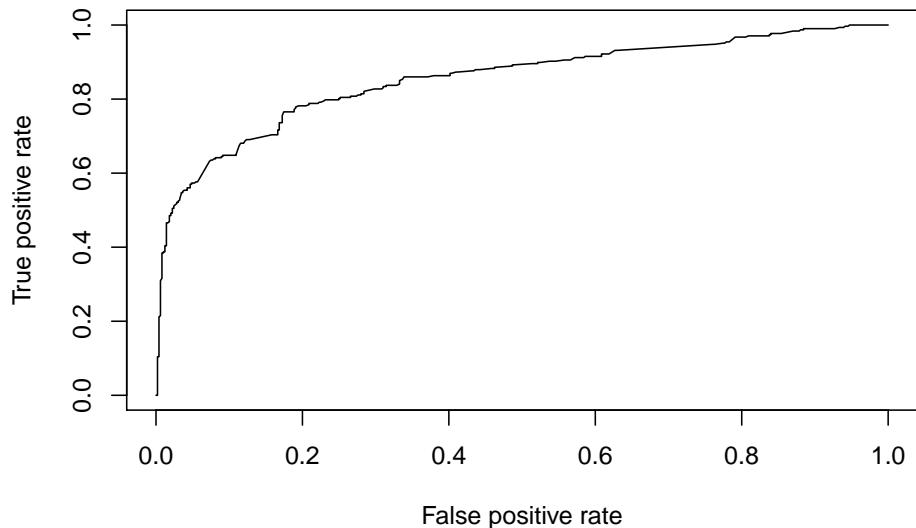
```

La precisión de 0.8075 en el conjunto de test es un buen resultado. Sin embargo, hay que tener en cuenta que este resultado depende en cierta medida de la división manual de los datos que hice anteriormente, por lo tanto, si deseamos una puntuación más precisa, sería mejor realizar algún tipo de validación cruzada,

como la validación cruzada k-fold.

Evaluar la capacidad predictiva

```
library(ROCR)
p <- predict(mod3, newdata=subset(test,select=c(2,3,4,5,6,7,8)), type="response")
pr <- ROCR::prediction(p, test$Survived)
prf <- performance(pr, measure = "tpr", x.measure = "fpr")
plot(prf)
```



```
auc <- performance(pr, measure = "auc")
auc <- auc@y.values[[1]]
auc
```

```
## [1] 0.8543155
```

Capítulo 8

Modelos Aditivos Generalizados

8.1. Suavizado

Suavizado Scatterplot

Consiste en resaltar la *tendencia subyacente* en los datos. La tendencia subyacente sería una función como:

$$f(x) = \mathbb{E}(y|x)$$

que también se puede escribir como:

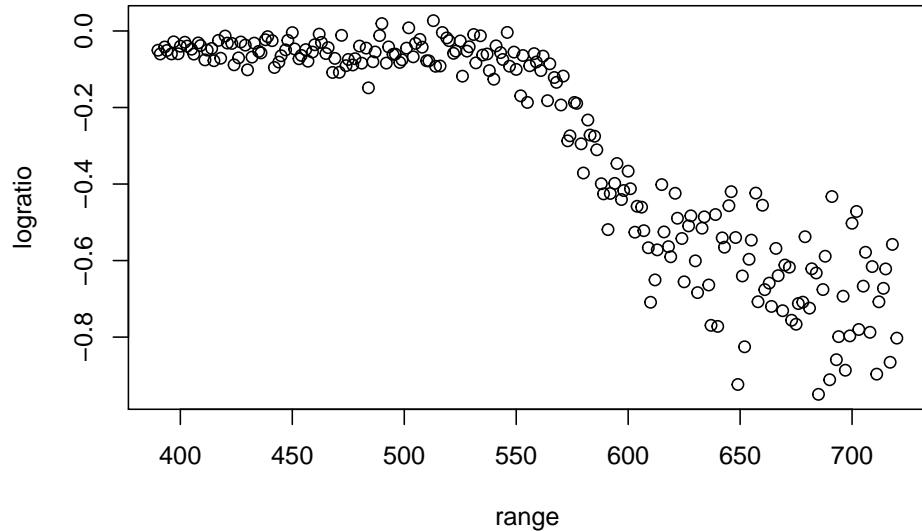
$$y_i = f(x_i) + \epsilon_i, \quad \mathbb{E}(\epsilon_i) = 0,$$

en cuyo caso el problema se denomina a menudo como *regresión no-paramétrica*, donde $f(\cdot)$ es una función *suave* no conocida que se estima a partir de los datos (x_i, y_i)

Existen varios métodos para suavizar una gráfica de dispersión, incluyendo *splines*, *regresión kernel*, *medias móviles*, *loess* o su versión ponderada *lowess*.

El conjunto de datos `lidar` consta de 221 observaciones de un experimento de detección y alcance de luz (LIDAR).

```
library(SemiPar)
#library(car)
data(lidar)
plot(logratio~range, data=lidar)
```

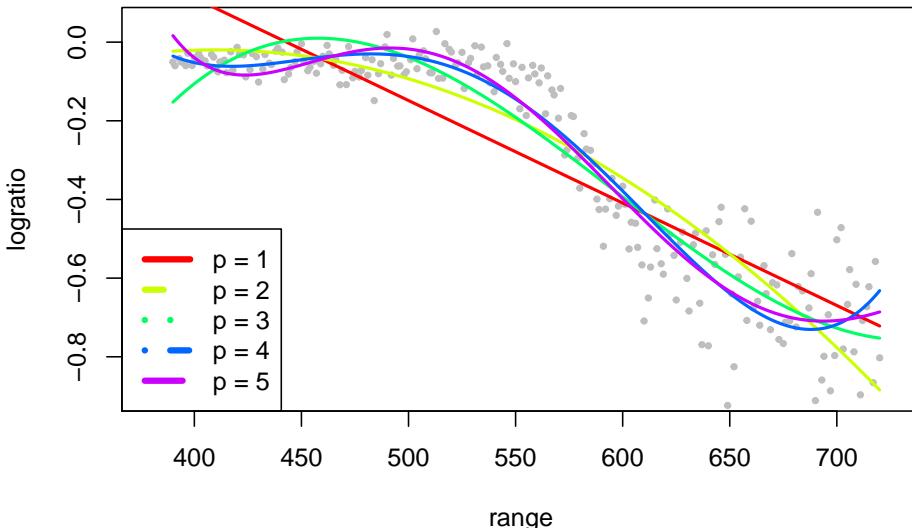


8.1.1. Regresión polinomial

Consiste en transformar la variable explicativa X , incluyendo polinomios de grado p tales que:

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_p X^p + \epsilon \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

```
library(lattice)
cols. <- rainbow(5)
range.seq <- seq(min(lidar$range),max(lidar$range),l=200)
plot(logratio~range,data=lidar,cex=.5,pch=19,col="grey",xlim=c(380,725),ylim=c(-0.90,0)
for(i in 1:5){
  lines(range.seq,predict(lm(logratio~poly(range,i),data=lidar),data.frame(range=range
})
legend("bottomleft",c("p = 1","p = 2","p = 3","p = 4","p = 5"),col=cols.,lty=1:5,lwd=4)
```

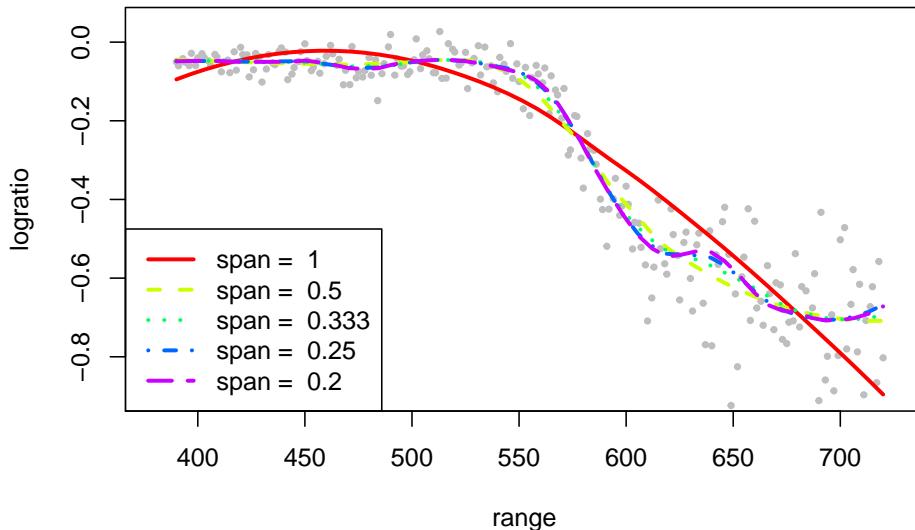


Podemos tratar estas relaciones no lineales de manera efectiva a través de técnicas más flexibles.

Regresión local:

LOESS y LOWESS son dos métodos de regresión no-paramétrica muy relacionados que combinan múltiples modelos de regresión en un modo basado en k-neast-neighbor.

```
attach(lidar)
span <- 1/c(1,2,3,4,5)
plot(logratio~range,data=lidar,cex=.5,pch=19,col="grey",xlim=c(380,725),ylim=c(-0.90,0.05))
cols <- rainbow(length(span))
for(i in 1:length(span)){
  lines(loess.smooth(range,logratio,span=span[i], degree=2),col=cols.[i],lwd=2.5,lty=i)
}
legend("bottomleft",paste("span = ",round(span,3)),col=cols.,lwd=2.5,lty=1:6)
```



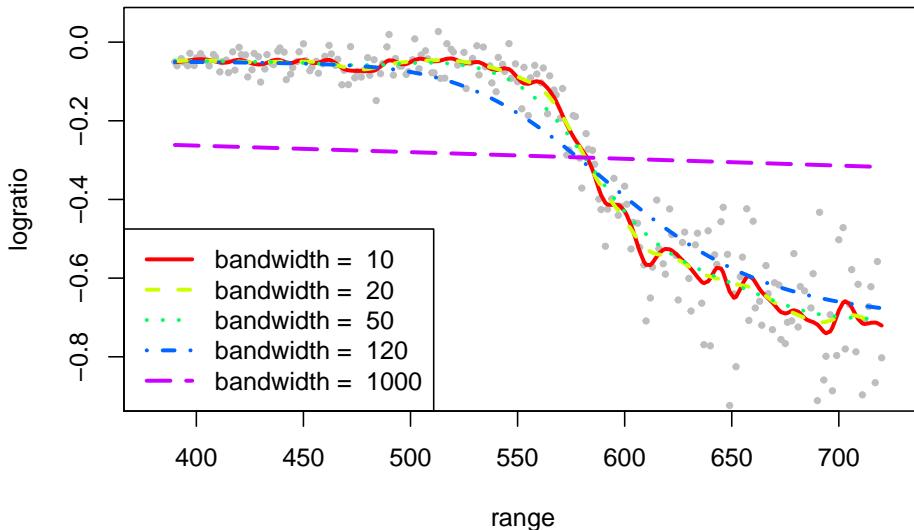
Un suavizado *kernel* es de la forma:

$$\hat{y}_i = \frac{\sum_{j=1}^n y_j K\left(\frac{x_i - x_j}{b}\right)}{\sum_{j=1}^n K\left(\frac{x_i - x_j}{b}\right)},$$

donde b es un parámetro de *ancho de banda (bandwidth)*, y K una función del núcleo/*kernel*, como en la estimación de densidad. Hay diferentes opciones para el kernel K (por ejemplo: Gaussian), el parámetro de elección crítico es el ancho de banda b .

En R la función `ksmooth` en `library(stats)` permite el suavizado tipo kernel (otras opciones son `locompoly` en `library(KernSmooth)`)

```
attach(lidar)
b <- c(10, 20, 50, 120, 1000)
plot(logratio~range,data=lidar,cex=.5,pch=19,col="grey",xlim=c(380,725),ylim=c(-0.90,0
cols. <- rainbow(length(b))
for(i in 1:length(b)){
  lines(ksmooth(range,logratio,x.points=range,"normal", bandwidth=b[i]),col=cols.[i],lwd=2.5,lty=1:5)
}
legend("bottomleft",paste("bandwidth = ",b),col=cols.,lwd=2.5,lty=1:5)
```



8.2. GAMs

Los GAMs (del inglés *generalized additive models*) son una generalización de los GLMs para incorporar formas no lineales de los predictores (plines, Polinomios, o funciones Step, etc....).

$$y \sim \text{ExpoFam}(\mu, \sigma^2, \dots)$$

$$\mu = \mathbb{E}[y]$$

$$g(\mu) = \eta = \beta_0 + f(x_1) + f(x_2) + \dots + f(x_p)$$

Ahora usamos las funciones *suaves* $f(\cdot)$ de nuestras variables predictoras, que pueden tomar formas más flexibles. Se supone que los valores observados son de alguna distribución de la familia exponencial, y μ está relacionado con los predictores a través de una función *enlace* (*o link*).

En este curso se estudiará la implementación de GAMs en R usando el paquete `mgcv` y funciones de base tipo `spline`.

Cargaremos la librería `mgcv`, y la función `gam`:

La función `gam`

La sintaxis es:

```
library(mgcv)
fit.gam <- gam(y ~ s(x))
```

En general, la sintaxis es

```
gam(formula,method="...",select="...",family=gaussian())
```

Los argumentos principales para esta función se deben especificar como una `formula`:

- La primera opción es la base utilizada para representar los términos suaves `s(x)` (ver `?s` o `?smooth.terms`). El tipo de función base se puede modificar utilizando `bs` dentro de `s(x,bs="ps")`:

El resto de argumentos es:

- `m` El orden de la penalización.
- `k` la dimensión de la base utilizada para representar el término de suavizado. `k` debe ser mayor que `m`.
- `by` una variable numérica o factorial de la misma dimensión que cada covariante.
- `sp` cualquier parámetro de suavizado suministrado para el término de suavizado.
- etc ...

```
summary(fit.gam)
```

`gam.check` produce algunos gráficos de residuos básicos, y proporciona información relacionada con el proceso de ajuste.

8.2.1. Ejemplo: Salarios

Salarios y otros datos para un grupo de 3.000 trabajadores varones en la región del Atlántico Medio.

```
#ISLR package contains the 'Wage' Dataset
require(ISLR)
attach(Wage) #Mid-Atlantic Wage Data
?Wage # To search more on the dataset

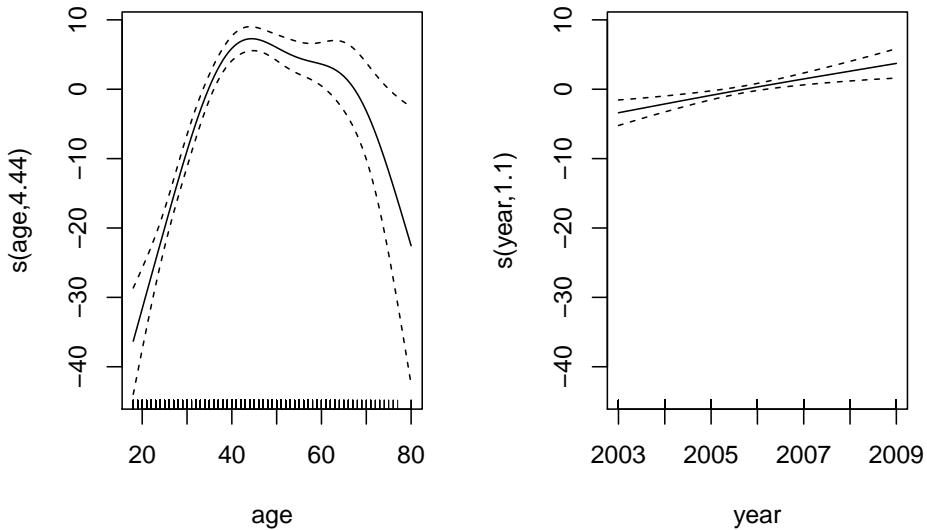
library(mgcv)
gam1 <- gam(wage~s(age,k=6)+s(year,k=6)+education,data = Wage)
summary(gam1)

##
## Family: gaussian
## Link function: identity
##
## Formula:
## wage ~ s(age, k = 6) + s(year, k = 6) + education
##
## Parametric coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                85.456     2.153   39.692 < 2e-16 ***
##
```

```

## education2. HS Grad      10.978    2.428   4.521  6.4e-06 ***
## education3. Some College 23.530    2.558   9.197  < 2e-16 ***
## education4. College Grad 38.159    2.543  15.005  < 2e-16 ***
## education5. Advanced Degree 62.559    2.760  22.668  < 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(age)  4.435  4.840 46.69 < 2e-16 ***
## s(year) 1.101  1.195 11.11 0.000379 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.29 Deviance explained = 29.2%
## GCV = 1240.7 Scale est. = 1236.3 n = 3000
par(mfrow=c(1,2)) #to partition the Plotting Window
plot(gam1, se = TRUE)

```

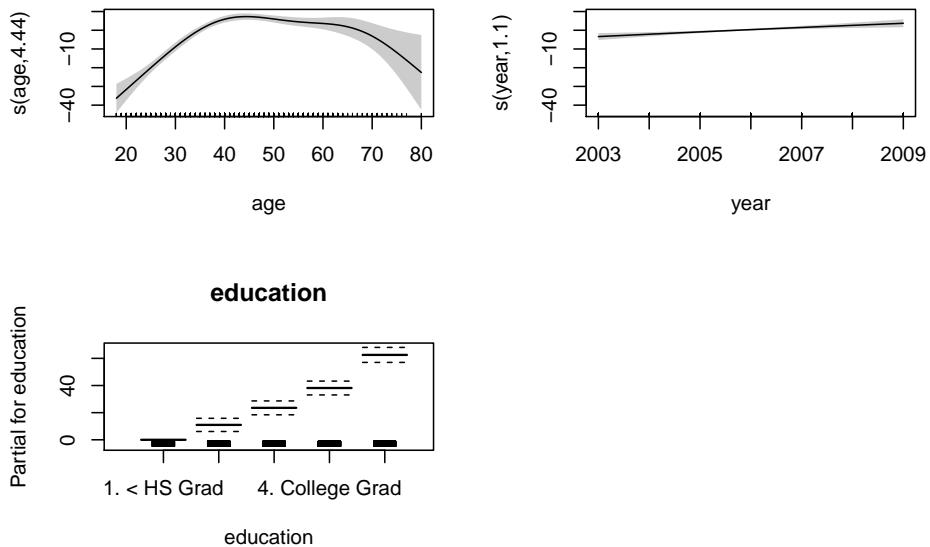


Con `all.terms = TRUE` podemos representar la parte paramétrica:

```

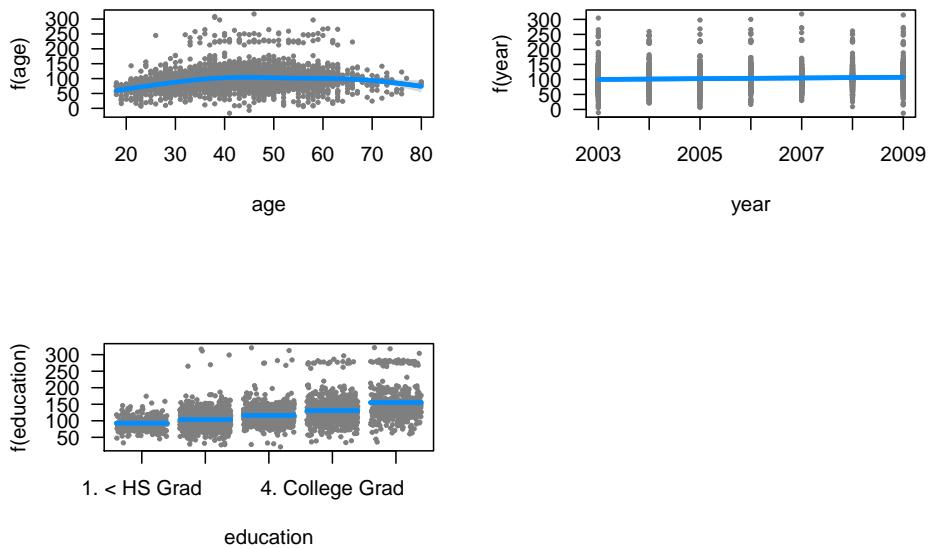
par(mfrow=c(2,2))
plot(gam1, all.terms=TRUE, scheme=1)

```



Recordemos que la librería `visreg` permite visualizar modelos de regresión:

```
library(visreg)
par(mfrow=c(2,2))
visreg(gam1)
```



8.3. Modelos semi-paramétricos

Los modelos semi-paramétricos permiten combinar componentes lineales (paramétricos) y no paramétricos, por ejemplo:

$$\mathbf{y} = \beta_0 + \beta_1 \mathbf{x}_1 + \dots + \beta_{j-1} \mathbf{x}_{j-1} + f(\mathbf{x}_j) + \epsilon$$

El procedimiento de ajuste es el mismo que se muestra en las secciones anteriores. Simplemente los efectos paramétricos están incluidos en la parte de efectos fijos, \mathbf{X} . Un caso interesado es cuando la parte paramétrica incluye una variable factorial con dos o más niveles. Como vimos en el caso lineal, podemos considerar diferentes situaciones: términos suaves paralelos (efecto aditivo) o términos suaves no paralelos (efecto de interacción). Además, podemos considerar la misma cantidad de alisamiento o una cantidad diferente.

Datos de cebollas

Para ilustrar un caso simple de un modelo semi-paramétrico, consideramos los `data(onions)` en la `library(SemiPar)`. Los datos consisten en 84 observaciones de un experimento que involucra la producción de cebollas en dos localidades del sur de Australia. Las variables son:

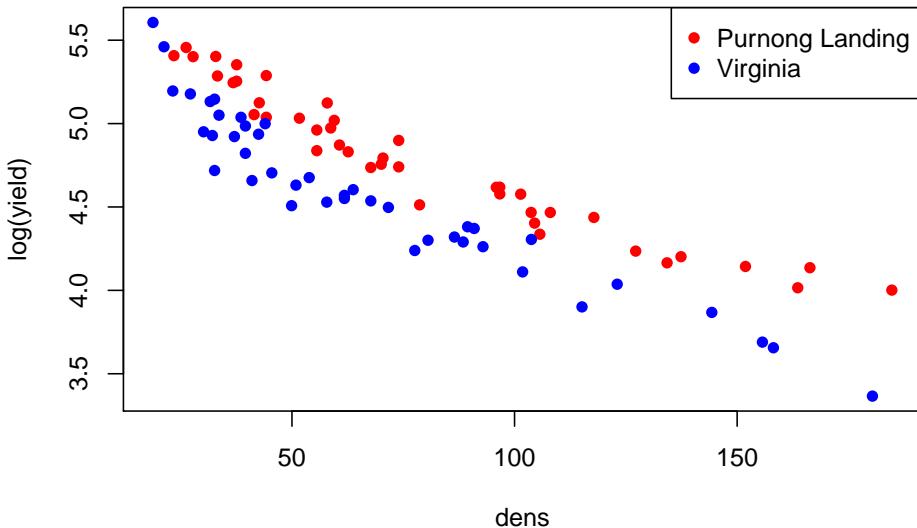
- `dens` densidad de área de las plantas (plantas por metro cuadrado)
- `yield` rendimiento de la cebolla (gramos por planta).
- `location` localización: 0=Purnong Landing, 1=Virginia.

La siguiente figura muestra que para una mayor densidad de plantas el rendimiento de Purnong Landing de onion es mayor.

```
library(SemiPar)
data(onions)
attach(onions)

## The following objects are masked from onions (pos = 11):
##
##     dens, location, yield

## The following objects are masked from onions (pos = 28):
##
##     dens, location, yield
points.cols <- c("red","blue")
plot(dens,log(yield),col=points.cols[location+1],pch=16)
legend("topright",c("Purnong Landing","Virginia"),col=points.cols,pch=rep(16,2))
```



El modelo lineal es:

$$\log(\text{yield}_i) = \beta_0 + \beta_1 \text{location}_{ij} + \beta_2 \text{dens}_i + \epsilon_i$$

donde

$$\text{location}_{ij} = \begin{cases} 0 & \text{si la } i\text{-ésima observación es de Purnong Landing} \\ 1 & \text{si la } i\text{-ésima observación es de Virginia} \end{cases}$$

La figura sugiere un término no lineal para `dens`, por lo tanto, un modelo mejor sería:

$$\log(\text{yield}_i) = \beta_0 + \beta_1 \text{location}_{ij} + f(\text{dens}_i) + \epsilon_i$$

```
# create a factor for location
L <- factor(location)

levels(L) <- c("Purnong Landing", "Virginia")

# fit a gam with location factor
fit1 <- gam(log(yield) ~ L + s(dens, k=20, m=2, bs="ps"),
             method="REML", select=TRUE)

summary(fit1)

## 
## Family: gaussian
## Link function: identity
```

```

## 
## Formula:
## log(yield) ~ L + s(dens, k = 20, m = 2, bs = "ps")
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 4.85011   0.01688 287.39   <2e-16 ***
## LVirginia -0.33284   0.02409 -13.82   <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(dens) 4.568     19 72.76   <2e-16 ***
## ---      
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.946   Deviance explained = 94.9%
## -REML = -54.242   Scale est. = 0.011737 n = 84

```

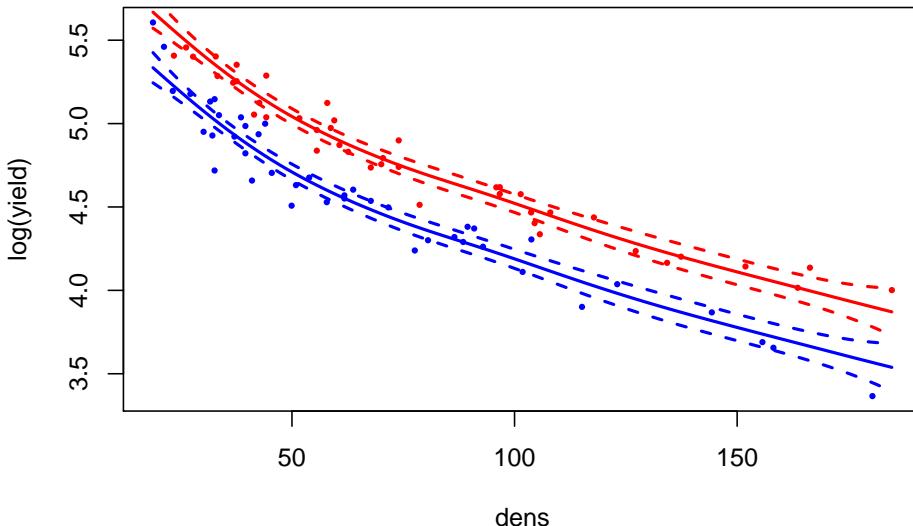
La siguiente figura muestra las curvas ajustadas para cada ubicación con el modelo fit1, asumimos que ambas curvas son paralelas.

```

L.P <- rep(levels(L)[1],100)
L.V <- rep(levels(L)[2],100)
dens.g <- seq(min(dens),max(dens),l=100)
fit1.P <- predict(fit1,newdata=data.frame(L=L.P,dens=dens.g),se.fit=TRUE)
fit1.V <- predict(fit1,newdata=data.frame(L=L.V,dens=dens.g),se.fit=TRUE)

plot(dens,log(yield),col=points.cols[location+1],pch=16,cex=.55)
lines(dens.g,fit1.P$fit,col=2,lwd=2)
lines(dens.g,fit1.P$fit+1.96*fit1.P$se.fit,col=2,lwd=2,lty=2)
lines(dens.g,fit1.P$fit-1.96*fit1.P$se.fit,col=2,lwd=2,lty=2)
lines(dens.g,fit1.V$fit,col=4,lwd=2)
lines(dens.g,fit1.V$fit+1.96*fit1.V$se.fit,col=4,lwd=2,lty=2)
lines(dens.g,fit1.V$fit-1.96*fit1.V$se.fit,col=4,lwd=2,lty=2)

```



Asumir curvas paralelas para ambas localizaciones implica que la disminución del rendimiento a medida que aumenta la densidad es la misma en ambas localizaciones. En lugar de ajustar un modelo de efectos aditivos, podemos ajustar un modelo de efectos con interacción, como por ejemplo:

$$\log(\text{yield}_i) = \beta_0 + \beta_1 \beta_1 \text{location}_{ij} + f(\text{dens}_i)_{L(j)} + \epsilon_i$$

donde

$$L(j) = \begin{cases} 0 & \text{si la } i\text{-ésima observación es de Purnong Landing} \\ 1 & \text{si la } i\text{-ésima observación es de Virginia} \end{cases}$$

El opción `by=` dentro de `s()` permite incluir interacciones curva factor:

```
fit2 <- gam(log(yield) ~ L + s(dens, k=20, m=2, bs="ps", by=L),
             method="REML", select=TRUE)
summary(fit2)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## log(yield) ~ L + s(dens, k = 20, m = 2, bs = "ps", by = L)
##
## Parametric coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.84407    0.01603 302.12  <2e-16 ***
## LVirginia   -0.33003    0.02271 -14.54  <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(dens):LPurnong Landing 3.097     18 37.62 <2e-16 ***
## s(dens):LVirginia       4.728     17 52.10 <2e-16 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) = 0.952 Deviance explained = 95.7%
## -REML = -53.616 Scale est. = 0.01045 n = 84

¿Qué modelo es mejor?

AIC(fit1)

## [1] -125.2307

AIC(fit2)

## [1] -131.1844

fit1$sp

##      s(dens)1      s(dens)2
## 164.66124239  0.00147493

fit2$sp

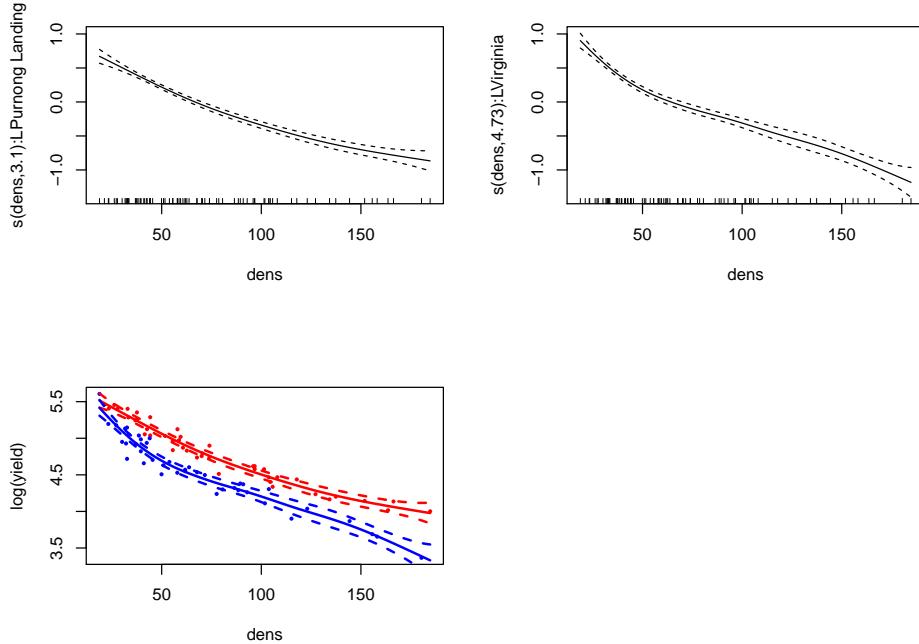
## s(dens):LPurnong Landing1 s(dens):LPurnong Landing2
##           4.264039e+02           1.604648e-03
##      s(dens):LVirginia1      s(dens):LVirginia2
##           6.437458e+01           1.053443e-03

# plot the smooth effects
par(mfrow=c(2,2))
plot(fit2, se=TRUE)

# In the same plot
fit2.P <- predict(fit2,newdata=data.frame(L=L.P,dens=dens.g),se.fit=TRUE)
fit2.V <- predict(fit2,newdata=data.frame(L=L.V,dens=dens.g),se.fit=TRUE)

plot(dens,log(yield),col=points.cols[location+1],pch=16,cex=.55)
lines(dens.g,fit2.P$fit,col=2,lwd=2)
lines(dens.g,fit2.P$fit+1.96*fit1.P$se.fit,col=2,lwd=2,lty=2)
lines(dens.g,fit2.P$fit-1.96*fit1.P$se.fit,col=2,lwd=2,lty=2)
lines(dens.g,fit2.V$fit,col=4,lwd=2)
lines(dens.g,fit2.V$fit+1.96*fit2.V$se.fit,col=4,lwd=2,lty=2)
lines(dens.g,fit2.V$fit-1.96*fit2.V$se.fit,col=4,lwd=2,lty=2)

```

Figura 8.1: *Fitted curves by location*

8.3.1. Ejemplo:

Continuemos con el ejemplo `Wage`

```
gam2 <- gam(wage ~ jobclass + s(age,k=6)+s(year,k=6)+education,data = Wage)
gam3 <- gam(wage ~ jobclass + s(age,by=jobclass,k=6)+s(year,k=6)+education,data = Wage)
summary(gam2)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## wage ~ jobclass + s(age, k = 6) + s(year, k = 6) + education
##
## Parametric coefficients:
##                                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)                      84.184     2.186  38.511 < 2e-16 ***
## jobclass2. Information          4.317     1.352   3.193  0.00142 **
## education2. HS Grad            10.751     2.426   4.432 9.67e-06 ***
## education3. Some College       22.741     2.567   8.861 < 2e-16 ***
```

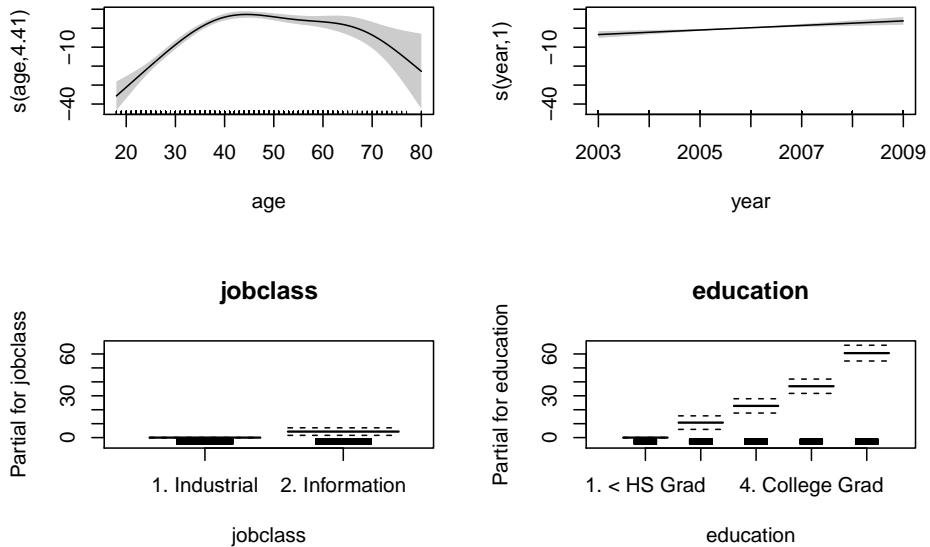
```

## education4. College Grad      36.847      2.572 14.325 < 2e-16 ***
## education5. Advanced Degree  60.594      2.824 21.458 < 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(age)   4.408 4.825 45.57 < 2e-16 ***
## s(year)  1.000 1.000 14.29 0.00016 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) = 0.292 Deviance explained = 29.5%
## GCV = 1237.3 Scale est. = 1232.6 n = 3000
summary(gam3)

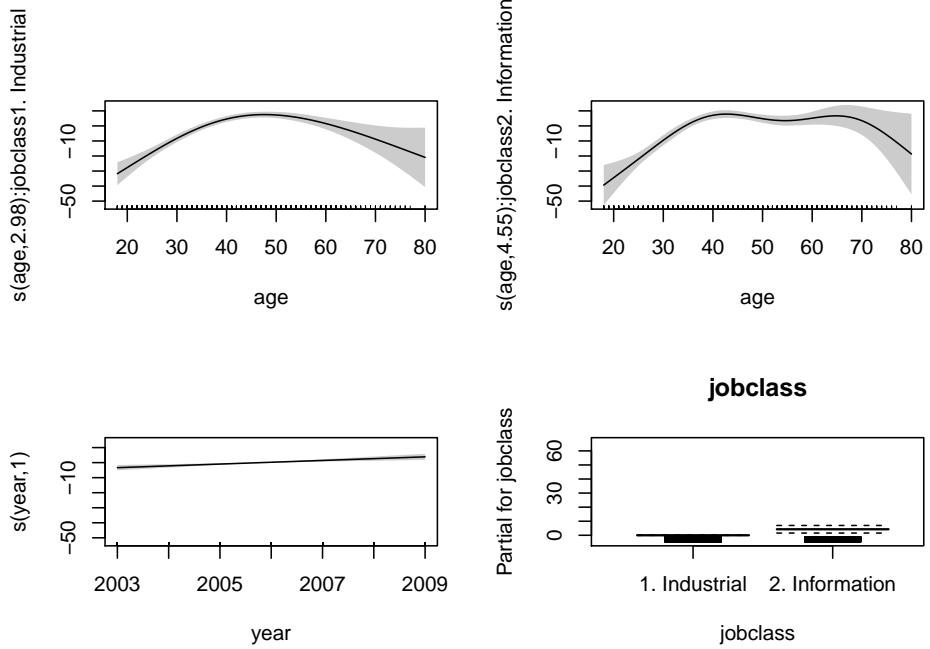
## 
## Family: gaussian
## Link function: identity
##
## Formula:
## wage ~ jobclass + s(age, by = jobclass, k = 6) + s(year, k = 6) +
##       education
##
## Parametric coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  84.039     2.186 38.453 < 2e-16 ***
## jobclass2. Information      4.284     1.352  3.169  0.00154 **
## education2. HS Grad         10.733     2.425  4.427  9.9e-06 ***
## education3. Some College    22.796     2.564  8.892 < 2e-16 ***
## education4. College Grad    36.940     2.571 14.367 < 2e-16 ***
## education5. Advanced Degree 60.621     2.822 21.482 < 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(age):jobclass1. Industrial 2.980 3.607 33.95 < 2e-16 ***
## s(age):jobclass2. Information 4.551 4.901 20.27 < 2e-16 ***
## s(year)                   1.000 1.000 14.27 0.000162 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) = 0.293 Deviance explained = 29.6%
## GCV = 1237.2 Scale est. = 1231.3 n = 3000

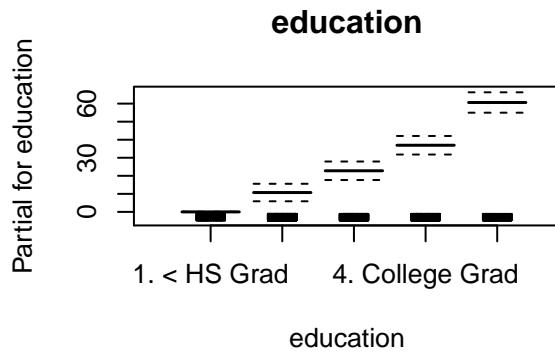
```

```
par(mfrow=c(2,2))
plot(gam2,all.terms=TRUE,scheme=1)
```



```
plot(gam3,all.terms=TRUE,scheme=1)
```



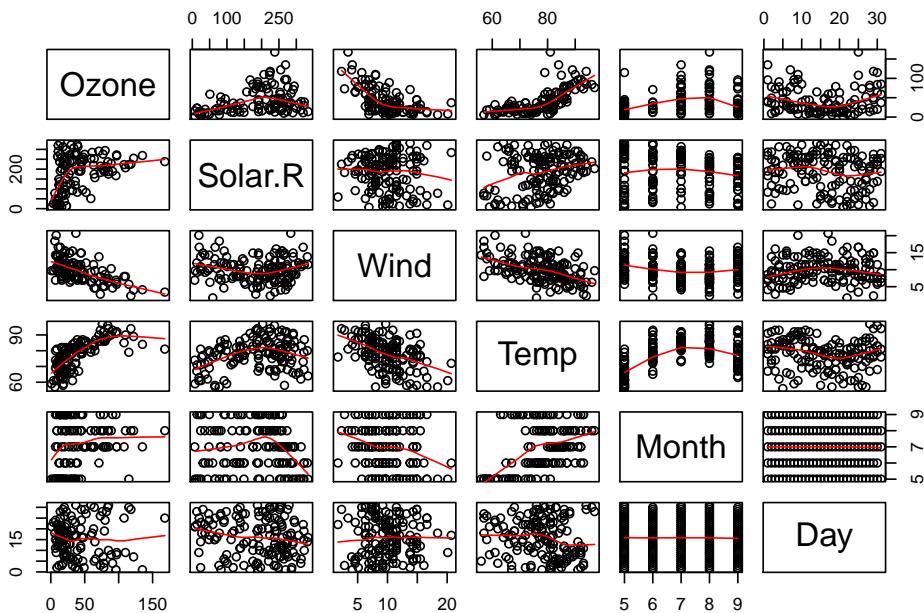


Ejercicio: Selecciona un modelo GAM.

8.4. Ejemplo: Calidad del Aire

Revisemos los datos ?airquality

```
data(airquality)
pairs(airquality, panel = panel.smooth)
```



Una simple gráfica de dispersión muestra que algunas de las variables tienen una relación no lineal.

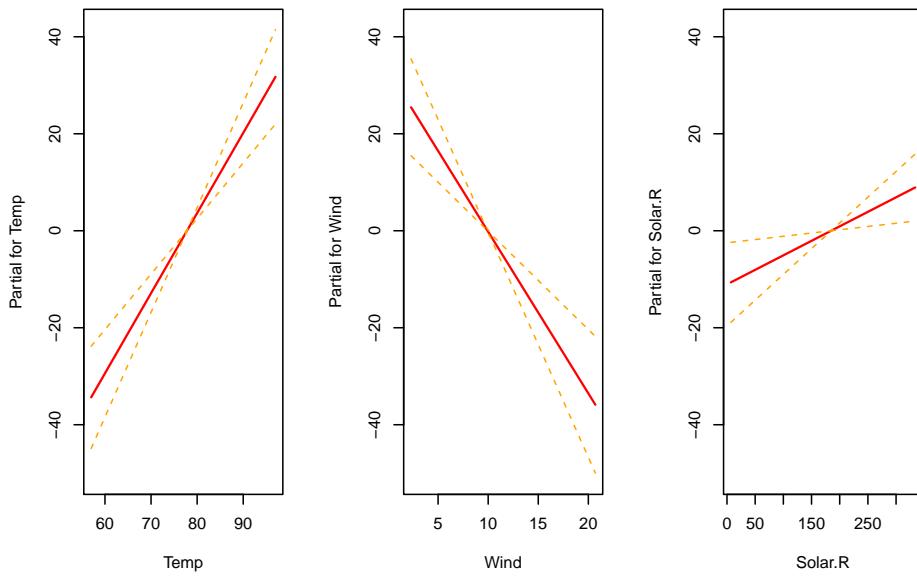
Consideraremos un modelo lineal para la variable de respuesta Ozone

```
airq.lm <- lm(Ozone ~ Temp + Wind + Solar.R, data=airquality)
summary(airq.lm)
```

```
##
## Call:
## lm(formula = Ozone ~ Temp + Wind + Solar.R, data = airquality)
##
## Residuals:
##    Min     1Q   Median     3Q    Max
## -40.485 -14.219  -3.551  10.097 95.619
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -64.34208  23.05472 -2.791 0.00623 **
## Temp         1.65209   0.25353  6.516 2.42e-09 ***
## Wind        -3.33359   0.65441 -5.094 1.52e-06 ***
## Solar.R      0.05982   0.02319  2.580  0.01124 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 21.18 on 107 degrees of freedom
## (42 observations deleted due to missingness)
## Multiple R-squared:  0.6059, Adjusted R-squared:  0.5948
## F-statistic: 54.83 on 3 and 107 DF,  p-value: < 2.2e-16
```

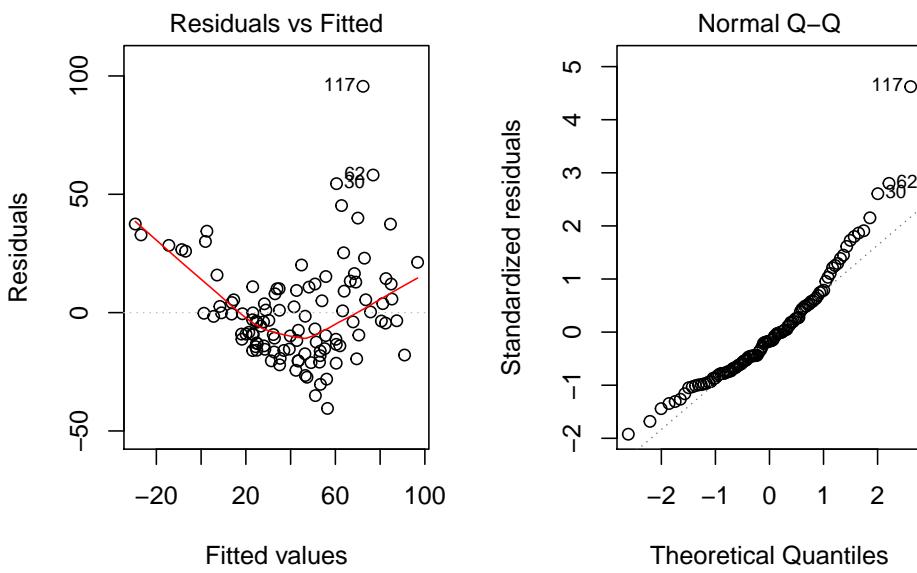
Veamos los resultados gráficamente:

```
par(mfrow=c(1,3))
termplot(airq.lm, se=TRUE)
```



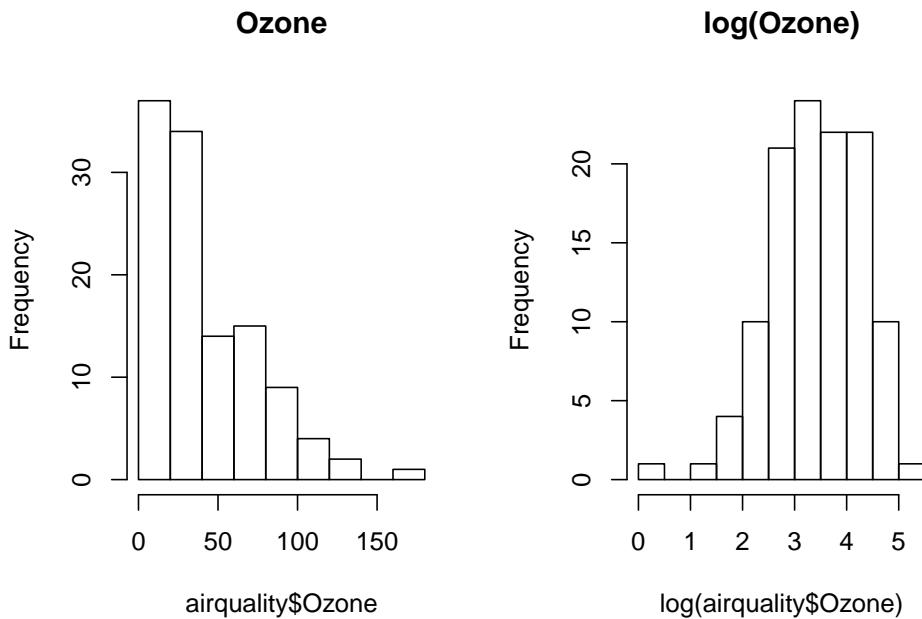
También podemos graficar los residuos del modelo para comprobar las hipótesis del modelo.

```
par(mfrow=c(1,2))
plot(airq.lm,which=1:2)
```



La falta de normalidad en los residuos se debe a la asimetría de la variable de respuesta Ozone, podemos aplicar una transformación log para conseguir asimetría, es decir:

```
par(mfrow=c(1,2))
hist(airquality$Ozone, main="Ozone")
hist(log(airquality$Ozone), main ="log(Ozone)")
```



Ahora podemos ajustar un modelo lineal del $\log(Ozono)$, ¿el modelo se ve más adecuado?

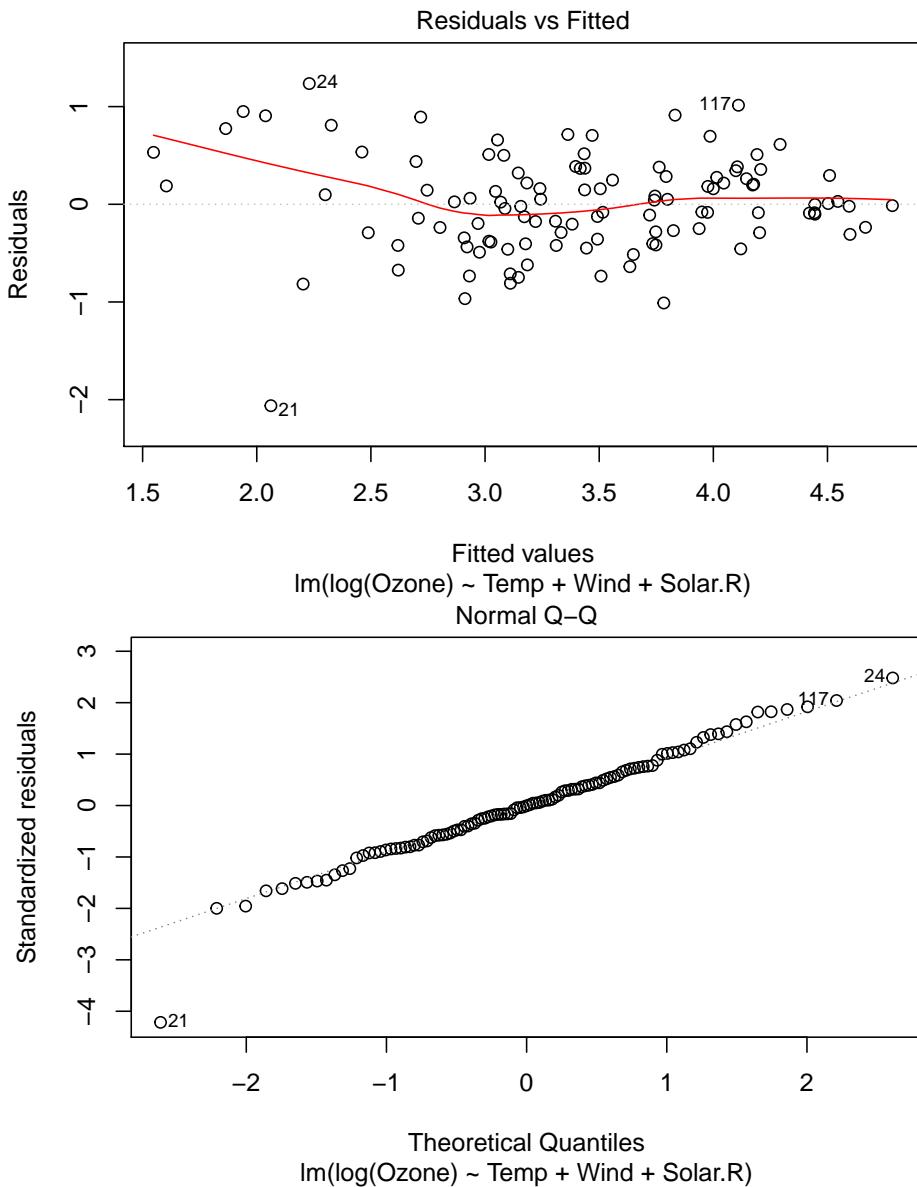
```
lairq.lm <- lm(log(Ozone) ~ Temp + Wind + Solar.R, data=airquality)
summary(lairq.lm)
```

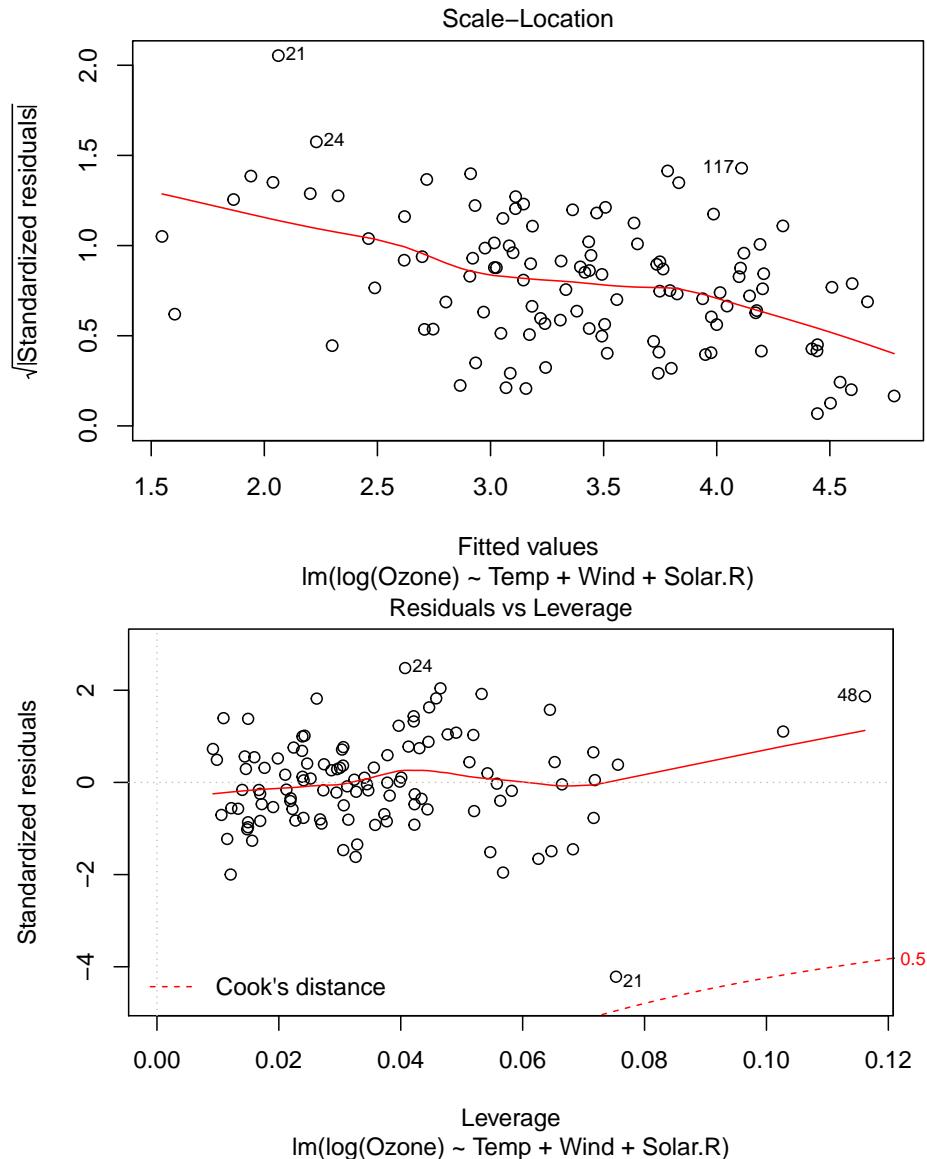
```
##
## Call:
## lm(formula = log(Ozone) ~ Temp + Wind + Solar.R, data = airquality)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.06193 -0.29970 -0.00231  0.30756  1.23578
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.2621323  0.5535669 -0.474 0.636798
## Temp        0.0491711  0.0060875  8.077 1.07e-12 ***
## Wind       -0.0615625  0.0157130 -3.918 0.000158 ***
## Solar.R     0.0025152  0.0005567  4.518 1.62e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```

## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5086 on 107 degrees of freedom
## (42 observations deleted due to missingness)
## Multiple R-squared: 0.6644, Adjusted R-squared: 0.655
## F-statistic: 70.62 on 3 and 107 DF, p-value: < 2.2e-16
plot(lairq.lm)

```





Ajustando un modelo lineal podemos concluir que puede haber alguna heterocedasticidad que no se tenga en cuenta en el modelo. Sin embargo, la causa más posible es que la relación entre la variable de respuesta y las covariables está lejos de ser lineal.

Ajustemos un modelo GAM, en primer lugar con una sola variable, es decir:

```
library(mgcv)
airq.gam1 <- gam(log(Ozone) ~ s(Wind, bs="ps", m=2, k=10),
```

```

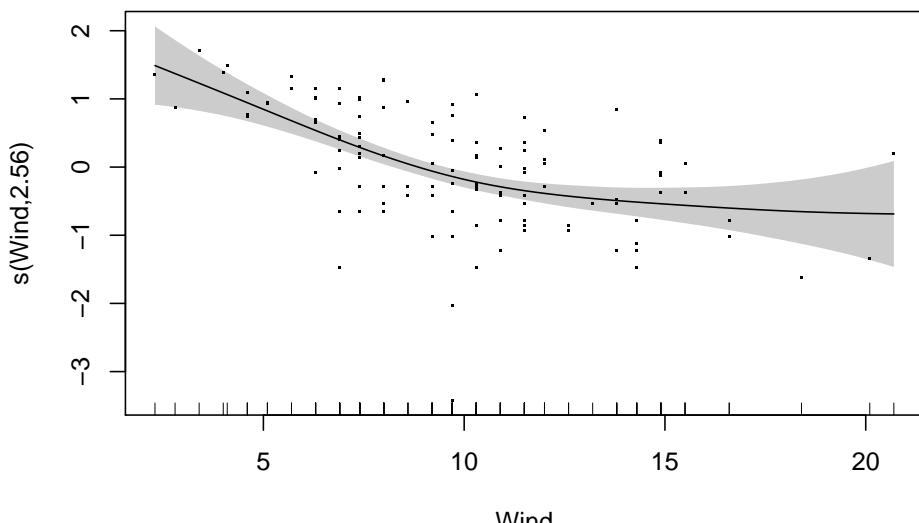
method="REML", select=TRUE,data=airquality)
summary(airq.gam1)

##
## Family: gaussian
## Link function: identity
##
## Formula:
## log(Ozone) ~ s(Wind, bs = "ps", m = 2, k = 10)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.4185    0.0655   52.19 <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(Wind) 2.565     9 6.453 2.76e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.336  Deviance explained =  35%
## -REML = 128.69  Scale est. = 0.49769  n = 116

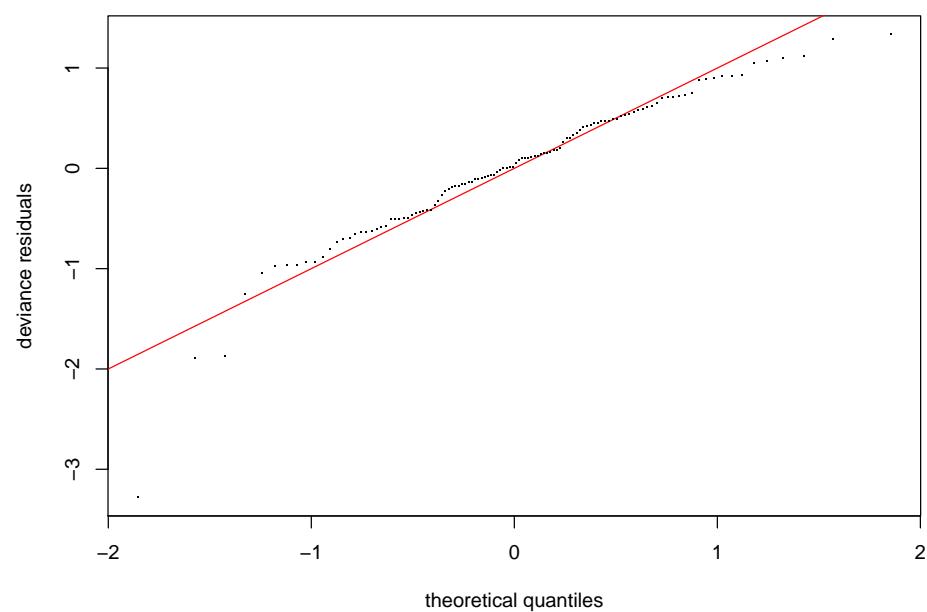
```

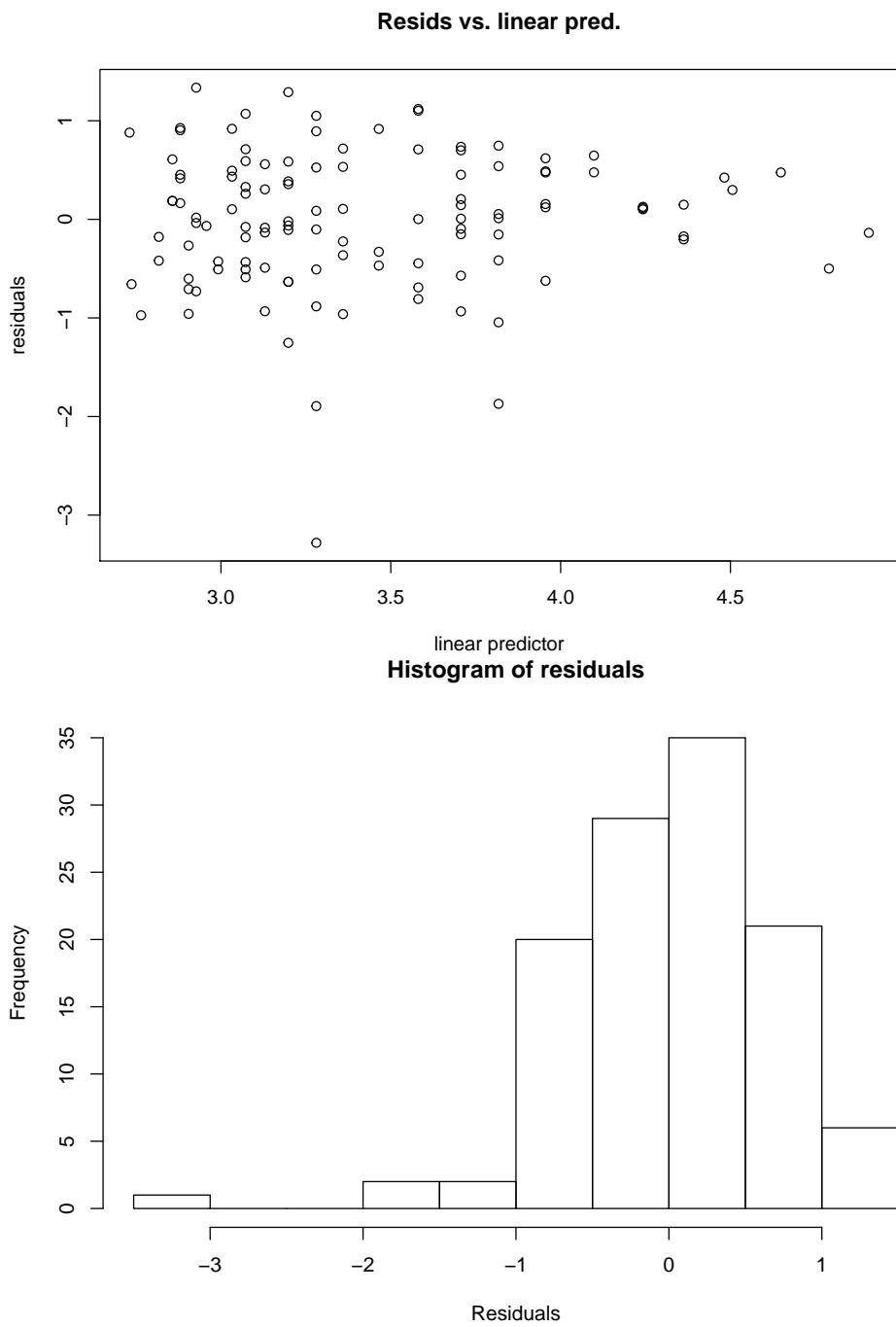
Los resultados muestran que el efecto suave `s(Wind)` es significativo (con un valor aproximado de p cercano a cero y `edf` 2,56). La siguiente figura muestra el efecto de viento suave. El incremento de la velocidad del viento disminuye los niveles de ozono (dramáticamente hasta 10mph). Tener en cuenta que incluyendo el intercepto, los efectos suaves se centran en cero.

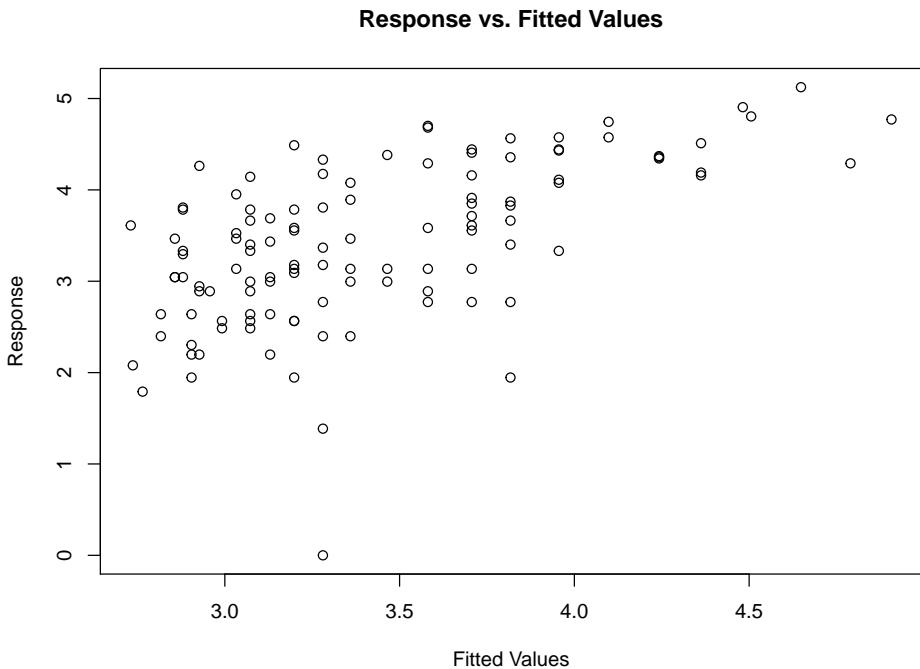
```
plot(airq.gam1,residuals=TRUE,scheme=1)
```



```
gam.check(airq.gam1)
```







```
##  
## Method: REML Optimizer: outer newton  
## full convergence after 8 iterations.  
## Gradient range [-1.455961e-06,1.183099e-06]  
## (score 128.6926 & scale 0.4976891).  
## Hessian positive definite, eigenvalue range [0.4379927,57.51548].  
## Model rank = 10 / 10  
##  
## 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(Wind) 9.00  2.56    0.69  <2e-16 ***  
## ---  
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Nótese que, en el modelo `airq.gam1` usamos $k=10$ nodos, la variable `Wind` tiene valores únicos de 31, y por lo tanto 10 nodos deberían ser suficientes. Ajustamos un modelo para los residuos del modelo `gam` ajustado

```
resids <- residuals(airq.gam1)  
resids.gam <- gam(resids~s(Wind,k=20,m=2),method="REML",  
                   select=TRUE,data=airq.gam1$model)  
summary(resids.gam)
```

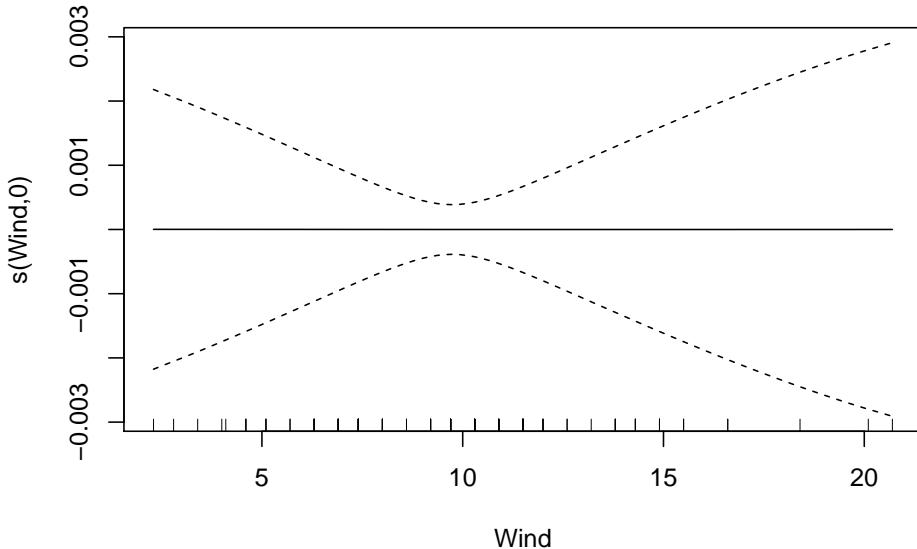
```

## 
## Family: gaussian
## Link function: identity
##
## Formula:
## resids ~ s(Wind, k = 20, m = 2)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.137e-15 6.477e-02     0      1
##
## Approximate significance of smooth terms:
##             edf Ref.df F p-value
## s(Wind)    7.419e-05    19 0      1
##
## R-sq.(adj) = -5.66e-07  Deviance explained = 7.89e-06%
## -REML = 124.14  Scale est. = 0.48659 n = 116

```

Los resultados muestran que no hay una variabilidad no-explicada entre la variable y los residuos. Por lo tanto, podemos concluir que no necesitamos más nodos. La siguiente figura apoya esta afirmación.

```
plot(resids.gam)
```



Ahora añadimos la función Temp:

```

airq.gam2 <- gam(log(Ozone) ~ s(Wind, bs="ps", m=2, k=10) + s(Temp, bs="ps", m=2, k=10),
                    method="REML", select=TRUE, data=airquality)
summary(airq.gam2)

```

```

## 
## Family: gaussian
## Link function: identity
##
## Formula:
## log(Ozone) ~ s(Wind, bs = "ps", m = 2, k = 10) + s(Temp, bs = "ps",
##           m = 2, k = 10)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.41852   0.04963   68.89 <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(Wind) 2.068      9 1.353 0.000981 ***
## s(Temp) 3.990      9 10.496 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.619  Deviance explained = 63.9%
## -REML = 101.64  Scale est. = 0.28568 n = 116

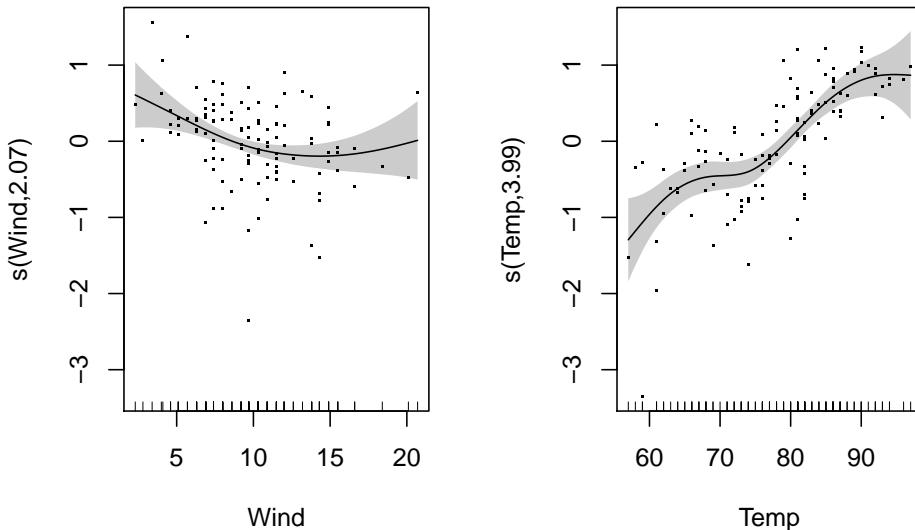
```

Por tanto el efecto de Temp es significativo.

```

par(mfrow=c(1,2))
plot(airq.gam2,residuals=TRUE,scheme=1)

```



Podemos comparar ambos modelos utilizando la función `anova`:

```
anova(airq.gam1,airq.gam2)
```

```
## Analysis of Deviance Table
##
## Model 1: log(Ozone) ~ s(Wind, bs = "ps", m = 2, k = 10)
## Model 2: log(Ozone) ~ s(Wind, bs = "ps", m = 2, k = 10) + s(Temp, bs = "ps",
##                         m = 2, k = 10)
##   Resid. Df Resid. Dev      Df Deviance
## 1     111.16      55.958
## 2     105.98      31.123 5.1832    24.835
```

Podemos también comparar el criterio AIC:

```
AIC(airq.gam1)
```

```
## [1] 255.0315
```

```
AIC(airq.gam2)
```

```
## [1] 195.6561
```

Incluimos la variable `Solar.R` que contiene valores faltantes (NA's).

```
sum(is.na(airquality$Solar.R))
```

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

Para comparar el modelo anterior `airq.gam2` y el nuevo modelo que incluye `Solar.R` utilizaremos los mismos datos, por lo que omitiremos los valores que faltan y volveremos a ajustar los modelos.

```

new.airquality <- na.omit(airquality)

airq.gam22=gam(log(Ozone)~s(Wind,bs="ps",m=2,k=10)+s(Temp,bs="ps",m=2,k=10),
               method="REML",select=TRUE,data=new.airquality)

airq.gam3=gam(log(Ozone)~s(Wind,bs="ps",m=2,k=10)+s(Temp,bs="ps",m=2,k=10)+s(Solar.R,bs="ps",m=2,k=20),
               method="REML",select=TRUE,data=new.airquality)

summary(airq.gam22)

##
## Family: gaussian
## Link function: identity
##
## Formula:
## log(Ozone) ~ s(Wind, bs = "ps", m = 2, k = 10) + s(Temp, bs = "ps",
##             m = 2, k = 10)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.41593   0.05128  66.61  <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(Wind) 2.1879      9 1.919  1e-04 ***
## s(Temp) 0.9874      9 8.601 7.73e-16 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) = 0.611 Deviance explained = 62.2%
## -REML = 95.215 Scale est. = 0.29194 n = 111
summary(airq.gam3)

##
## Family: gaussian
## Link function: identity
##
## Formula:
## log(Ozone) ~ s(Wind, bs = "ps", m = 2, k = 10) + s(Temp, bs = "ps",
##             m = 2, k = 10) + s(Solar.R, bs = "ps", m = 2, k = 20)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
```

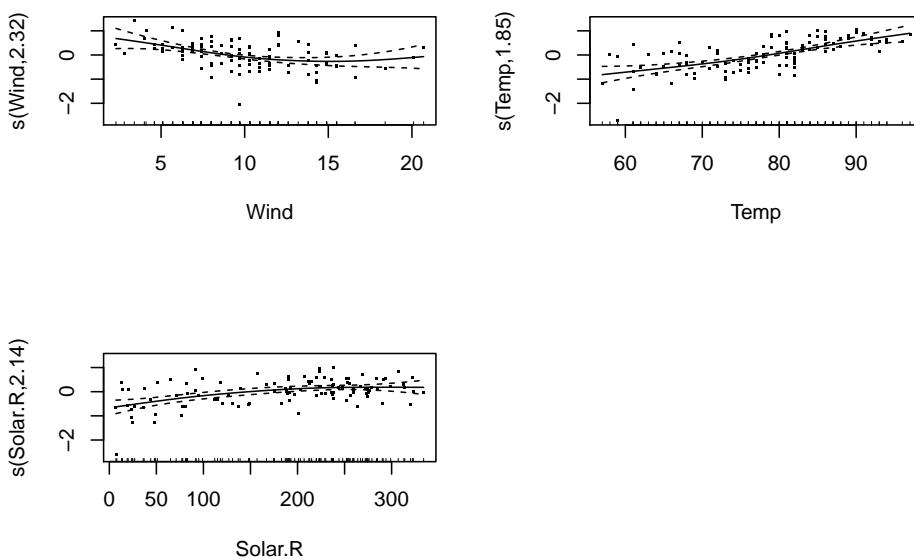
```

## (Intercept) 3.41593    0.04586   74.49   <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(Wind)    2.318     9 2.255 2.44e-05 ***
## s(Temp)    1.852     9 6.128 1.12e-12 ***
## s(Solar.R) 2.145    19 1.397 1.23e-06 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) = 0.689 Deviance explained = 70.7%
## -REML = 86.106 Scale est. = 0.23342 n = 111
AIC(airq.gam22)

## [1] 185.7769
AIC(airq.gam3)

## [1] 166.0423
par(mfrow=c(2,2))
plot(airq.gam3,residuals=TRUE)

```



Capítulo 9

Análisis multivariante

9.1. Análisis de Componentes Principales

El análisis de componentes principales (*principal component analysis*) o PCA es una de las técnicas de aprendizaje no supervisado, las cuales suelen aplicarse como parte del análisis exploratorio de los datos.

A diferencia de los métodos de aprendizaje supervisado, donde contamos con un grupo de variables o características ($X = X_1, X_2, \dots, X_p$) medidas sobre un conjunto de observaciones n , con la intención de obtener predicciones sobre una variable respuesta Y asociada, en los no supervisados solo contamos con un número de variables de las cuales nos interesa conocer o de las que queremos extraer información, por ejemplo, sobre la existencia de subgrupos entre las variables u observaciones.

Una de las aplicaciones de PCA es la **reducción de dimensionalidad** (variables), perdiendo la menor cantidad de información (varianza) posible: cuando contamos con un gran número de variables cuantitativas posiblemente correlacionadas (indicativo de existencia de información redundante), PCA permite reducirlas a un número menor de variables transformadas (componentes principales) que expliquen gran parte de la variabilidad en los datos. Cada dimensión o componente principal generada por PCA será una combinación lineal de las variables originales, y serán además independientes o no correlacionadas entre sí.

Se utiliza para enfatizar la variación y sacar a relucir patrones fuertes en un conjunto de datos. A menudo se utiliza para hacer que los datos sean fáciles de explorar y visualizar. Vamos a realizar un PCA de los resultados obtenidos en la competición de heptatlón femenino de los Juegos Olímpicos de Seúl (1988).

```
library(HSAUR2)
data("heptathlon")
```

```
head(heptathlon)
```

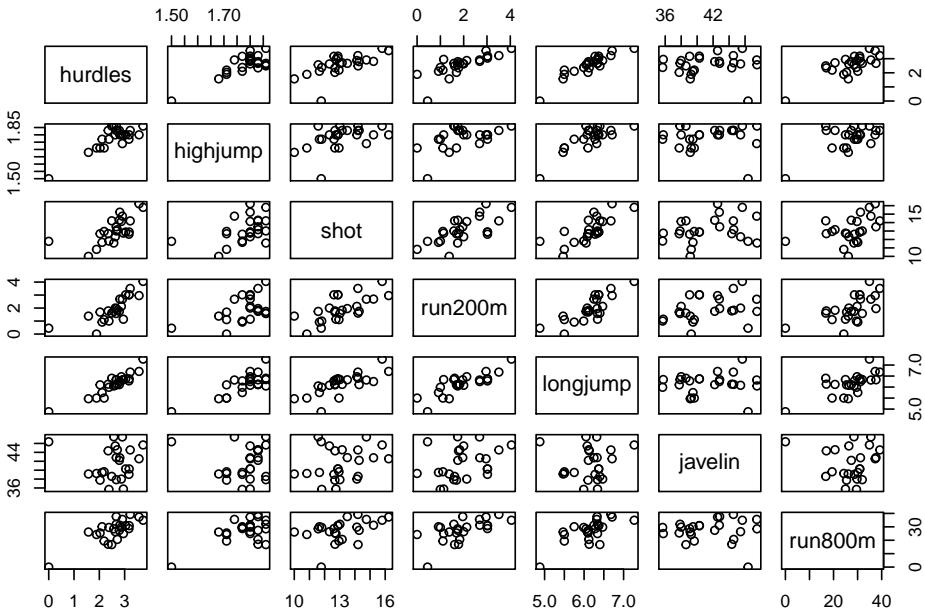
```
##                                     hurdles highjump shot run200m longjump javelin
## Joyner-Kersee (USA)      12.69    1.86 15.80   22.56    7.27  45.66
## John (GDR)              12.85    1.80 16.23   23.65    6.71  42.56
## Behmer (GDR)            13.20    1.83 14.20   23.10    6.68  44.54
## Sablovskaite (URS)     13.61    1.80 15.23   23.92    6.25  42.78
## Choubenkova (URS)      13.51    1.74 14.76   23.93    6.32  47.46
## Schulz (GDR)           13.75    1.83 13.50   24.65    6.33  42.82
##                               run800m score
## Joyner-Kersee (USA)    128.51  7291
## John (GDR)              126.12  6897
## Behmer (GDR)            124.20  6858
## Sablovskaite (URS)     132.24  6540
## Choubenkova (URS)      127.90  6540
## Schulz (GDR)           125.79  6411
```

Recodificamos las pruebas relativas a 3 carreras `hurdles`, `run200m` y `run800m`, restando el valor más alto en cada carrera, cada uno de los 35 tiempos de los atletas.

```
heptathlon$hurdles <- max(heptathlon$hurdles) - heptathlon$hurdles
heptathlon$run200m <- max(heptathlon$run200m) - heptathlon$run200m
heptathlon$run800m <- max(heptathlon$run800m) - heptathlon$run800m
```

Diagrama de dispersion

```
score <- which(colnames(heptathlon) == "score")
plot(heptathlon[, -score])
```



Matriz de correlación

```
round(cor(heptathlon[,-score]),3)
```

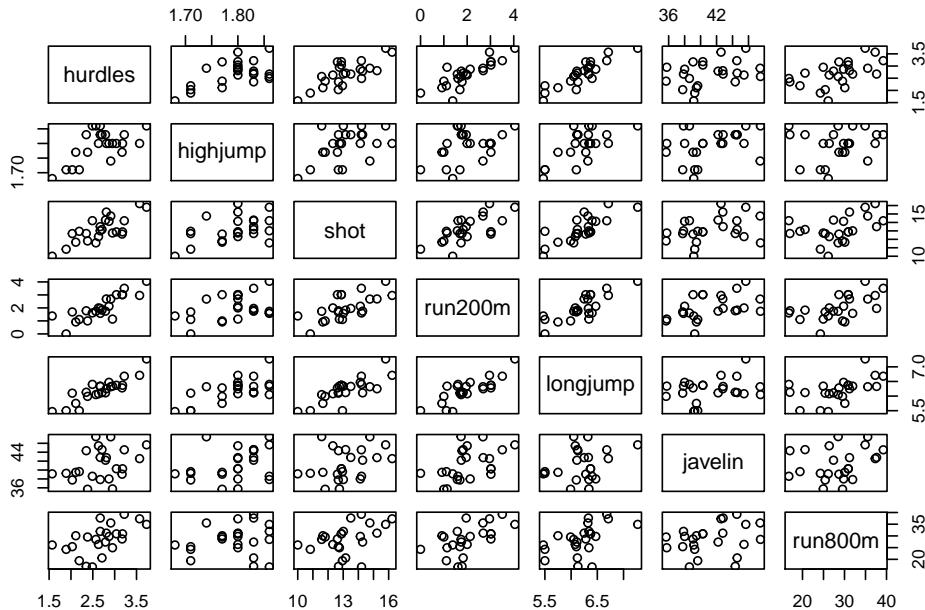
```
##          hurdles highjump   shot run200m longjump javelin run800m
## hurdles    1.000   0.811  0.651   0.774   0.912   0.008   0.779
## highjump   0.811   1.000  0.441   0.488   0.782   0.002   0.591
## shot       0.651   0.441  1.000   0.683   0.743   0.269   0.420
## run200m   0.774   0.488  0.683   1.000   0.817   0.333   0.617
## longjump   0.912   0.782  0.743   0.817   1.000   0.067   0.700
## javelin     0.008   0.002  0.269   0.333   0.067   1.000  -0.020
## run800m   0.779   0.591  0.420   0.617   0.700  -0.020   1.000
```

La matriz de resultados confirma que la gran mayoría de las correlaciones entre las pruebas son positivas, con una alta correlación entre el salto de longitud y el salto de altura (**longjump**) y los 100m vallas (**hurdles**). A algunos les gusta menos el salto de altura (**highjump**) y el disparo (**shot**) y la jabalina (**javelin**) que tiene una correlación cercana a cero con el resto de las pruebas.

Una posible explicación para este resultado podría ser que el entrenamiento para las otras 6 pruebas no añade mucho a la prueba de jabalina, que es una prueba más técnica.

Puede observarse que existe un valor atípico en casi todas las pruebas que corresponde a un atleta Launa (PNG) de Papúa Nueva Guinea - eliminaremos esta observación para ver si la matriz de correlación es significativamente diferente:

```
heptathlon <- heptathlon[-which(rownames(heptathlon)=="Launa (PNG)",)]
plot(heptathlon[, -score])
```



Eliminando al atleta de Papúa Nueva Guinea, las correlaciones cambian sustancialmente y en el diagrama de dispersión matricial no se observan valores extremos.

```
round(cor(heptathlon[,-score]), 3)
```

```
##          hurdles highjump   shot run200m longjump javelin run800m
## hurdles     1.000  0.582  0.767  0.830  0.889  0.332  0.559
## highjump    0.582   1.000  0.465  0.391  0.663  0.348  0.152
## shot        0.767   0.465  1.000  0.669  0.784  0.343  0.408
## run200m     0.830   0.391  0.669   1.000  0.811  0.471  0.573
## longjump    0.889   0.663  0.784   0.811   1.000  0.287  0.523
## javelin      0.332   0.348  0.343   0.471   0.287  1.000  0.256
## run800m     0.559   0.152  0.408   0.573   0.523  0.256  1.000
```

Para realizar el PCA, partiremos de la matriz de correlación, ya que las 7 pruebas se miden en diferentes escalas (metros, segundos). Este procedimiento se denomina PCA normalizado (`scale=TRUE` en la función `prcomp`).

```
?prcomp
heptathlon_pca <- prcomp(heptathlon[,-score], scale=TRUE)
head(heptathlon_pca, 5)
```

```
## $sdev
## [1] 2.0793370 0.9481532 0.9109016 0.6831967 0.5461888 0.3374549 0.2620420
## 
## $rotation
```

```

##          PC1        PC2        PC3        PC4        PC5
## hurdles -0.4503876  0.05772161 -0.1739345  0.04840598 -0.19889364
## highjump -0.3145115 -0.65133162 -0.2088272 -0.55694554  0.07076358
## shot     -0.4024884 -0.02202088 -0.1534709  0.54826705  0.67166466
## run200m  -0.4270860  0.18502783  0.1301287  0.23095946 -0.61781764
## longjump -0.4509639 -0.02492486 -0.2697589 -0.01468275 -0.12151793
## javelin   -0.2423079 -0.32572229  0.8806995  0.06024757  0.07874396
## run800m  -0.3029068  0.65650503  0.1930020 -0.57418128  0.31880178
##          PC6        PC7
## hurdles  0.84665086 -0.06961672
## highjump -0.09007544  0.33155910
## shot     -0.09886359  0.22904298
## run200m  -0.33279359  0.46971934
## longjump -0.38294411 -0.74940781
## javelin   0.07193437 -0.21108138
## run800m  -0.05217664  0.07718616
##
## $center
##   hurdles  highjump      shot    run200m  longjump   javelin   run800m
## 2.687500 1.793750 13.173333 2.023750 6.205417 41.278333 28.516667
##
## $scale
##   hurdles  highjump      shot    run200m  longjump   javelin
## 0.51456398 0.05232112 1.49714995 0.93676972 0.40165938 3.46870690
##   run800m
## 6.14724800
##
## $x
##          PC1        PC2        PC3        PC4
## Joyner-Kersee (USA) -4.757530189 -0.13986143 -0.006040526  0.293416339
## John (GDR)         -3.147943402  0.94859029 -0.243919842  0.549171385
## Behmer (GDR)        -2.926184760  0.69534239  0.622293440 -0.554744912
## Sablovskaite (URS) -1.288135516  0.17900713  0.250632380  0.637174187
## Choubenkova (URS)  -1.503450994  0.96177329  1.780588549  0.784035325
## Schulz (GDR)       -0.958467101  0.35121643  0.413086366 -1.113546938
## Fleming (AUS)      -0.953445060  0.49982537 -0.265135015 -0.140202490
## Greiner (USA)       -0.633239267  0.37592917 -1.140338594  0.142558348
## Lajbnerova (CZE)   -0.381571974 -0.71213459 -0.068395353  0.087212735
## Bouraga (URS)      -0.522322004  0.77688861 -0.481071429  0.283745698
## Wijnsma (HOL)      -0.217701500 -0.23369645 -1.154221444 -1.260128609
## Dimitrova (BUL)   -1.075984276  0.51552998 -0.312458252 -0.127032432
## Scheider (SWI)     0.003014986 -1.44688825  1.582739069 -1.254415325
## Braun (FRG)        0.109183759 -1.63595645  0.469577294  0.362580442
## Ruotsalainen (FIN) 0.208868056 -0.68866173  1.152140223 -0.112914470
## Yuping (CHN)        0.232507119 -1.95999641 -1.541230813  0.598325122
## Hagger (GB)         0.659520046 -0.08775813 -1.796509771 -0.182375000

```

```

## Brown (USA)          0.756854602 -2.04292201  0.451506018  0.476926314
## Mulliner (GB)       1.880932819  0.91530324 -0.359311801  0.799619094
## Hautenauve (BEL)    1.828170404  0.72629699 -1.048640439 -0.711793161
## Kytola (FIN)         2.118203163  0.39921397  0.190158154 -0.788445056
## Geremias (BRA)        2.770706272  0.03463584  0.170274969  1.385562494
## Hui-Ing (TAI)         3.901166920  1.20175472  0.943677497 -0.002429122
## Jeong-Mi (KOR)        3.896847898  0.36656804  0.390599321 -0.152299968
##                               PC5      PC6      PC7
## Joyner-Kersee (USA)   -0.36183307 -0.27050283 -0.47587527
## John (GDR)             0.75364464  0.37770017 -0.05172711
## Behmer (GDR)            -0.19035037 -0.25780287  0.11054960
## Sablovskaite (URS)    0.60362153 -0.21575716  0.53075152
## Choubenkova (URS)     0.58969949  0.08014332 -0.30081842
## Schulz (GDR)           0.71483887 -0.25436956  0.03838796
## Fleming (AUS)          -0.86581530  0.03691813  0.23005943
## Greiner (USA)            0.20807431 -0.14236240 -0.06374657
## Lajbnerova (CZE)        0.67727618  0.25014881  0.35555639
## Bouraga (URS)            -1.18784299  0.39881271  0.19712215
## Wijnsma (HOL)            0.37497195 -0.20267731  0.17459647
## Dimitrova (BUL)         -0.91992929  0.26727067  0.21111846
## Scheider (SWI)           -0.20526249  0.17597425 -0.03915701
## Braun (FRG)              -0.14712208  0.26134199 -0.01334416
## Ruotsalainen (FIN)      -0.31539746  0.18351622 -0.14127555
## Yuping (CHN)                0.17451428 -0.50175724  0.04999374
## Hagger (GB)                 -0.05104049  0.55058471 -0.46388534
## Brown (USA)                  -0.38154294 -0.26606429 -0.11099445
## Mulliner (GB)                 -0.06942955 -0.73259727 -0.31281502
## Hautenauve (BEL)               0.14092347  0.06933542 -0.07548638
## Kytola (FIN)                   0.41815113 -0.03363651  0.12143219
## Geremias (BRA)                  0.28541366  0.38083979  0.34574480
## Hui-Ing (TAI)                    -0.67080776 -0.52756760  0.09436975
## Jeong-Mi (KOR)                   0.42524426  0.37250885 -0.41055719

a1 <- heptathlon_pca$rotation[,1]

summary
summary(heptathlon_pca)

## Importance of components:
##                               PC1      PC2      PC3      PC4      PC5      PC6
## Standard deviation      2.0793  0.9482  0.9109  0.68320  0.54619  0.33745
## Proportion of Variance 0.6177  0.1284  0.1185  0.06668  0.04262  0.01627
## Cumulative Proportion  0.6177  0.7461  0.8646  0.93131  0.97392  0.99019
##                               PC7
## Standard deviation      0.26204
## Proportion of Variance 0.00981

```

```
## Cumulative Proportion 1.00000
```

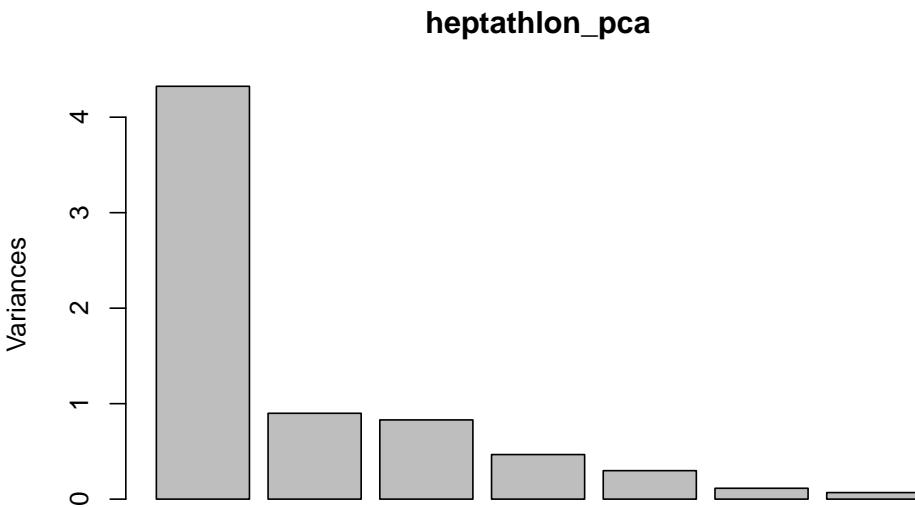
Comprobamos la variabilidad explicada:

```
head(heptathlon_pca$x)
```

		PC1	PC2	PC3	PC4
## Joyner-Kersee (USA)	-4.7575302	-0.1398614	-0.006040526	0.2934163	
## John (GDR)	-3.1479434	0.9485903	-0.243919842	0.5491714	
## Behmer (GDR)	-2.9261848	0.6953424	0.622293440	-0.5547449	
## Sablovskaite (URS)	-1.2881355	0.1790071	0.250632380	0.6371742	
## Choubenkova (URS)	-1.5034510	0.9617733	1.780588549	0.7840353	
## Schulz (GDR)	-0.9584671	0.3512164	0.413086366	-1.1135469	
		PC5	PC6	PC7	
## Joyner-Kersee (USA)	-0.3618331	-0.27050283	-0.47587527		
## John (GDR)	0.7536446	0.37770017	-0.05172711		
## Behmer (GDR)	-0.1903504	-0.25780287	0.11054960		
## Sablovskaite (URS)	0.6036215	-0.21575716	0.53075152		
## Choubenkova (URS)	0.5896995	0.08014332	-0.30081842		
## Schulz (GDR)	0.7148389	-0.25436956	0.03838796		

Gráficamente, se observa que el primer componente importante es el más dominante.

```
plot(heptathlon_pca)
```



El **biplot** es una representación gráfica de datos multivariantes. Así como una gráfica de dispersión muestra la distribución combinada de dos variables, una **biplot** representa tres o más variables.

Si ordenamos de mayor a menor la variable “puntuación” tenemos las tres me-

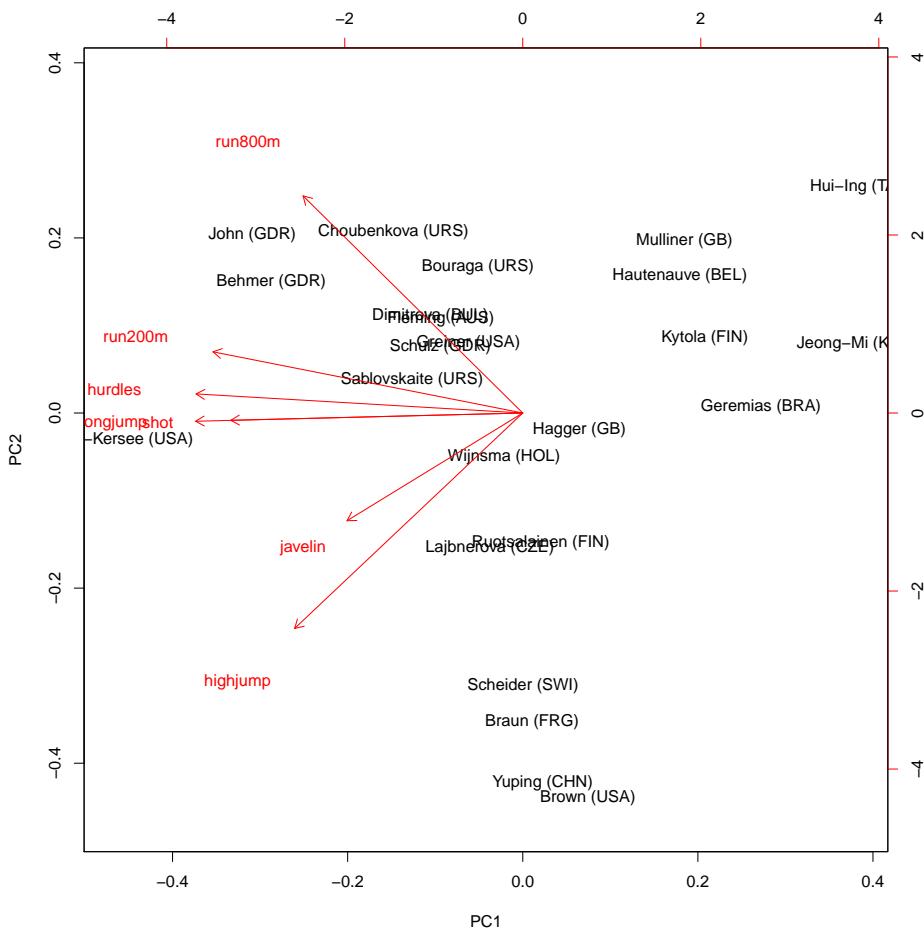
dallas de oro, plata y bronce.

```
head(heptathlon[order(heptathlon$score,decreasing = TRUE),],3)
```

```
##                               hurdles highjump   shot run200m longjump javelin
## Joyner-Kersee (USA)      3.73    1.86 15.80     4.05     7.27    45.66
## John (GDR)                3.57    1.80 16.23     2.96     6.71    42.56
## Behmer (GDR)              3.22    1.83 14.20     3.51     6.68    44.54
##                               run800m score
## Joyner-Kersee (USA)    34.92    7291
## John (GDR)                  37.31   6897
## Behmer (GDR)                39.23   6858
```

El biplot, nos muestra los atletas proyectados en sus dos primeros componentes principales, pero también las flechas nos dan información sobre las varianzas y covarianzas de las variables (direcciones de máxima variabilidad).

```
biplot(heptathlon_pca)
```

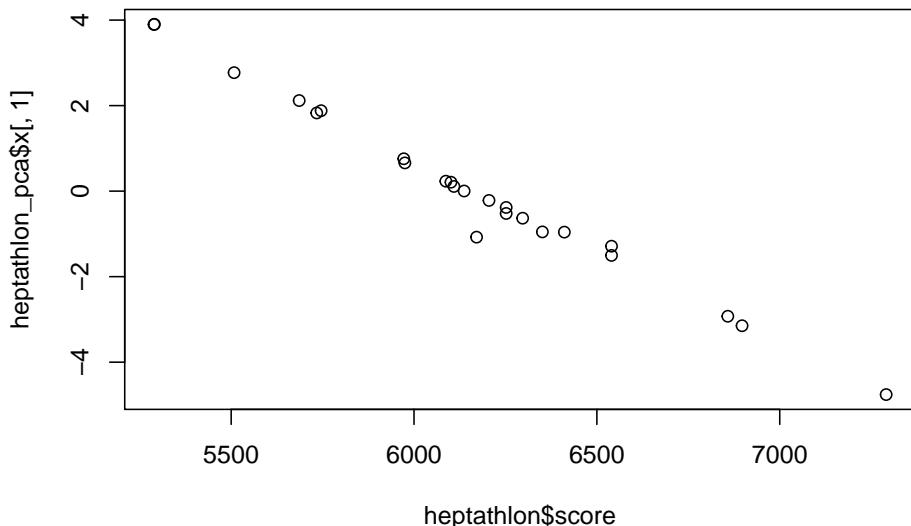


Por ejemplo, el ganador Joyner-Kersee (USA) acumula puntuaciones más altas en el longjump, hurdles y run200m.

Podemos analizar la correlación entre la puntuación de la variable y PC1. Esto indica que la correlación es muy negativa y muy fuerte con el score.

```
cor(heptathlon$score, heptathlon_pca$x[,1])
```

```
## [1] -0.9931168
plot(heptathlon$score, heptathlon_pca$x[,1])
```



objeto prcomp

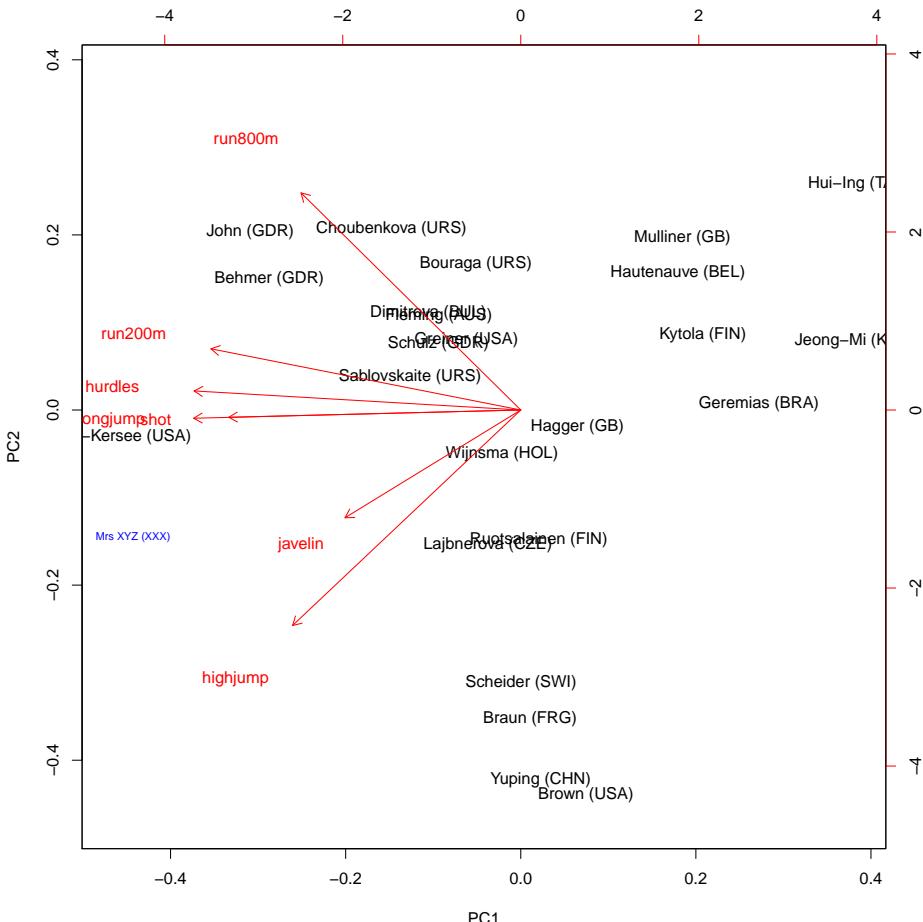
```
class(heptathlon_pca)
```

```
## [1] "prcomp"
```

Podemos usar predict, ej.:

```
new.athlete <- as.data.frame(t(as.vector(c(3.5,2,13,5,7,41,33))))
colnames(new.athlete) <- c("hurdles", "highjump",
                           "shot", "run200m", "longjump",
                           "javelin", "run800m")
rownames(new.athlete) <- "Mrs XYZ (XXX)"
pp<-predict(heptathlon_pca,newdata = new.athlete)
pp
```

```
##                  PC1          PC2          PC3          PC4          PC5          PC6
## Mrs XYZ (XXX) -4.354879 -1.430366 -1.130194 -1.901377 -2.089963 -0.8654821
##                  PC7
## Mrs XYZ (XXX)  1.253643
biplot(heptathlon_pca)
text(pp[,1],pp[,2],"Mrs XYZ (XXX)",col="blue",cex=.65)
```



9.2. K-medias

K-means Clustering es un algoritmo de aprendizaje no supervisado que intenta agrupar datos basados en su similitud. El aprendizaje sin supervisión significa que no hay resultados que predecir, y el algoritmo sólo trata de encontrar patrones en los datos. En k-means clustering, tenemos que especificar el número de clusters en los que queremos que se agrupen los datos. El algoritmo asigna aleatoriamente cada observación a un cúmulo, y encuentra el centroide de cada cúmulo. Luego, el algoritmo itera a través de dos pasos:

1. Reasigne los puntos de datos al cluster cuyo centroide está más cerca.
2. Calcular el nuevo centroide de cada cúmulo.

Estos dos pasos se repiten hasta que la variación dentro del conglomerado no pueda reducirse más. La variación dentro del conglomerado se calcula como la

suma de la distancia euclídea entre los puntos de datos y sus respectivos centros de conglomerados.

Ejemplo:

```
library(rattle)
data(wine)
head(wine)

##   Type Alcohol Malic  Ash Alcalinity Magnesium Phenols Flavanoids
## 1    1    14.23  1.71 2.43      15.6     127    2.80     3.06
## 2    1    13.20  1.78 2.14      11.2     100    2.65     2.76
## 3    1    13.16  2.36 2.67      18.6     101    2.80     3.24
## 4    1    14.37  1.95 2.50      16.8     113    3.85     3.49
## 5    1    13.24  2.59 2.87      21.0     118    2.80     2.69
## 6    1    14.20  1.76 2.45      15.2     112    3.27     3.39
## Nonflavanoids Proanthocyanins Color  Hue Dilution Proline
## 1          0.28          2.29  5.64 1.04     3.92    1065
## 2          0.26          1.28  4.38 1.05     3.40    1050
## 3          0.30          2.81  5.68 1.03     3.17    1185
## 4          0.24          2.18  7.80 0.86     3.45    1480
## 5          0.39          1.82  4.32 1.04     2.93    735
## 6          0.34          1.97  6.75 1.05     2.85    1450
```

Siempre se recomienda estandarizar las variables

```
wine.stand <- scale(wine[ -1 ]) # To standarize the variables
# K-Means
k.means.fit <- kmeans(wine.stand, 3) # k = 3
attributes(k.means.fit)

## $names
## [1] "cluster"      "centers"       "totss"        "withinss"
## [5] "tot.withinss" "betweenss"     "size"         "iter"
## [9] "ifault"
##
## $class
## [1] "kmeans"
```

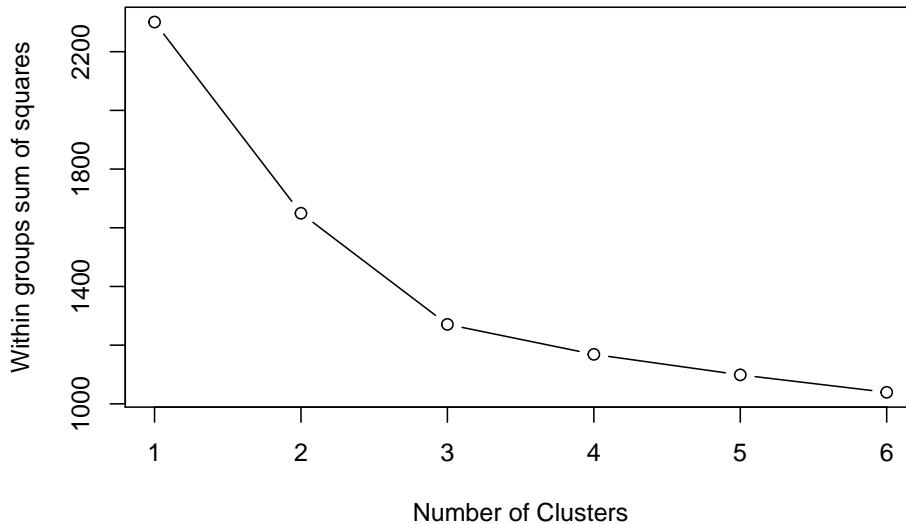
Una pregunta fundamental es cómo determinar el valor del parámetro k . Si consideramos el porcentaje de varianza explicado en función del número de grupos:

Uno debe elegir un número de grupos para que la incorporación de otro grupo no dé un mejor modelado de los datos. Más precisamente, si el porcentaje de varianza explicado por los clusters se traza de acuerdo al número de grupos, los primeros clusters añadirán mucha información (explican mucha varianza), pero en algún momento la ganancia marginal disminuirá, dando un ángulo en el gráfico. En este punto se elige el número de racimos, de ahí el “criterio del

codo”.

```
wssplot <- function(data, nc=15, seed=1234){
  wss <- (nrow(data)-1)*sum(apply(data,2,var))
  for (i in 2:nc){
    set.seed(seed)
    wss[i] <- sum(kmeans(data, centers=i)$withinss)}
  plot(1:nc, wss, type="b", xlab="Number of Clusters",
       ylab="Within groups sum of squares")}

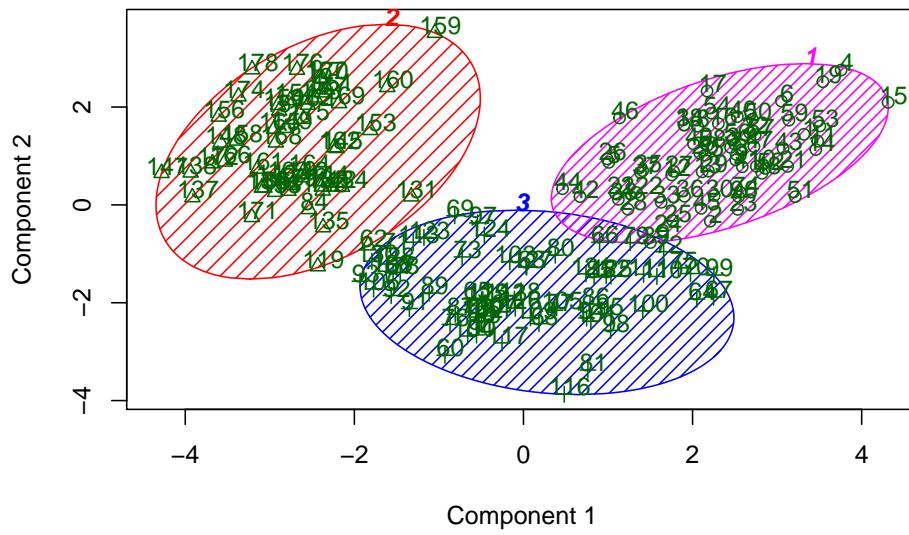
wssplot(wine.stand, nc=6)
```



La `library(cluster)` permite representar los datos en dos dimensiones:

```
library(cluster)
clusplot(wine.stand, k.means.fit$cluster,
         main='2D representation of the Cluster solution',
         color=TRUE, shade=TRUE,
         labels=2, lines=0)
```

2D representation of the Cluster solution



These two components explain 55.41 % of the point variability.

Sabiendo que hay tres tipos de `wine$Type` wines, podemos calcular la matriz de confusión.

```
table(wine$type)

##
##   1   2   3
## 59  71  48

table(wine[,1],k.means.fit$cluster)

##
##      1   2   3
## 1 59  0  0
## 2  3  3  65
## 3  0  48  0
```

9.3. Cluster jerárquico

Los métodos jerárquicos utilizan una matriz de distancia como entrada para el algoritmo de agrupación. La elección de una métrica apropiada influirá en la forma de los cúmulos, ya que algunos elementos pueden estar próximos entre sí de acuerdo a una distancia y más separados de acuerdo a otra.

```
d <- dist(wine.stand, method = "euclidean") # Euclidean distance matrix.
```

El criterio de desviación mínima de Ward minimiza la desviación total dentro

del grupo de empresas.

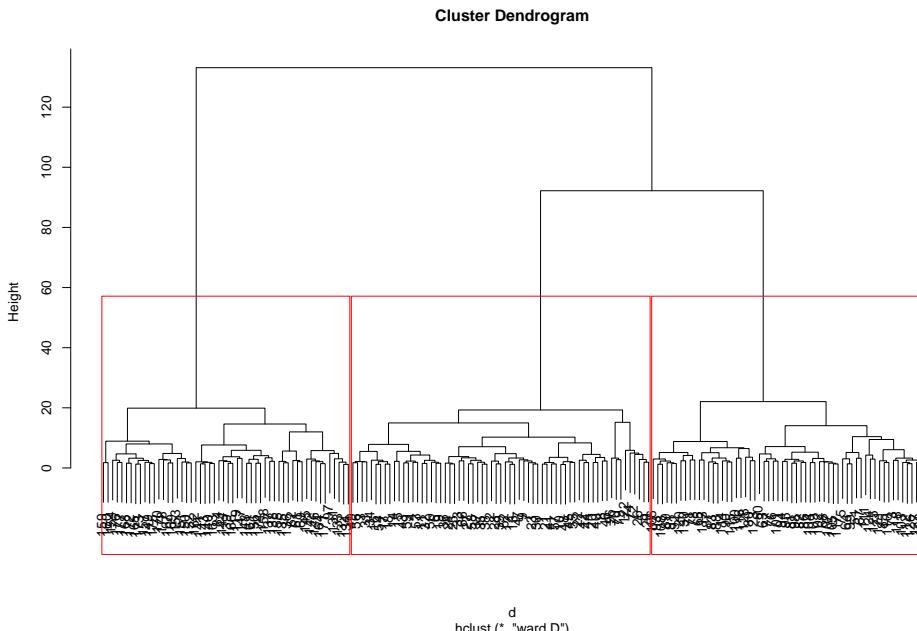
```
H.fit <- hclust(d, method="ward")

## The "ward" method has been renamed to "ward.D"; note new "ward.D2"
class(H.fit)

## [1] "hclust"
```

La opción `plot` devuelve un `hclust` que muestra el dendograma:

```
plot(H.fit) # display dendrogram
groups <- cutree(H.fit, k=3) # cut tree into 5 clusters
# draw dendrogram with red borders around the 5 clusters
rect.hclust(H.fit, k=3, border="red")
```



```
table(wine[,1],groups)
```

```
##      groups
##      1 2 3
## 1 58 1 0
## 2 7 58 6
## 3 0 0 48
```


Capítulo 10

Introducción a las redes neuronales artificiales

En esta sección describimos una clase de métodos de aprendizaje que se desarrollaron por separado en diferentes campos: estadística e inteligencia artificial basadas en modelos esencialmente idénticos.

Las redes neuronales artificiales se inspiran en el comportamiento conocido del cerebro humano (principalmente el referido a las neuronas y sus conexiones), trata de crear modelos artificiales que solucionen problemas difíciles de resolver mediante técnicas algorítmicas convencionales.

La idea central es extraer las combinaciones lineales de las variables de entrada como características derivadas y, a continuación, modelar el objetivo como una función no lineal de estas características. El resultado es un poderoso método de aprendizaje, con amplias aplicaciones en muchos campos.

la neurona artificial pretende mimetizar las características más importantes de la neurona biológica. En general, recibe las señales de entrada de las neuronas vecinas ponderadas por los pesos de las conexiones. La suma de estas señales ponderadas proporciona la entrada total o neta de la neurona y, mediante la aplicación de una función matemática - denominada función de salida - , sobre la entrada neta, se calcula un valor de salida, el cual es enviado a otras neuronas.

Tanto los valores de entrada a la neurona como su salida pueden ser señales excitatorias (cuando el valor es positivo) o inhibitorias (cuando el valor es negativo).

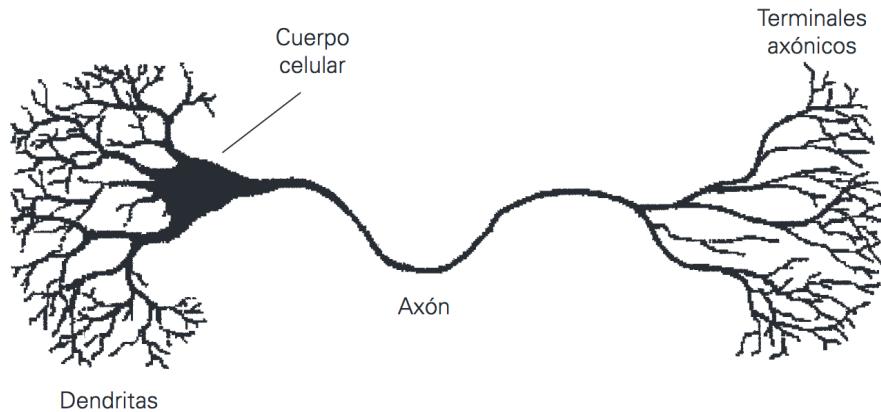


Figura 10.1: Estructura general de una neurona biológica.

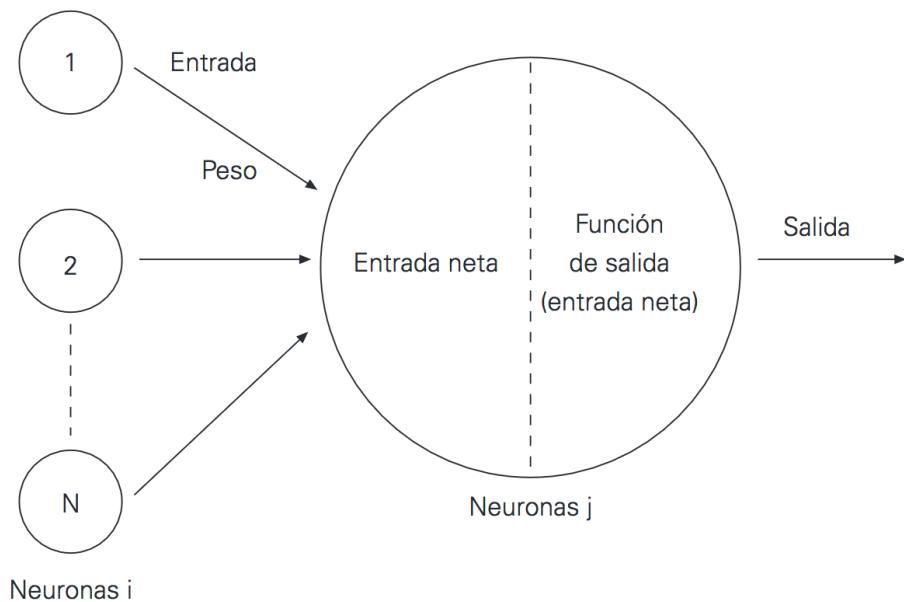


Figura 10.2: Funcionamiento general de una neurona artificial.

10.1. Arquitecturas de las RNA

Las neuronas que componen una RNA se organizan de forma jerárquica formando capas. Una capa o nivel es un conjunto de neuronas cuyas entradas de información provienen de la misma fuente (que puede ser otra capa de neuronas) y cuyas salidas de información se dirigen al mismo destino (que puede ser otra capa de neuronas). En este sentido, se distinguen tres tipos de capas: la capa de entrada recibe la información del exterior; las capas ocultas son aquellas cuyas entradas y salidas se encuentran dentro del sistema y, por tanto, no tienen contacto con el exterior; por último, la capa de salida envía la respuesta de la red al exterior.

En función de la organización de las neuronas en la red formando capas o agrupaciones podemos encontrarnos con dos tipos de arquitecturas básicas: redes multicapa (multi-layer) y redes monocapa (single-layer).

10.2. Ejemplo: datos Boston housing

Vamos a utilizar el conjunto de datos de **Boston** en el paquete MASS. Recordemos que el conjunto de datos de **Boston** es una colección de datos sobre valores de vivienda en los suburbios de Boston. Nuestro objetivo es predecir el valor medio de las viviendas ocupadas por sus propietarios ('medv') utilizando todas las demás variables continuas disponibles.

```
set.seed(500)
library(MASS)
data <- Boston
```

Primero tenemos que comprobar que no hay datos faltantes, de lo contrario tenemos que arreglar el conjunto de datos.

```
apply(data, 2, function(x) sum(is.na(x)))

##      crim      zn    indus      chas      nox      rm      age      dis      rad
##      0       0       0       0       0       0       0       0       0       0
##      tax      ptratio    black     lstat     medv
##      0       0       0       0       0
```

Hemos comprobado que no hay datos faltantes (NA). Procedemos dividiendo aleatoriamente los datos en un conjunto de entrenamiento y un conjunto de prueba, luego ajustamos un modelo de regresión lineal y lo probamos en el conjunto de prueba. Nótese que usamos la función `glm()` en lugar de la función `lm()` esto será útil más tarde al validar el modelo lineal.

```
index <- sample(1:nrow(data), round(0.75*nrow(data)))
train <- data[index,]
test <- data[-index,]
lm.fit <- glm(medv ~ ., data=train)
```

```

summary(lm.fit)

##
## Call:
## glm(formula = medv ~ ., data = train)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -14.9143  -2.8607  -0.5244   1.5242  25.0004
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 43.469681  6.099347  7.127 5.50e-12 ***
## crim        -0.105439  0.057095 -1.847 0.065596 .
## zn          0.044347  0.015974  2.776 0.005782 **
## indus       0.024034  0.071107  0.338 0.735556
## chas         2.596028  1.089369  2.383 0.017679 *
## nox        -22.336623  4.572254 -4.885 1.55e-06 ***
## rm          3.538957  0.472374  7.492 5.15e-13 ***
## age         0.016976  0.015088  1.125 0.261291
## dis        -1.570970  0.235280 -6.677 9.07e-11 ***
## rad          0.400502  0.085475  4.686 3.94e-06 ***
## tax         -0.015165  0.004599 -3.297 0.001072 **
## ptratio     -1.147046  0.155702 -7.367 1.17e-12 ***
## black        0.010338  0.003077  3.360 0.000862 ***
## lstat       -0.524957  0.056899 -9.226 < 2e-16 ***
##
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 23.26491)
##
## Null deviance: 33642 on 379 degrees of freedom
## Residual deviance: 8515 on 366 degrees of freedom
## AIC: 2290
##
## Number of Fisher Scoring iterations: 2
pr.lm <- predict(lm.fit,test)
MSE.lm <- sum((pr.lm - test$medv)^2)/nrow(test)

```

La función `sample(x,size)` simplemente produce un vector del tamaño especificado de muestras seleccionadas aleatoriamente desde el vector x. Por defecto el muestreo es sin reemplazo: el índice es esencialmente un vector aleatorio de muestras.

Como se trata de un problema de regresión, vamos a utilizar el error cuadrático

medio (ECM) como medida de lo lejos que están nuestras predicciones de los datos reales.

10.3. Ajuste de la red neuronal

Antes de entrenar una red neural, es necesario hacer algún paso previo. Como primer paso, vamos a abordar el preprocesamiento de datos.

- Es una buena práctica normalizar los datos antes de entrenar una red neuronal.
- Se pueden elegir diferentes métodos para escalar los datos (normalización-z, escala min-max, etc...).
- Elegiremos el método min-max y escalaremos los datos en el intervalo $[0, 1]$. Por lo general, la escala en los intervalos $[0, 1]$ ó $[-1, 1]$ tiende a dar mejores resultados.
- Por lo tanto, escalamos y dividimos los datos antes de seguir adelante:

```
maxs <- apply(data, 2, max)
mins <- apply(data, 2, min)

scaled <- as.data.frame(scale(data, center = mins, scale = maxs - mins))

train_ <- scaled[index,]
test_ <- scaled[-index,]
```

No hay una regla fija en cuanto a cuántas capas y neuronas usar, aunque hay varias reglas generales más o menos aceptadas. Normalmente, si es necesario, una capa oculta es suficiente para un gran número de aplicaciones.

En cuanto al número de neuronas, debería estar entre el tamaño de la capa de entrada y el tamaño de la capa de salida, normalmente 2/3 del tamaño de entrada. No hay garantía de que ninguna de estas reglas se ajuste mejor a su modelo de modo que es recomendable probar con diferentes combinaciones.

Para este ejemplo, vamos a usar 2 capas ocultas con esta configuración: 13:5:3:1.

- La capa de entrada tiene 13 entradas,
- las dos capas ocultas tienen 5 y 3 neuronas y
- la capa de salida tiene, por supuesto, una sola salida ya que estamos haciendo regresión.

Vamos a entrar en la red:

```
library(neuralnet)
n <- names(train_)
```

```
f <- as.formula(paste("medv ~", paste(n[!n %in% "medv"], collapse = " + ")))
nn <- neuralnet(f,data=train_,hidden=c(5,3),linear.output=TRUE)
```

Nota:

- La expresión `y~.` no es aceptada por la función `neuralnet()`.
- El argumento `oculto` acepta un vector con el número de neuronas para cada capa oculta, mientras que el argumento `linear.output` se usa para especificar si queremos hacer una regresión `linear.output=TRUE` o clasificación `linear.output=FALSE`.

La librería `neuralnet` proporciona una buena herramienta para representar gráficamente el modelo con los pesos en cada conexión:

```
plot(nn)
```

Las líneas negras muestran las conexiones entre cada capa y los pesos de cada conexión, mientras que las líneas azules muestran el término de sesgo añadido en cada paso. El sesgo puede ser pensado como el intercepto de un modelo lineal.

La red neuronal es esencialmente una caja negra, por lo que no podemos decir mucho sobre el ajuste, los pesos y el modelo. Basta decir que el algoritmo de entrenamiento ha convergido y por lo tanto el modelo está listo para ser utilizado.

10.4. Predicción con una red neuronal

Ahora podemos tratar de predecir los valores para el equipo de prueba y calcular el ECM. Recuerda que la red producirá una predicción normalizada, por lo que necesitamos reducirla para hacer una comparación significativa (o simplemente una predicción simple).

```
library(neuralnet)
pr.nn <- neuralnet::compute(nn,test_[,1:13])

pr.nn_ <- pr.nn$net.result*(max(data$medv)-min(data$medv))+min(data$medv)
test.r <- (test_$medv)*(max(data$medv)-min(data$medv))+min(data$medv)

MSE.nn <- sum((test.r - pr.nn_)^2)/nrow(test_)
MSE.nn

## [1] 15.75184

Podemos ahora comparar los dos ECM:
print(paste(MSE.lm,MSE.nn))

## [1] "21.6297593507225 15.7518370200153"
```

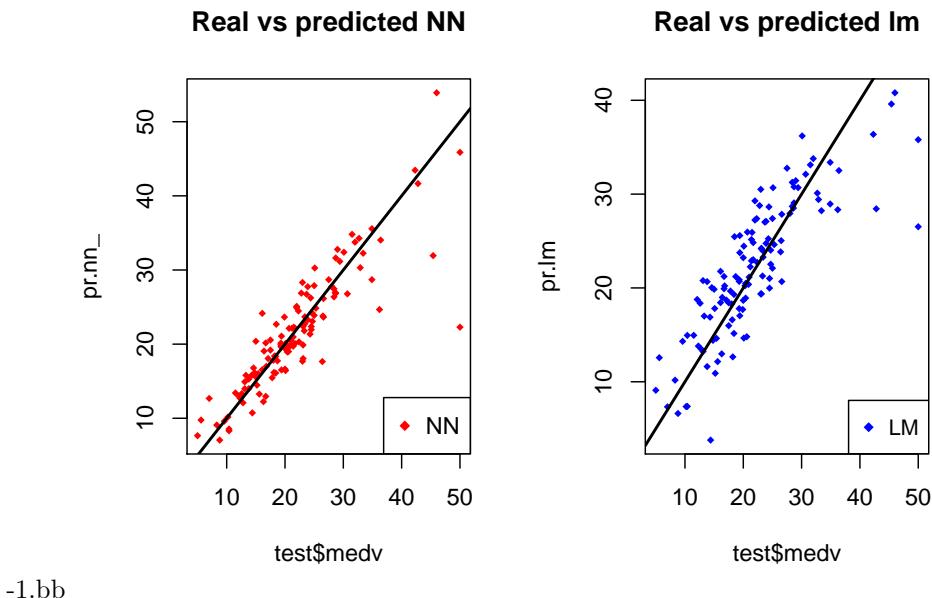
Aparentemente la red neuronal está haciendo un mejor trabajo que el modelo lineal en la predicción de `medv`. Una vez más, hay que tener cuidado porque este resultado depende de la división de prueba de entrenamiento realizada anteriormente.

A continuación, vamos a realizar una rápida validación cruzada para tener más confianza en los resultados.

```
par(mfrow=c(1,2))

plot(test$medv,pr.nn_,col='red',main='Real vs predicted NN',pch=18,cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend='NN',pch=18,col='red')

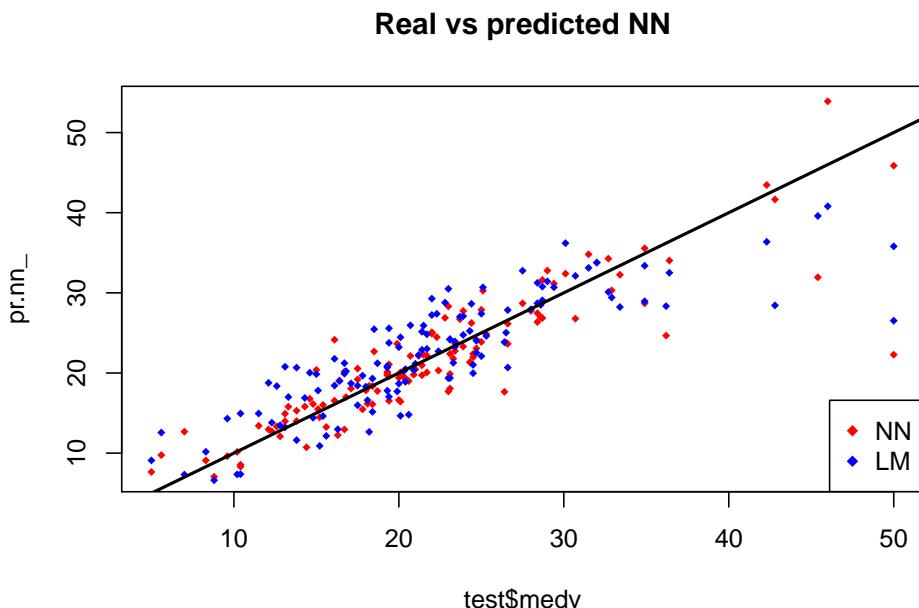
plot(test$medv,pr.lm,col='blue',main='Real vs predicted lm',pch=18, cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend='LM',pch=18,col='blue', cex=.95)
```



-1.bb

Inspeccionando visualmente podemos ver que las predicciones hechas por la red neuronal están (en general) más concentradas alrededor de la línea (una alineación perfecta con la línea indicaría un ECM de 0 y por lo tanto una predicción perfecta ideal) que las hechas por el modelo lineal.

```
plot(test$medv,pr.nn_,col='red',main='Real vs predicted NN',pch=18,cex=0.7)
points(test$medv,pr.lm,col='blue',pch=18,cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend=c('NN','LM'),pch=18,col=c('red','blue'))
```



10.5. Evaluando la predicción mediante validación cruzada

La validación cruzada es otro paso muy importante en la construcción de modelos predictivos. Aunque existen diferentes tipos de métodos de validación cruzada, la idea básica es repetir el proceso siguiente varias veces:

División Entrenamiento-Prueba:

- Realizar la división entre muestra de entrenamiento-prueba.
- Ajustar el modelos al conjunto de entrenamiento.
- Comprobar el modelo en el conjunto de prueba.
- Calcular el error de predicción.
- Repetir el proceso K veces.

Luego, calculando el error promedio, podemos hacernos una idea de cómo le está yendo al modelo.

Vamos a implementar una validación cruzada usando un bucle para la red neuronal y la función `cv.glm()` en el paquete `boot` para el modelo lineal. Para $K = 10$.

```
library(boot)
set.seed(200)
```

```

lm.fit <- glm(medv~.,data=data)
cv.glm(data,lm.fit,K=10)$delta[1]

## [1] 23.8356

set.seed(450)
cv.error <- NULL
k <- 10

for(i in 1:k){
  index <- sample(1:nrow(data),round(0.9*nrow(data)))
  train.cv <- scaled[index,]
  test.cv <- scaled[-index,]

  nn <- neuralnet(f,data=train.cv,hidden=c(5,2),linear.output=T)

  pr.nn <- neuralnet::compute(nn,test.cv[,1:13])
  pr.nn <- pr.nn$net.result*(max(data$medv)-min(data$medv))+min(data$medv)

  test.cv.r <- (test.cv$medv)*(max(data$medv)-min(data$medv))+min(data$medv)

  cv.error[i] <- sum((test.cv.r - pr.nn)^2)/nrow(test.cv)
}

}

```

Calculamos el ECM promedio y graficamos los resultados como una gráfica de caja.

```

mean(cv.error)

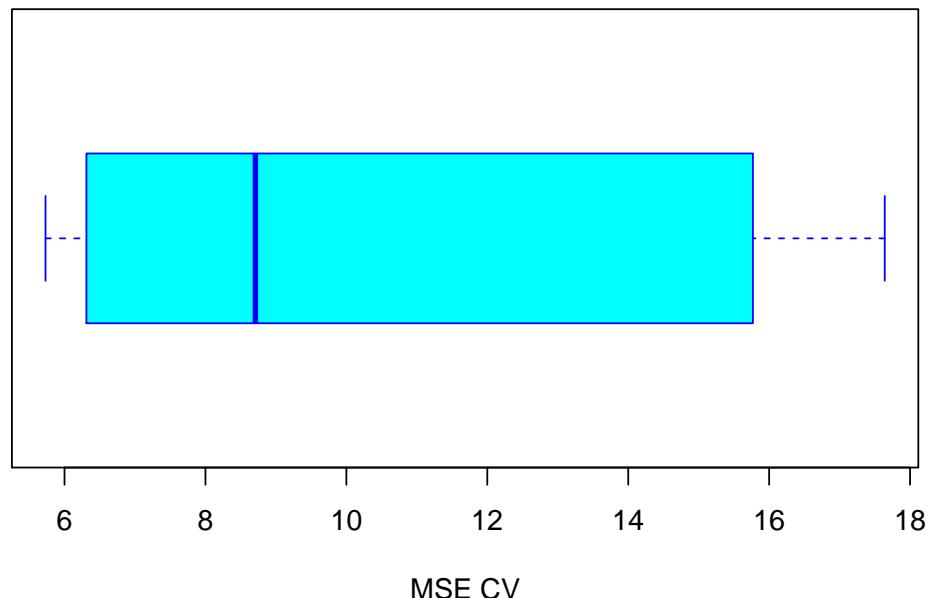
## [1] 10.32698
cv.error

## [1] 17.640653 6.310575 15.769519 5.730131 10.520947 6.121161 6.389967
## [8] 8.004786 17.369282 9.412778

Boxplot
boxplot(cv.error,xlab='MSE CV',col='cyan',
        border='blue',names='CV error (MSE)',
        main='CV error (MSE) for NN',horizontal=TRUE)

```

CV error (MSE) for NN



Como se puede ver, la media del ECM de la red neuronal (10.33) es inferior al del modelo lineal, aunque parece haber cierto grado de variación en los ECMs de la validación cruzada. Esto puede depender de la división de los datos o de la inicialización aleatoria de los pesos en la red neuronal. Al ejecutar la simulación en diferentes momentos con diferentes semillas, puede obtener una estimación puntual más precisa para el ECM medio.

10.5.1. Comparación de modelos RNA

A continuación vamos a comparar 2 modelos:

1. 2 capas ocultas de 5 y 3.
2. 1 capa oculta de 8

```
Boston.nn.5.3 <- neuralnet(f
    , data=train_
    , hidden=c(5,3)
    , linear.output=TRUE)

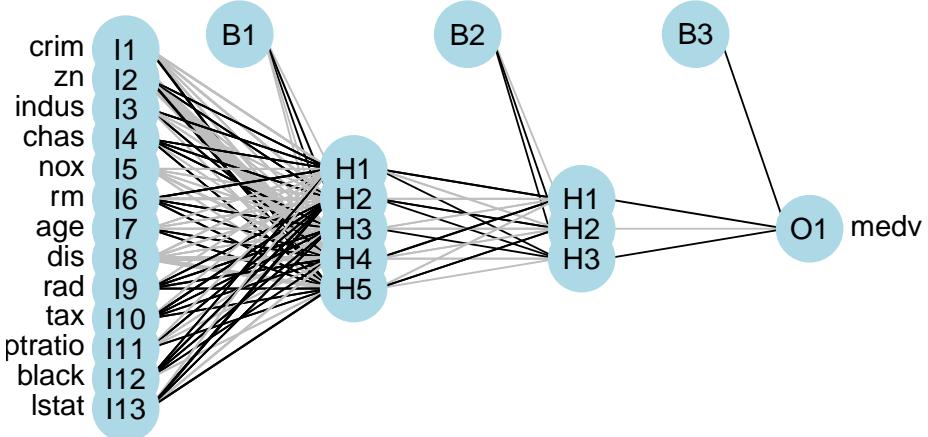
Boston.nn.8 <- neuralnet(f
    , data=train_
    , hidden=8
    , linear.output=TRUE)
```

Como alternativa a la función `plot.nn()`, la librería `NeuralNetTools` incluye

10.5. EVALUANDO LA PREDICCIÓN MEDIANTE VALIDACIÓN CRUZADA189

funciones gráficas más elegantes. Este elegante gráfico resuelve el problema del desorden visual utilizando el grosor de la línea para representar la magnitud del peso y el color de la línea para representar el signo de peso (negro = positivo, gris = negativo).

```
library(NeuralNetTools)
plotnet(Boston.nn.5.3)
```



```
plotnet(Boston.nn.8)
```

