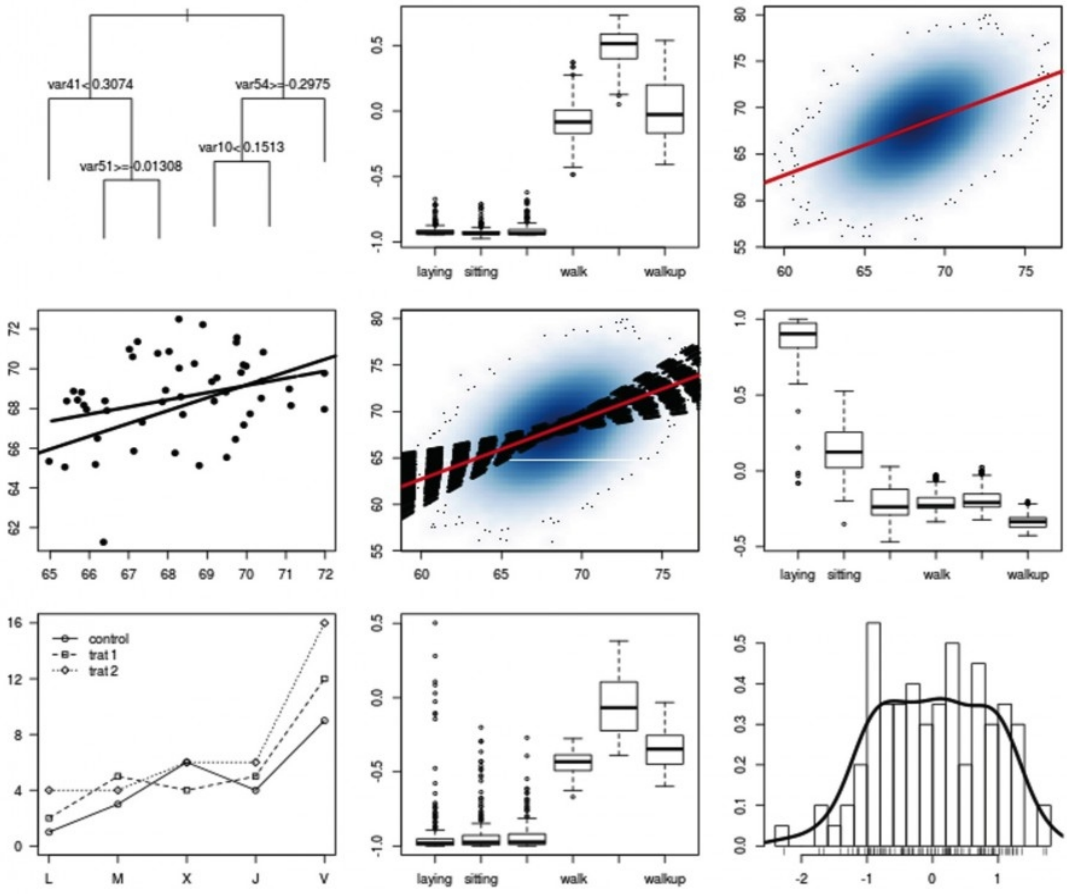
Fundamentos estadísticos para investigación. Introducción a R.



Antonio Maurandi López
Laura del Río Alonso
Carlos Balsalobre Rodríguez

Fundamentos estadísticos para investigación. Introducción a R.

Antonio Maurandi López
Licenciado en CC Matemáticas. Facultativo de estadística del Servicio de Apoyo a la Investigación (SAI), profesor asociado del Área de Didáctica de las CC Matemáticas de la Universidad de Murcia. Director del curso "Fundamentos estadísticos para investigación. Introducción a R." Miembro fundador de la Comunidad R-Hispano.
Dra. Laura del Río Alonso
Profesora titular del Departamento de Sanidad Animal de la Universidad de Murcia. Directora del curso"Fundamentos estadísticos para investigación. Introducción a R."
Carlos Balsalobre Ródríguez
Licenciado en Matemáticas por la Universidad de Murcia. Profesor asociado del Departamento de Estadística e Investigación Operativa de la Universidad Miguel Hernández de Elche.

En Murcia, a 23 de marzo de 2013

Fundamentos estadísticos para investigación.Introducción a R.
Antonio Maurandi, Laura del Río y Carlos Balsalobre
ISBN papel: 978-84-686-3628-3
ISBN pdf: 978-84-686-3629-0
Editado por Bubok Publishing S.L.
Impreso en España/Printed in Spain

Al profesor José Antonio Palazón:
¿Está usted de broma señor Palazón?

Agradecimientos

Los autores queremos agradecer la gran aportación de ***María Elvira Ferre Jaén*** *y* ***Álvaro Hernández Vicente****, estudiantes del grado de matemáticas, por su excelente trabajo en la maquetación en Markdown y LaTeX del texto original y revisión de errores, pero sobre todo, por el talento demostrado en su forma de trabajar.*

Prefacio

En mayo de 2012 en la Universidad de Murcia se organizó el curso *Fundamentos estadísticos para investigación. Introducción a R.* El material preparado para él se recoge en este libro; se han incorporado las mejoras y correcciones que surgieron del desarrollo de las sesiones de trabajo.

El curso nace, inicialmente, con la vocación de ayudar a investigadores de todas las áreas de conocimiento, incluso para algunos ya iniciados en estadística, (muchas veces usuarios de otros programas: SPSS, SAS, Minitab, etc...) que pretenden dar *el salto* a R y la vez repasar conceptos de una forma intuitiva, evitando excesivos formalismos. Pero, ¿todas las áreas?, ¿hay algún área en la que no sea necesario el análisis de datos o la estadística? En realidad, hoy día, desde la lingüística en el análisis computacional de textos, a la *simple* representación gráfica de la información, pasando por los estudios epidemiológicos, la biotecnología, la genética, la ecología, las ciencias de la educación, psicometría, econometría, finanzas, ciencias sociales, medicina, matemáticas, bellas artes, etc. requieren antes o después de análisis de datos. R no es solo un programa, es una forma de trabajar y manera de pensar, además admite sabores y versiones personales, pero sobre todo es un lenguaje, más concretamente es la *lingua franca* actual del análisis de datos.

El libro está pensado para gente que al comienzo sabe nada o muy poco de lenguajes formales de programación y con conceptos estadísticos que se sitúan en un nivel bastante inicial, donde para avanzar es preciso una aproximación intuitiva que no exija un gran rigor formal. R no solo puede resolver los problemas clásicos que otros programas

ya resolvían, sino que además, puede ayudar a *pensar y aprender*, sabiendo mejor lo que hacemos, e invitando a hacerlo mejor y disfrutando más; porque, entre otras cosas, podemos aceptar que *sabes lo que haces*.

Creemos fervientemente que el software libre es factor de cohesión social y un vehículo de conocimiento, pensamos que en el entorno universitario debería de primar el uso de software libre en todos los sentidos, ya que, como dice Richard Stallman (fundador de la *Free Software Fundation*): *"porque [el software libre] es el único que permite cumplir con sus misiones fundamentales [de la universidad]: difundir el conocimiento y enseñar a los estudiantes a ser buenos miembros de su comunidad"*.

23 de marzo de 2013. AML

Índice general

Capítulo 1

Introducción a R.

1. ¿Qué es R?

R es un potente lenguaje orientado a objetos y destinado al análisis estadístico y la representación de datos. Se trata de software libre que permite su utilización libre y gratuitamente. La comunidad científica internacional lo ha elegido como la *'lingua franca'* del análisis de datos. Y tiene una gran implantación en universidades y cada vez más en mundo empresarial.

Otra definición: "R es un paquete estadístico de última generación al mismo tiempo que un lenguaje de programación".

1.1 ¿Por qué emplear R?

Estadística se puede hacer con miles de paquetes estadísticos, incluso con hojas de cálculo e incluso los más osados con lápiz y papel (aunque la exactitud de las máquinas es una potencia que no podemos desdeñar: cálculo de p-valores, estadísticos exactos...).

¿Qué tiene R que tanto nos gusta?:

- Es libre. Se distribuye bajo licencia GNU, lo cual significa que lo puedes utilizar y ¡mejorar!
- Es multiplataforma, hay versiones para Linux, Windows, Mac, iPhone...
- Se puede analizar en R cualquier tipo de datos.
- Es potente. Es muy potente.
- Su capacidad gráfica difícilmente es superada por ningún otro paquete estadístico.

- Es compatible con 'todos' los formatos de datos (csv, xls, sav, sas...)
- Es ampliable, si quieres añadir algo: ¡empaquétalo!
- Hay miles de técnicas estadísticas implementadas, cada día hay más.
- ...

1.2 Instalación de R

La instalación difiere según el entorno. En Windows es un ejecutable, en Linux se trata de ejecutar una serie de comandos en consola...

Página web de CRAN: http://cran.es.r-project.org/

R 2.15.0

- En Windows: "Download R 2.15.0 for Windows (47 megabytes, 32/64 bit)" es un ejecutable, lo bajamos y lo instalamos.
- Linux: `sudo apt-get update`; `sudo apt-get install r-base` (en más detalle se puede leer: http://numerorojo.wordpress.com/2008/04/27/instalar-r-en-ubuntu/)
- En Mac OX: http://cran.es.r-project.org/bin/macosx/

Lanzar R y salir:

- en Linux: `R`
- en Windows: doble clic en el icono "R"

Para salir de la aplicación ejecutamos en ambos:

```
q()   # para salir
```

1.3 Ayuda en R

Cómo obtenemos ayuda en R.

Función	Acción
`help.start()`	Ayuda general
`help(mean)` o simplemente `?mean`	Función de ayuda

`help.search("mean")`	
`RSiteSearch("mean")`	Ayuda online
`apropos("mean", mode ="function")`	Lista todas las funciones que tienen la palabra 'mean' en el nombre
`data()`	Muestra los conjuntos de datos de ejemplo que hay disponibles

Una ayuda no desdeñable es R-help-es: https://stat.ethz.ch/mailman/listinfo/r-help-es. Es una lista de distribución donde uno puede preguntar y responder dudas que surgen al trabajar con R. Esta lista surgió muy cerca de esta universidad y gracias a ella se han realizado ya 3 jornadas de usuarios de R en español; la primera en Murcia, la segunda en Oviedo y la tercera en Madrid (visita:http://www.r-es.org/).

1.4 Espacio y directorio de trabajo

El *"workspace"* es el espacio de trabajo en que se incluyen todos los objetos definidos por el usuario (ya veremos qué son estos objetos, que incluyen variables, vectores, dataframes...), se almacena en memoria intermedia mientras trabajas con R.

Cuando termina una sesión de R el propio R te pregunta si quieres guardar el *"workspace"* para usos futuros. Este espacio, *"workspace"*, se recarga al volver a iniciar la sesión. Directorio de trabajo o *"working directory"* es el directorio donde por defecto "lee" R. También es donde guardará el workspace al finalizar la sesión y donde buscará un workspace guardado al inicio. Si quieres que R lea un fichero que no esté en *"working directory"* hay que especificar la ruta completa.

Funciones para manejar el *"workspace"*:

Función	Acción
`getwd()`	Muestra el wd
`setwd("midirectorio")`	Ajusta el wd al especificado
`ls()` o `dir()`	Lista lo que hay en el wd
`history()`	Muestra los últimos comandos ejecutados
`savehistory()`	Guarda el historial de comandos, por defecto en .RHistory
`loadhistory()`	Carga el historial de comandos

`save.image("mywspace.R")`	Guarda los objetos del workspace, por defecto en .RData
`load("mywspace.R")`	Carga el workspace mywspace.R

1.5 Cosas varias: asignaciones, variables, comentarios, etc.

Todo lo precedido por almohadillas *'#'* R lo considera un comentario y no lo 'interpreta'.
En R empleamos el operador *'<-'* para hacer asignaciones, y una variable se crea *"al vuelo"*, esto es, en el mismo instante en el que la asignas. Es más, no puedes declararlas con anterioridad y dejarlas vacías.

Así

```
mivariable <- 7  # asigna el número 7 a una variable
```

Si quiero ver qué hay en mi variable ejecutaré:

```
mivariable
```

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

Nota: Cualquier cosa que se puede asignar a una variable es un *'objeto'* de R.

Si guardo el *"workspace"* ahora guardaré mi nueva variable en él.

1.6 Una sesión "tipo" en R

```
>getwd()
  [1] "C:/Documents and Settings/Administrador/Mis documentos"

>setwd("C:/CursoR_UMU") # ¡¡Ojo con la barra!!
```

```
set.seed(1)
x <- runif(20)
x
```

```
##  [1] 0.26551 0.37212 0.57285 0.90821 0.20168
0.89839 0.94468 0.66080
##  [9] 0.62911 0.06179 0.20597 0.17656 0.68702
0.38410 0.76984 0.49770
## [17] 0.71762 0.99191 0.38004 0.77745
```

```
summary(x)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0618  0.3450  0.6010  0.5550  0.7720  0.9920
```

```
hist(x)
```

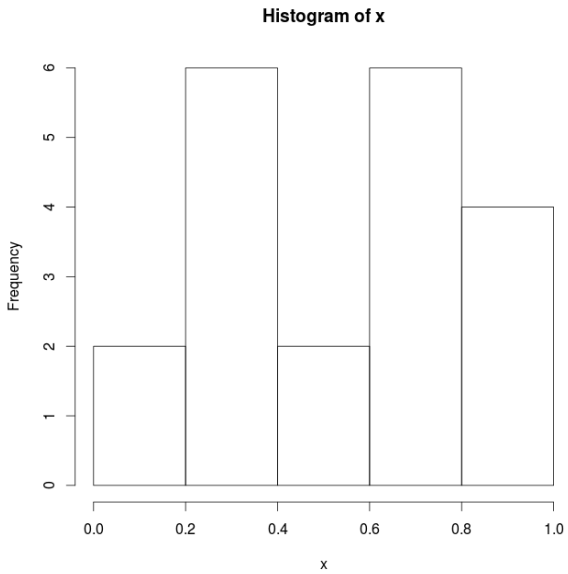


Figura 1.1: Histograma

```
savehistory()

save.image()
q()
```

1.7 R como calculadora

Podemos emplear R como una calculadora.

```
45 + 23
```

```
## [1] 68
```

```
100/4
```

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

```
sqrt(25)
```

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

```
# usando variables
x <- 100/4
5
```

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

```
pi
```

```
## [1] 3.142
```

```
r = 5
area <- 2 * pi + r
area
```

```
## [1] 11.28
```

```
c(25, 100, 2, 3) * 5
```

```
## [1] 125 500  10  15
```

Nota: Cambia el número de decimales por defecto con `options(digits=12)` (para saber en que está establecido usa `getOption("digits")`).

1.8 En R trabajamos con scripts

Para cargar un fichero con instrucciones, un *script*, empleamos la instrucción/función `'source()'`.

```
source("fichero_de_comandos.r")
```

Un fichero que crease un histograma de una curva normal debería de contener, por ejemplo, estas instrucciones:

```
x <- rnorm(100)
# crea 100 observaciones aleatorias de una normal tipificada
hist(x)
```

Así que creamos un fichero de texto plano con el contenido anterior, lo guardamos con el nombre "script.R" (en realidad da lo mismo la extensión, es una convención). Ejecutamos:

```
source("script.R")
```

Ejercicio: Ejecutar mediante el comando `source` el fichero "`01.ej.source.R.grafico.histyNorm.R`"

1.9 Ampliar la funcionalidad de R. Packages

Con la instalación simple de R tenemos muchísimas posibilidades, no obstante existen multitud de módulos opcionales que llamamos paquetes, *"packages"* en inglés. Los paquetes son colecciones de funciones y datos.

El directorio de tu PC donde se almacenan los *packages* es denominado *"library"*

```
.libPaths()
[1] "C:/Archivos de programa/R/R-2.14.1/library"
```

Para ver qué paquetes tienes instalados empleamos la función `library()`. No es lo mismo instalar que cargar un paquete. La instrucción `search()` nos dice que paquetes están instalados y cargados en el sistema listos para usarse.

Instalamos un paquete con la instrucción
`install.packages("nombre_paquete")`
sólo instalamos una vez cada paquete, cargamos el paquete con la instrucción `library("nombre_paquete")` y tendremos que ejecutar este comando en cada sesión en que queramos emplear dicho paquete.

```
install.packages("foreign")

---Please select a CRAN mirror for use in this session---
    probando la URL
'http://cran.es.r-project.org/bin/windows/contrib/
2.14/foreign_0.8-49.zip'
Content type 'application/zip' length 221650 bytes (216 Kb)
URL abierta
downloaded 216 Kb
package 'foreign' successfully unpacked and MD5 sums checked
The downloaded packages are in
C:\WINDOWS\Temp\RtmpO20ggV\downloaded_packages


library("foreign")
Mensajes de aviso perdidos
package 'foreign' was built under R version 2.14.2
```

Los paquetes disponibles están en CRAN, específicamente en http://cran.r-project.org/

Normalmente son necesarios más paquetes que los que vienen por defecto, así que siempre se está instalando paquetes nuevos. En este curso veremos algunos de los más usuales.

Para obtener información sobre un paquete empleamos la función
`help(package="nombredelpaquete")`.

Nota: ¡En el momento de escribir este manual existen 3760 paquetes!

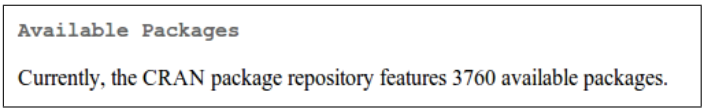


Figura 1.2: Paquetes disponibles

Ejercicio: Visitar la web de CRAN: http://cran.r-project.org/

Nota. Los *Task View* sin colecciones de paquetes para trabajos concretos o para áreas de trabajo concretas. Instalándose una *Task View* automáticamente conjuntos enteros de paquetes se instalan sin tener que ir instalando paquete a paquete.

Bayesian	Bayesian Inference
ChemPhys	Chemometrics and Computational Physics
ClinicalTrials	Clinical Trial Design, Monitoring, and Analysis
Cluster	Cluster Analysis & Finite Mixture Models
DifferentialEquations	Differential Equations
Distributions	Probability Distributions
Econometrics	Computational Econometrics
Environmetrics	Analysis of Ecological and Environmental Data
ExperimentalDesign	Design of Experiments (DoE) & Analysis of Experimental Data
Finance	Empirical Finance
Genetics	Statistical Genetics
Graphics	Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization
HighPerformanceComputing	High-Performance and Parallel Computing with R
MachineLearning	Machine Learning & Statistical Learning
MedicalImaging	Medical Image Analysis
Multivariate	Multivariate Statistics
NaturalLanguageProcessing	Natural Language Processing
OfficialStatistics	Official Statistics & Survey Methodology
Optimization	Optimization and Mathematical Programming
Pharmacokinetics	Analysis of Pharmacokinetic Data
Phylogenetics	Phylogenetics, Especially Comparative Methods
Psychometrics	Psychometric Models and Methods
ReproducibleResearch	Reproducible Research
Robust	Robust Statistical Methods
SocialSciences	Statistics for the Social Sciences
Spatial	Analysis of Spatial Data
Survival	Survival Analysis
TimeSeries	Time Series Analysis
gR	gRaphical Models in R

Figura 1.3: Task Views

To automatically install these views, the ctv package needs to be installed, e.g., via

```
install.packages("ctv")
library("ctv")
```

and then the views can be installed via `install.views` or `update.views` (which first assesses which of the packages are already installed and up-to-date), e.g.,

```
install.views("Econometrics")
```

or
`update.views("Econometrics")`

1.10 Otra sesión "tipo" de R

Probemos a comunicarnos con R y ver su versatilidad e interactividad.

```
peso <- c(4.4, 5.4, 6.4, 3.2, 7.5, 3, 6.1, 3.1, 6.1, 7, 3.4)
# c() concatena y crea un vector
edad <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 1)
edad
```

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

```
peso
```

```
##  [1] 4.4 5.4 6.4 3.2 7.5 3.0 6.1 3.1 6.1 7.0 3.4
```

```
mean(peso)  # función media
```

```
## [1] 5.055
```

```
mean(edad)
```

```
## [1] 4.6
```

```
cor(peso, edad)  # coeficiente de correlación de Pearson
```

```
## Error: dimensiones incompatibles
```

```
length(edad)
```

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

```
length(peso)  # función longitud/dimensión
```

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

```
edad <- c(edad, 5)  # ¡Ojo! le añadimos a edad una
       observación más
length(edad)
```

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

```
cor(peso, edad)
```

```
## [1] -0.1883
```

```
plot(edad, peso)
```

```
q()
```

1.11 RStudio. Una GUI para R multiplataforma

Una de las cosas que menos gustan en R es que *se ve diferente en Windows que en Linux*. Algo que suele molestar a los usuarios algo experimentados en R es que los menús te limiten (algo parecido le pasa a los usuarios de linux). En general el usuario de R no quiere que los menús de la aplicación le sugieran qué hacer con sus datos, más bien simplemente quiere "decirle" al software qué hacer con los datos, es él el que manda, *"The Boss"*. Esto se hace mediante comandos y un poco de formación.

Una programa que soluciona en gran medida estos problemas es **RStudio** http://rstudio.org/.

RStudio es una GUI, "Graphical user interface" para R programada en C#, multiplataforma (Windows, Linux y Mac). Que aúna todos los entornos y guarda la filosofía de los comandos aportando algunas 'ayudas' que hacen más llevadero el día a día.

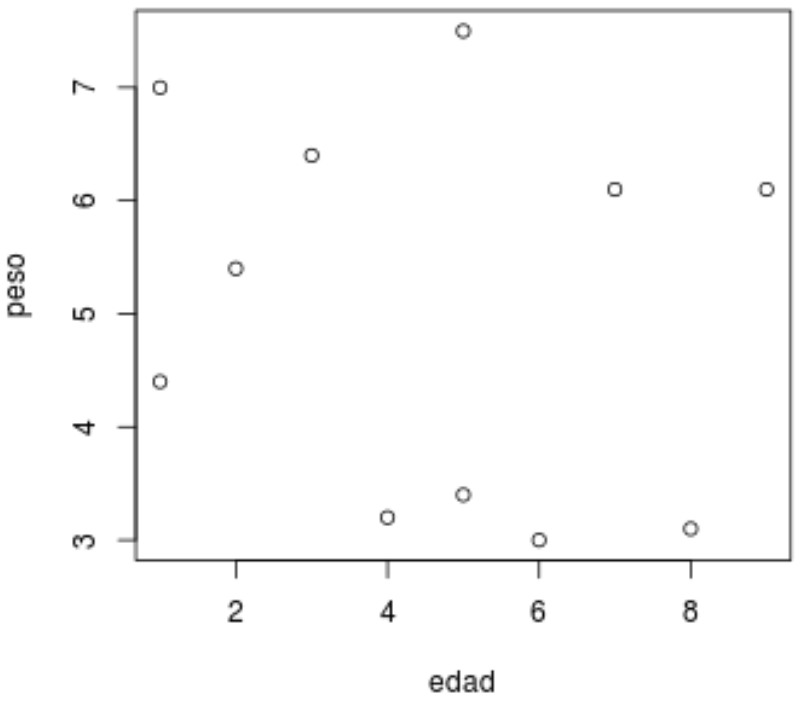


Figura 1.4: Gráfico de dispersión

Otra definición de RStudio™: es un entorno libre y de código abierto de desarrollo integrado (IDE) de R. Se puede ejecutar en el escritorio (Windows, Mac o Linux) o incluso a través de Internet mediante el servidor RStudio.

Entre otras cosas encontramos que RStudio nos permite:

- Abrir varios scripts a la vez.
- Ejecutar pedazos de código con sólo marcarlo en los script.
- Mostrar el workspace.
- Mostrar los objetos del workspace.
- Integra la ayuda.

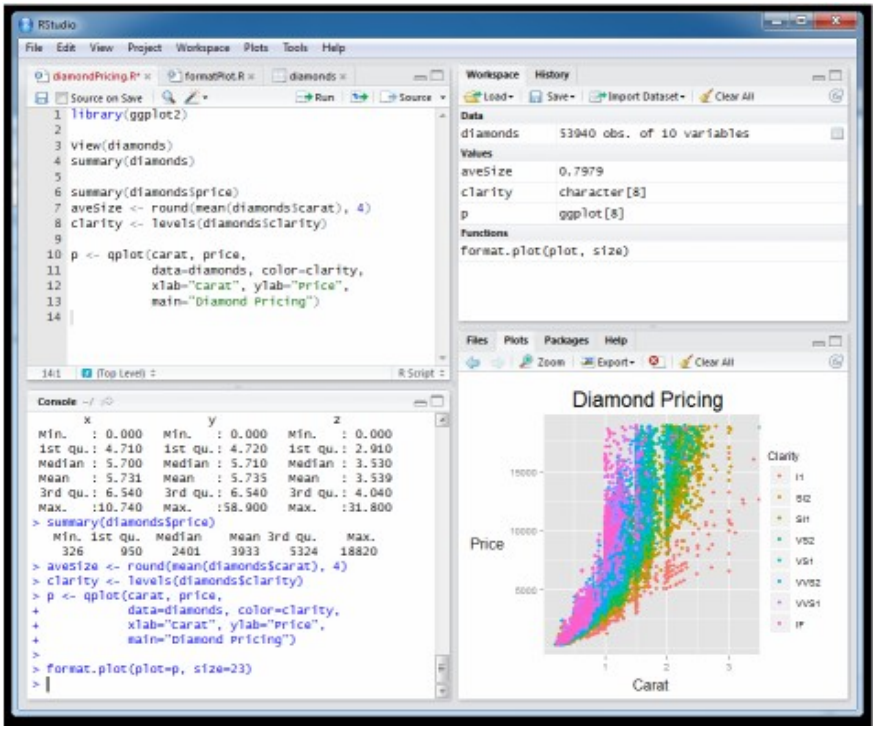


Figura 1.5: RStudio

- Integra la librería.
- Etc.

2. Estructuras de datos

Trabajaremos con ejemplos y fijándonos en los diferentes tipos de datos y de estructuras que R tiene para albergarlos.

Un *dataset*, es (normalmente) un "conjunto de datos" que está ordenado por filas y columnas, y donde cada fila representa una observación y cada columna una variable.

Importamos un fichero de datos desde un *csv* (no nos preocupemos de momento en cómo importar datos).

```
read.table("ejemplo01.csv",sep=';', head=T)
  paciente   admision edad diabetes status
1        1 11/11/2010   23   tipo 1  bueno
2        2 11/01/2009   56   tipo 1  bueno
3        3 15/03/2012   78   tipo 2   malo
4        4 28/04/2008   34   tipo 1  bueno
5        5 09/08/2011   23   tipo 2 mejora
6        6 24/04/2012   12   tipo 1  bueno
7        7 08/08/2008   56   tipo 2  bueno
>
```

Hay diversos tipos de variables, una forma de clasificarlas podría ser esta: datos nominales, ordinales y numéricas.

En el ejemplo anterior:

- Son variables numéricas: `paciente` y `edad`.
- Son variables ordinales: `status`.
- Son variables nominales: `diabetes`.

Nota. Para almacenar los datos R cuenta, como todos los lenguajes de programación, de una gran variedad de estructuras de datos. Estas van de lo más sencillo a las estructuras más complejas.

2.1 Vectores

Son matrices de una dimensión que solamente pueden contener valores numéricos, alfanuméricos y valores lógicos. Emplearemos la función `c()` para formar vectores (función *"combine"*)

```
x <- c(1, 2, 3, 4, 5, 6, 7, 8)
y <- c("juan", "pepe", "iñaky", "amparito", "mariano", "juancar",
"fulano", "elefante")
z <- c(TRUE, TRUE, FALSE, TRUE, FALSE, FALSE, TRUE, TRUE)
```

Podemos acceder un vector usando los corchetes.

```
y[3]
```

```
## [1] "iñaky"
```

```
y[6]
```

```
## [1] "juancar"
```

```
y[8]
```

```
## [1] "elefante"
```

```
x[3]
```

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

```
x[1]
```

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

```
y[c(6, 8)]
```

```
## [1] "juancar"  "elefante"
```

Nota: En R los *índices (numeraciones)* comienzan en el 1, no en el 0 como ocurre en muchos lenguajes de programación.

Extra: El operador "`:`" genera secuencias.

```
c(2:6)
```

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

```
c(1:3)
```

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

Nota: Podemos crear un vector de un cierto tipo y dejarlo vacío para ir llenándolo después en un bucle por ejemplo. Se puede hacer con las instrucciones `v<-vector("numeric")` o `v<-vector(çharacter")`.

2.2 Matrices

Una matriz es un vector con un atributo adicional (dim), que a su vez es un vector numérico de longitud 2 que define el número de filas y columnas. Se crean con la función `matrix()`.

```
matrix(data = NA, nrow = 2, ncol = 2, byrow = F, dimnames = NULL)
```

```
##      [,1] [,2]
## [1,]   NA   NA
## [2,]   NA   NA
```

```
matrix(data = 1:4, nrow = 2, ncol = 2, byrow = F, dimnames = NULL)
```

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

```
matrix(data = 5, nrow = 2, ncol = 2, byrow = F, dimnames = NULL)
```

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

```
matrix(data = 1:6, nrow = 2, ncol = 2, byrow = F, dimnames = NULL)
```

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

```
matrix(data = 1:6, nrow = 2, ncol = 3, byrow = F, dimnames = NULL)
```

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

```
x <- matrix(data = 1:6, nrow = 2, ncol = 3, byrow = F,
    dimnames = NULL)
dim(x)
```

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

Las matrices son muy versátiles, podemos convertir vectores en matrices.

```
x <- 1:15
x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

```
dim(x)
```

```
## NULL
```

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

```
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]    2    7   12
## [3,]    3    8   13
## [4,]    4    9   14
## [5,]    5   10   15
```

```
x <- matrix(1:10, nrow = 2)  # ¡no tenemos que especificar
las columnas!
x
```

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

Los **Arrays** son otro tipo similar a las matrices pero pueden tener más de dos dimensiones. Nos nos detenemos en ellos.

2.3 Dataframes

Un *dataframe* (a veces se traduce como *'marco de datos'*) es una generalización de las matrices donde cada columna puede contener tipos de datos distintos al resto de columnas, manteniendo la misma longitud. Es lo que más se parece a una tabla de datos de SPSS o SAS, o de cualquier paquete estadístico estándar. Se crean con la función `data.frame()`.

```
nombre <- c("juan", "pp", "iñaky", "amparo", "mariano",
    "juancar", "fulano", "elefante")
edad <- c(23, 24, 45, 67, 32, 56, 78, 45)
peso <- c(34, 34, 56, 78, 34, 56, 76, 87)
length(nombre)
```

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

```
length(edad)
```

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

```
length(peso)
```

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

```
caseid <- c(1:length(peso))
df <- data.frame(caseid, nombre, edad, peso)
df
```

```
##   caseid   nombre edad peso
## 1      1     juan   23   34
## 2      2       pp   24   34
## 3      3    iñaky   45   56
## 4      4   amparo   67   78
## 5      5  mariano   32   34
## 6      6  juancar   56   56
## 7      7   fulano   78   76
## 8      8 elefante   45   87
```

Seleccionar columnas concretas de un *dataframe* con los corchetes [ ].

```
df[1:2]
```

```
##   caseid   nombre
## 1      1     juan
## 2      2       pp
## 3      3    iñaky
## 4      4   amparo
## 5      5  mariano
## 6      6  juancar
## 7      7   fulano
## 8      8 elefante
```

```
df[c(1, 3)]
```

```
##   caseid edad
## 1      1   23
## 2      2   24
## 3      3   45
## 4      4   67
## 5      5   32
## 6      6   56
## 7      7   78
## 8      8   45
```

Nota: Prueba con `df ["nombre"]`

Para acceder al vector que forma una de las columnas de un *dataframe* usamos el operador `$`.

```
df$nombre
```

```
## [1] juan     pp       iñaky    amparo   mariano
  juancar  fulano   elefante
## Levels: amparo elefante fulano iñaky juan juancar
  mariano pp
```

```
df$peso
```

```
## [1] 34 34 56 78 34 56 76 87
```

Ampliar un dataframe:

```
diabetes <- c("Tipo1", "Tipo1", "Tipo2", "Tipo2", "Tipo1",
    "Tipo1", "Tipo2", "Tipo1")
estado <- c("bueno", "malo", "bueno", "bueno", "bueno",
    "malo", "bueno", "malo")
length(diabetes)
```

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

```
length(estado)
```

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

```
df <- data.frame(df, diabetes, estado)
df
```

```
##   caseid   nombre edad peso diabetes estado
## 1      1     juan   23   34    Tipo1  bueno
## 2      2       pp   24   34    Tipo1   malo
## 3      3    iñaky   45   56    Tipo2  bueno
## 4      4   amparo   67   78    Tipo2  bueno
## 5      5  mariano   32   34    Tipo1  bueno
## 6      6  juancar   56   56    Tipo1   malo
## 7      7   fulano   78   76    Tipo2  bueno
## 8      8 elefante   45   87    Tipo1   malo
```

Extra: Crear un tabla cruzada.

```
table(df$diabetes, df$estado)
```

```
##
##         bueno malo
##   Tipo1     2    3
##   Tipo2     3    0
```

Funciones `attach()`, `detach()` dataframes para acceder más *fácilmente*.

```
df$peso
```

```
## [1] 34 34 56 78 34 56 76 87
```

```
attach(df)
```

```
## The following object(s) are masked _by_ '.GlobalEnv':
##
##     caseid, diabetes, edad, estado, nombre, peso
```

```
peso
```

```
## [1] 34 34 56 78 34 56 76 87
```

```
detach(df)
```

Nota: si ya existe un objeto con ese nombre puede llevarnos a confusión. Una alternativa al attach es el with().

```
with(df, {+peso})
```

```
## [1] 34 34 56 78 34 56 76 87
```

En nuestro ejemplo estamos empleando la variable `caseid` para identificar las observaciones de forma unívoca. Podemos emplear la opción `rownames` en la función `data.frame()` con tal de usar una variable para cuestiones como etiquetar casos, etc.

```
df <- data.frame(caseid, nombre, edad, peso, diabetes, estado,
    row.names = caseid)
```

2.4 Factores

Las variables pueden ser clasificadas como *nominales, ordinales* o *numéricas*. Las variables nominales son variables categóricas sin un orden definido, como *diabetes* en nuestro ejemplo anterior.

```
df$diabetes
```

```
## [1] Tipo1 Tipo1 Tipo2 Tipo2 Tipo1 Tipo1 Tipo2 Tipo1
## Levels: Tipo1 Tipo2
```

Las variables ordinales, por ejemplo `estatus`, implican orden pero no cantidad.

```
df$estado
```

```
## [1] bueno malo  bueno bueno bueno malo  bueno malo
## Levels: bueno malo
```

En R prestamos un interés especial a estas variables nominales y ordinales y las llamamos *factores*. Los factores son cruciales porque van a determinar cómo se analizarán los datos y cómo se mostrarán en gráficos. Para convertir un vector en un factor empleamos la función `factor()`.

```
estado <- c("bueno", "malo", "bueno", "bueno", "bueno",
    "malo", "bueno", "malo")
diabetes <- c("Tipo1", "Tipo1", "Tipo2", "Tipo2", "Tipo1",
    "Tipo1", "Tipo2", "Tipo1")
str(diabetes)
```

```
##  chr [1:8] "Tipo1" "Tipo1" "Tipo2" "Tipo2" "Tipo1" ...
```

```
str(estado)
```

```
##  chr [1:8] "bueno" "malo" "bueno" "bueno" "bueno" ...
```

```
diabetes <- factor(diabetes)
str(diabetes)
```

```
##  Factor w/ 2 levels "Tipo1","Tipo2": 1 1 2 2 1 1 2 1
```

```
estado <- factor(estado)
str(estado)
```

```
##  Factor w/ 2 levels "bueno","malo": 1 2 1 1 1 2 1 2
```

Nota: La función `str()` nos muestra la estructura de un objeto. Observa que antes de hacer a diabetes un factor era un vector de Chars.

3. Entrada de datos

R es 'compatible' con prácticamente cualquier formato de datos.

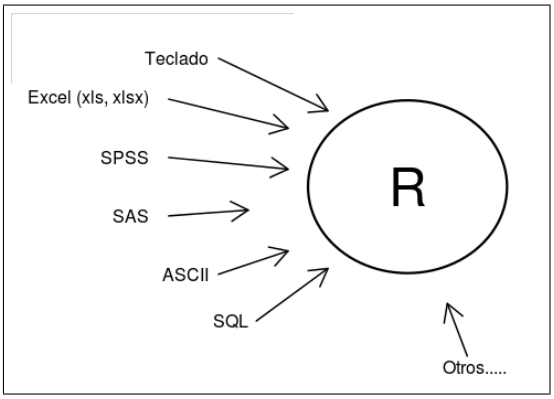


Figura 1.6: Entrada de datos

3.1 Desde el teclado

Si tenemos un *dataframe* y queremos editarlo "a mano" usamos la funcion `edit()`.

```
df<-data.frame(edad=numeric(0),sexo=character(0), peso=numeric(0))
df
[1] edad sexo peso
<0 rows> (or 0-length row.names)
df<-edit(df)
```

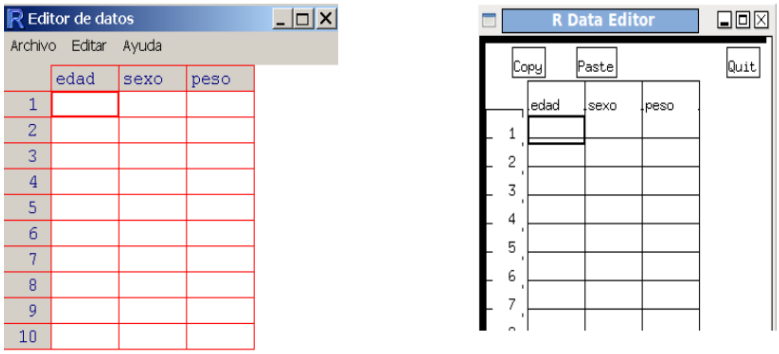


Figura 1.7: Editores visuales de datos: edit(df)

3.2 Importar datos desde ficheros de texto

La función más importante para importar datos, al menos una de las más usadas es `read.table()`, automáticamente te convierte los datos en un *dataframe*.

```
df <- read.table(file, header = logical_value, sep = "delimiter",
    row.names = "name")
```

- `header` será TRUE o FALSE según la primera fila del fichero ASCCI represente el nombre de las variables.
- `sep` es el delimitador, es decir el separador de campos. Por defecto es el espacio " " pero se puede especificar lo que sea: `sep="."`, `sep=","`, `sep="\t"` (tabulación)...
- `row.names` es un parámetro opcional para especificar una o varias variables de identificador de casos.

```
df <- read.table("ejemplo01.csv", sep = ";", head = T)
```

Por defecto las variables de tipo carácter se convierten a factores, si queremos que no lo haga habrá que indicarle `stringsAsFactor = FALSE` como opción.

Hay muchas más opciones, prueba a mirar la ayuda.... `help(read.table)`. Algunas muy importantes tratan el tema de tratar los valores faltantes (`na.strings` = "NA") , o de la separación de decimales para números (`dec` = ".").

3.3 Desde Excel

Lo mejor para leer ficheros *excel* es guardar éstos en formato *csv*, es decir, texto delimitado por comas. También podemos leer el *xls* directamente con el paquete RODBC.

```
> #install.packages("RODBC")
--- Please select a CRAN mirror for use in this session ---
probando la URL 'http://cran.es.r-project.org/bin/windows/
contrib/2.14/RODBC_1.3-5.zip'
Content type 'application/zip' length 753058 bytes (735 Kb)
URL abierta
downloaded 735 Kb

package 'RODBC' successfully unpacked and MD5 sums checked

The downloaded packages are in
        C:\WINDOWS\Temp\RtmpecjIRj\downloaded_packages
> library("RODBC")
Mensajes de aviso perdidos
package 'RODBC' was built under R version 2.14.2
> channel <- odbcConnectExcel ("ejemplo01.xls")
> df3<-sqlFetch(channel, "sheet1")
> df3
  paciente   admision edad diabetes status
1        1 2010-11-11   23   tipo 1  bueno
2        2 2009-01-11   56   tipo 1  bueno
3        3 2012-03-15   78   tipo 2   malo
4        4 2008-04-28   34   tipo 1  bueno
5        5 2011-08-09   23   tipo 2 mejora
6        6 2012-04-24   12   tipo 1  bueno
7        7 2008-08-08   56   tipo 2  bueno
8        8 2099-01-21   88   tipo 1   malo
> odbcClose(channel)
```

Otras funciones que te pueden interesar: `read.xlsx()`

```
library(xlsx)
ficheroexcel <- "c:/ficheroexcel.xlsx"
mydataframe <- read.xlsx(ficheroexcel, 1)# La instrucción
anterior de la primera hoja del fichero ficheroexcel.xlsx
```

3.4 Desde SPSS

Emplearemos la función`spss.get()` del paquete *Hmisc* (se requiere el paquete *foreign*).

```
install.packages("foreign")
install.packages("Hmisc")
library(Hmisc)
df4 <- spss.get("mydata.sav", use.value.labels = TRUE)
```

3.5 Desde SAS

Pues emplear las funciones `read.ssd()` del paquete *foreign* y `sas.get()` del *Hmisc*. Aunque si estos ficheros son posteriores a SAS 9.1 no funcionarán; en ese caso deberás exportar los datos desde *SAS* a *csv* y emplear la función `read.table()` (*que siempre es mejor, más que ir aprendiendo tanta función*).

```
SAS :
proc export data=mydata
outfile="mydata.csv"
dbms=csv;
run;
```

En R:

```
mydata <- read.table("mydata.csv", header = TRUE, sep = ",")
```

3.6 Anotaciones

Al inicio de un análisis no necesitamos muchos detalles más que los vistos hasta ahora, pero para interpretar los resultados y que con el tiempo no dejen de tener sentido conviene 'anotarlos'.

Etiquetas de variables
Para una variable cualquiera, por ejemplo *"edad"*, podríamos querer que tuviera una etiqueta (según la nomenclatura de *spss*), que fuera "Edad el día de la hospitalización (en años)"

```
names(df)[3] <- "Edad el día de la hospitalización (en años)"
```

Etiquetas de valores
Podemos tener un factor codificado, por ejemplo `sexo`, donde 1 es 'Masculino' y 2 'Femenino'. Podemos crear unas etiquetas con el siguiente código.
Primero le añadimos a nuestro ejemplo una variable más (una columna) de unos y doses representando el sexo codificado.

```
sexo <- c(1, 1, 1, 1, 2, 2, 2)
df <- data.frame(df, sexo)
df$sexo <- factor(df$sexo)
```

Prueba a ver la estructura del *dataframe*.

```
str(df)
```

Creamos la etiqueta

```
df$sexo <- factor(df$sexo, levels = c(1, 2), labels =
    c("masculino", "femenino"))
```

Prueba a ver la estructura del *dataframe* con un `str(df)`.

Algunas funciones útiles

Función	Acción
length(obj)	Número de componentes, elementos
dim(obj)	Dimensión de un objeto
str(obj)	Estructura de un objeto
class(obj)	Clase (class) o tipo de objeto
names(obj)	Nombres de los componentes de un objeto
c(obj,obj,...)	Combina objetos en un vector
head(obj)	Lista la primera parte de un objeto
tail(obj)	Lista la última parte (cola) de un objeto
ls()	Lista los objetos actuales

rm(obj)	Borra un objeto
newobj<-edit(obj)	Edita un objeto y lo guarda
fix(obj)	Edita sobre un objeto ya creado

4. Manipulación de datos básica

Trabajaremos desde ahora con la GUI RStudio, veremos que todo es más sencillo y procuraremos coger un poco de práctica en trabajo sobre scripts, que es como la mayor parte de los especialistas trabajan. Pronto veremos que es lo más cómodo.

Vamos a crear un *dataset* y trabajaremos con él a lo largo del capítulo.
Para que sea asequible vamos a crear sólo 3 observaciones. Esta es una modificación del *ejemplo02.csv* que se puede descargar del material del curso.

Figura 1.8: Editor de scripts de RStudio

```
# getwd()
manager <- c(1:5)
date <- c("10/11/08", "10/12/08", "10/13/08", "10/14/08",
    "10/15/08")
country <- c("US", "US", "UK", "UK", "UK")
gender <- c("M", "F", "F", NA, "F")
age <- c(NA, 45, 25, 39, 99)
q1 <- c(5, 3, 3, 3, 2)
q2 <- c(5, 5, 5, NA, 2)
q3 <- c(5, 5, 2, NA, 1)
df <- data.frame(manager, date, country, gender, age, q1, q2,
    q3, stringsAsFactors = FALSE)
df
```

```
##   manager     date country gender age q1 q2 q3
## 1       1 10/11/08      US      M  NA  5  5  5
## 2       2 10/12/08      US      F  45  3  5  5
## 3       3 10/13/08      UK      F  25  3  5  2
## 4       4 10/14/08      UK   <NA>  39  3 NA NA
## 5       5 10/15/08      UK      F  99  2  2  1
```

	manager	date	country	gender	age	q1	q2	q3
1	1	10/11/08	US	M	NA	5	5	5
2	2	10/12/08	US	F	45	3	5	5
3	3	10/13/08	UK	F	25	3	5	2
4	4	10/14/08	UK	NA	39	3	NA	NA
5	5	10/15/08	UK	F	99	2	2	1

Figura 1.9: df

4.1 Operadores aritméticos

Si queremos crear una nueva variable usando operadores aritméticos es tan sencillo como saberse los operadores.

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
/	División
^ ó **	Potencias
x % %y	Módulo (x mod y) 5 % %2 es 1
x %/ %y	División entera 5 %/ %2 is 2

Así si queremos crear una variable q1mas2 que sea q1+q2

```
q1masq2 <- q1 + q2
```

Un método interesante para crear nuevas variables y que automáticamente se incluyan en el *dataframe* es el empleo de la función `transform()`.

```
# creamos variables con la función transform

df <- transform(df, sumx = q1 + q2, meanx = (q1 + q2)/2)

df[, 4:10] # se muestran las 6 últimas columnas de df
         # más información en la sección 4.8
```

```
##    gender age q1 q2 q3 sumx meanx
## 1       M  NA  5  5  5   10     5
## 2       F  45  3  5  5    8     4
## 3       F  25  3  5  2    8     4
## 4    <NA>  39  3 NA NA   NA    NA
## 5       F  99  2  2  1    4     2
```

4.2 Recodificación de variables

Para recodificar variables es interesante introducir los operadores lógicos, pues a menudo recodificamos según una regla lógica.

Operador	Descripción
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que
==	Exactamente igual a
!=	No igual a/que
!x	Diferente de x
x | y	x o y
x & y	x e y
isTRUE(x)	Prueba si x es TRUE

```
# Creamos la variable agecat (categoría de edad), le asignamos
# la palabra 'anciano' si age>75, si está entre 44 y 77 le
# asignamos 'maduro' y siestá por debajo de 44 'joven'.

df$agecat[df$age > 75] <- "anciano"
df$agecat[df$age <= 75 & df$age > 44] <- "maduro"
df$agecat[df$age <= 44] <- "joven"

df[, 3:11]
```

```
##   country gender age q1 q2 q3 sumx meanx  agecat
## 1      US      M  NA  5  5  5   10     5    <NA>
## 2      US      F  45  3  5  5    8     4  maduro
## 3      UK      F  25  3  5  2    8     4   joven
## 4      UK   <NA>  39  3 NA NA   NA    NA   joven
## 5      UK      F  99  2  2  1    4     2 anciano
```

La sentencia `variable[condicion] <- expresión` sólo hará la asignación cuando la condición sea TRUE.

4.3 Renombrar variables

Para renombrar variables lo más burdo sería hacer un `fix(df)` e *"ir a mano"*. En contra se suele emplear el comando `rename`, en RStudio el comando `fix` tampoco nos va a funcionar por la naturaleza misma de la GUI.

```
# renombrar variables install.packages('reshape')
library(reshape)
df <- rename(df, c(manager = "manID", date = "testdate"))

# Otra opción es con la función names(), sin necesidad de
# más paquetes que los básicos.
names(df)
```

```
##  [1] "manID"    "testdate" "country"  "gender"
"age"      "q1"
##  [7] "q2"       "q3"       "sumx"     "meanx"
"agecat"
```

```
names(df)[3] <- "pais"
```

```
# del mismo modo
names(df)[6:8] <- c("it1", "it2", "it3")
df[c(-9, -10)]# no se muestran las columnas 9 y 10
```

```
##   manID testdate pais gender age it1 it2 it3  agecat
## 1     1 10/11/08   US      M  NA   5   5   5    <NA>
## 2     2 10/12/08   US      F  45   3   5   5  maduro
## 3     3 10/13/08   UK      F  25   3   5   2   joven
## 4     4 10/14/08   UK   <NA>  39   3  NA  NA   joven
## 5     5 10/15/08   UK      F  99   2   2   1 anciano
```

Notar que hemos comentado las instrucciones `install.packages('reshape')` y `library(reshape)` pero puede que en tu caso sean necesarias si es la primera vez que las ejecutas.

4.4 Valores faltantes (missing values)

El tratamiento de datos faltantes, más conocidos como *missing* es fundamental para la correcta interpretación de un análisis.
- En R valores imposibles, (*por ejemplo los resultantes de dividir por 0*), se representan por el código: `NaN` (*Not a Number*).
- Los valores faltantes (*simplemente no sabemos qué valor toma*) por el código: `NA` (*Not available*).

Hay muchas funciones para identificar estos valores, la más usada es `is.na()`.

```
# missings
y <- c(1, 2, 3, NA)
is.na(y)  # ¿cuáles son NA?,
```

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

```
# Ojo: is.na() devuelve un objeto del mismo tamaño que
  el que recibe.
is.na(df[, 4:10])
```

```
##      gender   age   it1   it2   it3  sumx meanx
## [1,]  FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
## [2,]  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [3,]  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [4,]   TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
## [5,]  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

Recodificar valores a missings Para recodificar valores faltantes podemos empelar las mismas técnicas de recodificación que ya hemos visto.

```
# recodificar
df$age[is.na(df$age)] <- 99
# observa qué ocurre con 'is.na(df)'
is.na(df[, 4:10])
```

```
##       gender   age   it1   it2   it3  sumx meanx
## [1,]  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [2,]  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [3,]  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [4,]   TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
## [5,]  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
# ahora devolvemos el df a su estado original.
df$age[df$age == 99] <- NA
is.na(df[, 4:10])
```

```
##       gender   age   it1   it2   it3  sumx meanx
## [1,]  FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
## [2,]  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [3,]  FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [4,]   TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
## [5,]  FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
```

Excluir valores missing del análisis Emplearemos la opción `na.rm=TRUE` para que **no** considere los valores faltantes.

```
# excluir los missings del análisis
x <- c(1, 2, NA, 3)
y <- x[1] + x[2] + x[3] + x[4]
z <- sum(x)
y
```

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

```
z
```

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

```
# ¡¡¡¡Ambos son NA!!!!
sum(x, na.rm = T)  # ¡Ahora no!
```

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

De una forma más general podemos emplear la función `na.omit()` que elimina cualquier fila que tenga valores faltantes.

```
df <- na.omit(df)
df[, 4:11]
```

```
##   gender age it1 it2 it3 sumx meanx agecat
## 2      F  45   3   5   5    8     4 maduro
## 3      F  25   3   5   2    8     4  joven
```

4.5 Conversión de tipos

Muchas veces necesitamos que R entienda un número como un carácter, o viceversa. En R cuando añadimos a un vector numérico un elemento no numérico convierte todo el vector en no numérico automáticamente.

Lógico	Convierte
is.numeric()	as.numeric()
is.character()	as.character()
is.vector()	as.vector()
is.matrix()	as.matrix()
is.data.frame()	as.data.frame()

```
# conversión de tipos.
a <- c(1, 2, 3)
a
```

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

```
is.numeric(a)
```

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

```
is.vector(a)
```

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

```
a <- as.character(a)
a
```

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

```
is.numeric(a)
```

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

```
is.vector(a)
```

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

4.6 Ordenar datos

Para ordenar las filas de un datafarme empelamos la función `order()`, por defecto ordena ascendentemente. Empleamos el signo *'menos'* para cambiar el sentido.

```
df[c(-10, -11)]# no se muestran las columnas 9 y 10
```

```
##   manID testdate pais gender age it1 it2 it3 sumx
## 2     2 10/12/08   US      F  45   3   5   5    8
## 3     3 10/13/08   UK      F  25   3   5   2    8
```

```
df_ordenado <- df[order(df$age), ]
df_ordenado[c(-9, -10)]# no se muestran las columnas 9 y 10
```

```
##   manID testdate pais gender age it1 it2 it3 sumx
## 3     3 10/13/08   UK      F  25   3   5   2    8
## 2     2 10/12/08   US      F  45   3   5   5    8
```

```
df_ordenado2 <- df[order(df$gender, -df$age), ]
df_ordenado2[c(-9, -10)]# no se muestran las columnas 9 y 10
```

```
##   manID testdate pais gender age it1 it2 it3 sumx
## 2     2 10/12/08   US      F  45   3   5   5    8
## 3     3 10/13/08   UK      F  25   3   5   2    8
```

4.7 Ampliar/unir conjuntos de datos

Podemos ampliar un *dataframe* añadiendo variables (columnas) o casos (filas). Y añadir columnas con las funciones `merge()` y `cbind()`.

```
dfA <- df  # nos creamos un par de dataframes aunque sean iguales
dfB <- df
df.total <- merge(dfA, dfB, by = "manID")

df.total
```

```
##   manID testdate.x pais.x gender.x age.x it1.x it2.x
## 1     2   10/12/08     US        F    45     3     5
## 2     3   10/13/08     UK        F    25     3     5
##    it3.x sumx.x meanx.x agecat.x testdate.y pais.y
## 1      5      8       4    maduro   10/12/08     US
## 2      2      8       4     joven   10/13/08     UK
##   gender.y age.y it1.y it2.y it3.y sumx.y meanx.y
## 1        F    45     3     5     5      8       4
## 2        F    25     3     5     2      8       4
##   agecat.y
## 1   maduro
## 2    joven
```

```
# unir matrices
A <- matrix(1:9, 3, 3)
B <- matrix(1:9, 3, 3)
AB.total <- cbind(A, B)
AB.total
```

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

```
# unir matrices
lA <- matrix(c("a", "b", "c", "d"), 2, 2)
lB <- matrix(c("A", "B", "C", "D"), 2, 2)
lAB.total <- cbind(A, B)
lAB.total
```

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

Para unir filas (observaciones) empleamos la función `rbind(dfA, dfB)`. **¡Ojo!** Ambos *dataframes* han de tener las mismas columnas.

```
df.total2 <- rbind(dfA, dfB)
df.total2
df_total2[c(-9, -10)]# no se muestran las columnas 9 y 10
```

```
##    manID testdate pais gender age it1 it2 it3 sumx
## 2      2 10/12/08   US      F  45   3   5   5    8
## 3      3 10/13/08   UK      F  25   3   5   2    8
## 21     2 10/12/08   US      F  45   3   5   5    8
## 31     3 10/13/08   UK      F  25   3   5   2    8
```

4.8 Seleccionar subconjuntos de un dataframe

Muy a menudo necesitamos efectuar operaciones sobre una parte de un conjunto de datos, ya sea de un subconjunto de variables o de un subconjunto de observaciones. R es muy ágil en estas operaciones y hay muchas formas de acceder a observaciones o variables concretas.

Usaremos los elementos de un *dataframe*, los corchetes para acceder a partes de él.

`dataframe[índice de fila, índice de columna]`

Ojo: la regla es *"filas por columnas"*.

Seleccionar variables (columnas...) `nuevo_df<- df[ , índices]`

Observar que dejamos en blanco la primera posición entre los corchetes para referirnos a la columnas

```
# Seleccionar columnas de un dataframe
df <- data.frame(manager, date, country, gender, age, q1, q2,
q3, stringsAsFactors = FALSE)
df
```

```
##   manager     date country gender age q1 q2 q3
## 1       1 10/11/08      US      M  NA  5  5  5
## 2       2 10/12/08      US      F  45  3  5  5
## 3       3 10/13/08      UK      F  25  3  5  2
## 4       4 10/14/08      UK   <NA>  39  3 NA NA
## 5       5 10/15/08      UK      F  99  2  2  1
```

```
new_df <- df[, c(6:8)]
new_df
```

```
##   q1 q2 q3
## 1  5  5  5
## 2  3  5  5
## 3  3  5  2
## 4  3 NA NA
## 5  2  2  1
```

```
new_df2 <- df[, "country"]
new_df2  # por el nombre de la 'columna'
```

```
## [1] "US" "US" "UK" "UK" "UK"
```

```
new_df3 <- df[, 3]
new_df3  # por el índice
```

```
## [1] "US" "US" "UK" "UK" "UK"
```

```
# aunque más generalmente si especificamos el nombre no hace
# falta tratar con los índices
new_df4 <- df["country"]
new_df4  # por el nombre de la 'columna'
```

```
##   country
## 1      US
## 2      US
## 3      UK
## 4      UK
## 5      UK
```

```
str(new_df4)
```

```
## 'data.frame':    5 obs. of  1 variable:
##  $ country: chr  "US" "US" "UK" "UK" ...
```

```
str(new_df3)
```

```
##  chr [1:5] "US" "US" "UK" "UK" "UK"
```

Igualmente podríamos haber construido un vector de chars:
`variables<-c("country","date"); new_df<-df[,variables].`

Seleccionar para eliminar: Atención al signo '-' delante del índice.

```
# eliminamos 'q3' y 'q4' que tienen índices '8' y '9'
new_df5 <- df[c(-8, -9)]
new_df5
```

```
##   manager     date country gender age q1 q2
## 1       1 10/11/08      US      M  NA  5  5
## 2       2 10/12/08      US      F  45  3  5
## 3       3 10/13/08      UK      F  25  3  5
## 4       4 10/14/08      UK   <NA>  39  3 NA
## 5       5 10/15/08      UK      F  99  2  2
```

Seleccionar observaciones (filas...) Haremos uso de lo mismo que para las columnas pero dejando vacía la segunda posición del corchete y jugando con los operadores lógicos.

nuevo_df<- df[índices,]

```
# Seleccionar observaciones
df
```

```
##   manager     date country gender age q1 q2 q3
## 1       1 10/11/08      US      M  NA  5  5  5
## 2       2 10/12/08      US      F  45  3  5  5
## 3       3 10/13/08      UK      F  25  3  5  2
## 4       4 10/14/08      UK   <NA>  39  3 NA NA
## 5       5 10/15/08      UK      F  99  2  2  1
```

```
# seleccionar las 3 primeras filas
df6 <- df[1:3, ]
df6
```

```
##   manager     date country gender age q1 q2 q3
## 1       1 10/11/08      US      M  NA  5  5  5
## 2       2 10/12/08      US      F  45  3  5  5
## 3       3 10/13/08      UK      F  25  3  5  2
```

```
df7 <- df[which(df$gender == "F" & df$country == "UK"), ]
df7
```

```
##   manager     date country gender age q1 q2 q3
## 3       3 10/13/08      UK      F  25  3  5  2
## 5       5 10/15/08      UK      F  99  2  2  1
```

5. Funciones

No nos vamos a parar en muchos detalles de las funciones, sólo decir que hay miles y miles, agrupadas en paquetes, y cada día hay más paquetes que aporta la comunidad y que se encuentran en CRAN como ya mencionamos al principio del texto.

5.1 Funciones matemáticas

Para empezar sí debemos de saber que existen un buen número de funciones matemáticas estándar, su sintaxis es muy parecida a otros lenguajes de programación.
Algunas son:

Función	Lo que hace
abs(x)	Valor absoluto de x, abs(-4) devuelve 4
sqrt(x)	Raíz cuadrada de x, sqrt(25) devuelve 5
ceiling(x)	Entero más pequeño mayor que x, ceiling(4.6) devuelve 5
floor(x)	Entero más grande no mayor que x, floor(4.6) devuelve 4
trunc(x)	Truncamiento de x
round(x, digits=n)	Redondea x a un número específico de decimales
log(x,base=n)	Logaritmos (log() es el logaritmo natural, log(exp(1)) devuelve 1

5.2 Funciones estadísticas

Función	Lo que hace
mean(x)	Media
median(x)	Mediana
sd(x)	Desviación estándar
var(x)	Varianza
sum(x)	Suma
range(x)	Rango

min(x)	Mínimo
max(x)	Máximo
scale(x,center=TRUE,scale=TRUE)	Estandarizar

Ejercicio: Probar estas funciones con `x<-c(1,2,3,4,5,6,7,8,9)`

Existen también funciones de densidad de probabilidad, etc. que veremos más adelante.

Las funciones de caracteres (`strings`, `chars`), que buscan, sustituyen, cuentan caracteres en cadenas, etc. nos las vamos a saltar también porque no son *"importantes"* para nuestros objetivos.

5.3 Miscelánea de funciones interesantes

```
# Miscelánea de funciones interesantes medir objetos
x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
length(x)
```

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

```
# generar secuencias con saltos
seq(1, 10, 2)  # seq(from, to, by)
```

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

```
seq(1, 20, 3)
```

```
## [1]  1  4  7 10 13 16 19
```

```
# repetir
rep(1, 5)  # rep(ob, número de repet)
```

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

```
rep(c("A", "B"), 5)
```

```
##  [1] "A" "B" "A" "B" "A" "B" "A" "B" "A" "B"
```

```
rep(1:3, 2)
```

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

```
# concatenar
cat("hola", "amigo")
```

```
## hola amigo
```

```
cat("hola", "amigo", "\n", "¿cómo estás?", file =
"ficheto-cocatenado.txt")# escribimos en un fichero
```

Una funcionalidad que resulta muy útil al trabajar con R es que **podemos aplicar funciones a una gran cantidad de objetos**: vectores, arrays, matrices, dataframes...

```
# aplicar funciones a objetos 'complejos'
y <- c(1.23, 4.56, 7.89)
round(y)
```

```
## [1] 1 5 8
```

```
z <- matrix(runif(12), 3)
z
```

```
##         [,1]   [,2]   [,3]   [,4]
## [1,] 0.6840 0.8269 0.2239 0.7958
## [2,] 0.6897 0.9827 0.4000 0.6045
## [3,] 0.3431 0.6809 0.1155 0.3928
```

```
log(z)
```

```
##          [,1]      [,2]     [,3]     [,4]
## [1,] -0.3798 -0.19009 -1.4967 -0.2285
## [2,] -0.3716 -0.01749 -0.9162 -0.5034
## [3,] -1.0698 -0.38431 -2.1586 -0.9345
```

```
mean(z)  # devuelve un escalar
```

```
## [1] 0.5616
```

5.4 Función apply()

En el último ejemplo `mean(z)` devuelve un escalar cuando le habíamos proporcionado una `matriz`, ¿qué hacemos si queremos las medias por filas?.
Para eso en R podemos emplear la función `apply()`. Si quieres saber más prueba `?apply`.

```
apply(z, 1, mean)  # medias de las filas
```

```
## [1] 0.6326 0.6692 0.3831
```

```
apply(z, 2, mean)  # medias de las columnas
```

```
## [1] 0.5722 0.8302 0.2465 0.5977
```

`apply(x, MARGIN, FUN)` donde FUN: cualquier función, y MAGIN: índice de la dimensión (1=fila, 2=columna) a la que se la quieres aplicar.

Nota: `apply()` es una potentísima herramienta.

5.5 Control de flujo

Sobre control de flujo no vamos a tratar nada en este curso, sólo mencionaremos que como en cualquier lenguaje de programación existe y es fundamental. Lo mencionamos para quien quiera explorarlo y sacarle partido.

Sentencias FOR Ejecutará una sentencia hasta que la variable no esté contenida en la secuencia

`for (var in seq) sentencia`

```
# FOR
x <- 0
x
```

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

```
for (i in 1:3) {
    x <- x + 1
}
x
```

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

```
for (i in 1:2) {
    print("¡¡Viva el software libre!!")
}
```

```
## [1] "¡¡Viva el software libre!!"
## [1] "¡¡Viva el software libre!!"
```

Sentencias WHILE Un bucle while ejecuta una sentencia mientras se cumpla una condición.

`while (condición) sentencia`

```
i <- 5
while (i > 0) {
    print("¡Perfecto mundo!")
    i <- i - 1
}
```

```
## [1] "¡Perfecto mundo!"
## [1] "¡Perfecto mundo!"
## [1] "¡Perfecto mundo!"
## [1] "¡Perfecto mundo!"
## [1] "¡Perfecto mundo!"
```

Sentencias IF-ELSE Ejecutan sentencias si se da una condición `if`, si no se da ejecutan la sentencia `else`. Se pueden anidar

```
if (condición) sentencia
ó
if (condición) sentencia1 else sentencia2
```

```
# IF-ELSE
if (TRUE) {
    print("esta siempre se ejecuta")
}

if (FALSE) {
    print("esta nunca se ejecuta")
} else {
    print("¿lo ves?")
}
```

5.6 Funciones escritas por el usuario

Una de las mayores potencialidades de R es la posibilidad de escribir funciones 'ad hoc'. Si además consideramos que estas se pueden empaquetar en grupos y tener siempre disponibles para todas las sesiones nos podemos hacer una idea de lo potente y personalizable que puede llegar a ser R.

La sintaxis es sencilla:

```
mi_funcion <- function (arg1, arg2,...){
sentencias
return (objeto)
}
```

```
f.potencia <- function(num, exp = 2) {
    return(num * exp)
}
f.potencia(5, 2)
```

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

```
f.potencia(5, 5)
```

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

```
f.potencia(5)
```

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

Ejercicio: Crear una función que acepte dos argumentos, uno que sea un char con dos posibilidades "param" o "noparam", que por defecto sea el valor "param". El segundo argumento que sea un vactor numérico. Y que nos devuelva para "param", la media, la desviación típica y la varianza del vector. Y para "noparam", la medina, los cuartiles 0.25 y 0.75, máximo y mínimo del vector.

Ejercicio: Crear una función que resuelva la ecuación de segundo grado. Y probarla para la ecuación x^2 + 5x - 16 = 0

```
# posible solución
e2grado <- function(a, b, c) {
    discriminante <- (b^2) - (4 * a * c)
    solucion1 <- (-b + sqrt(discriminante))/(2 * a)
    solucion2 <- (-b - sqrt(discriminante))/(2 * a)
    c(solucion1, solucion2)
}

e2grado(1, 5, -16)
```

6. Estadística descriptiva con R

6. 1. Algunas definiciones por encima:

Vamos ahora repasar conceptos estadísticos importantes de una manera intuitiva que nos van a servir para encarar más adelante el concepto de contraste estadístico y *p-valor* más eficientemente.

6.1.1 Población y muestra. Variables

Población: Llamamos *población estadística*, *universo* o *colectivo* al conjunto de referencia sobre el cuál van a recaer las observaciones.

Individuos: Se llama *unidad estadística* o *individuo* a cada uno de los elementos que componen la población estadística. El individuo es un ente observable que no tiene por qué ser una persona, puede ser un objeto, un ser vivo, o incluso algo abstracto.

Muestra: Es un subconjunto de elementos de la población. Se suelen tomar muestras cuando es difícil o costosa la observación de todos los elementos de la población estadística.

Censo: Decimos que realizamos un censo cuando se observan todos los elementos de la población estadística.

Caracteres: La observación del individuo la describimos mediante uno o más caracteres. El carácter es, por tanto, una cualidad o propiedad inherente en el individuo. Tipos de caracteres:

- Cualitativos: aquellos que son categóricos, pero no son numéricos. Por ejemplo: color de los ojos, profesión, marca de coche...
- Ordinales: aquellos que pueden ordenarse pero no son numéricos. Por ejemplo: preguntas de encuesta sobre el grado de satisfacción de algo (mucho, poco, nada; bueno, regular, malo).
- Cuantitativos: son numéricos. Por ejemplo: peso, talla, número de hijos, número de libros leídos al mes...

Modalidad o Valor: Un carácter puede mostrar distintas modalidades o valores, es decir, son distintas manifestaciones o situaciones posibles que puede presentar un carácter estadístico. Las modalidades o valores son incompatibles y exhaustivos.
Generalmente se utiliza el término modalidad cuando hablamos de caracteres cualitativos y el término valor cuando estudiamos caracteres cuantitativos.
Por ejemplo: el carácter cualitativo “Estado Civil”, puede adoptar las modalidades: casado, soltero, viudo. El carácter cuantitativo “Edad” puede tomar los valores: diez, once, doce, quince años...

Variable Estadística: Al conjunto de los distintos valores numéricos que adopta un carácter cuantitativo se llama variable estadística.

Tipos de variables estadísticas:

- Discretas: Aquellas que toman valores aislados (números naturales), y que no pueden tomar ningún valor intermedio entre dos consecutivos fijados. Por

ejemplo: número de goles marcados, número de hijos, número de discos comprados, número de pulsaciones...

- Continuas: Aquellas que toman infinitos valores (números reales) en un intervalo dado, de forma que pueden tomar cualquier valor intermedio, al menos teóricamente, en su rango de variación. Por ejemplo: talla, peso, presión sanguínea, temperatura...

Observación: Una observación es el conjunto de modalidades o valores de cada variable estadística medidos en un mismo individuo.
Por ejemplo, en una población de 100 individuos podemos estudiar, de forma individual, tres caracteres: edad (18, 19...), sexo (hombre, mujer) y 'ha votado en las elecciones'(sí, no). Realizamos 100 observaciones con tres datos cada una, es decir, una de las observaciones podría ser (43, H, S).

6.1.2 Distribución de probabilidad y Función de densidad de una v.a.

Intuitivamente, una *variable aleatoria* puede tomarse como una cantidad cuyo valor no es fijo pero puede tomar diferentes valores; una *distribución de probabilidad* se usa para describir la probabilidad de que se den los diferentes valores (se denota usualmente por F(x)).

$$F_x(x) = P(X \leq x)$$

La distribución de probabilidad de una v.a. describe teóricamente la forma en que varían los resultados de un experimento aleatorio. Intuitivamente se trataría de una lista de los resultados posibles de un experimento con las probabilidades que se esperarían ver asociadas con cada resultado.

La *función de densidad de probabilidad*, ***función de densidad***, o, simplemente, ***densidad de una variable aleatoria continua***es una función, usualmente denominada f(x) que describe la densidad de la probabilidad en cada punto del espacio de tal manera que la probabilidad de que la variable aleatoria tome un valor dentro de un determinado conjunto sea la integral de la función de densidad sobre dicho conjunto.

$$F(x) = \int_{-\infty}^{x} f(t)\, dt$$

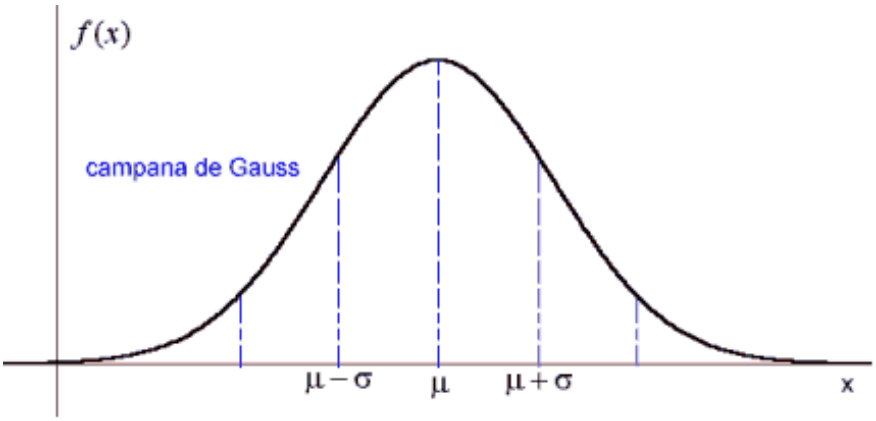


Figura 1.10: Función de densidad. Campana de Gauss

6.1.3 Parámetros y estadísticos

Parámetro: Es una cantidad numérica calculada sobre una población.

- La altura media de los individuos de un país.
- La idea es resumir toda la información que hay en la población en unos pocos números (parámetros).

Estadístico: Es una cantidad numérica calculada sobre una muestra.

- La altura media de los que estamos en este aula.
- ¿Somos una muestra (representativa) de la población?

Si un estadístico se usa para aproximar un parámetro también se le suele llamar **estimador**.

6.1.4 Tipos de estadísticos

Los estadísticos se calculan, y estos estiman parámetros.

Hay diferentes tipos según las *cosas* que queramos saber de la distribución de una variable.

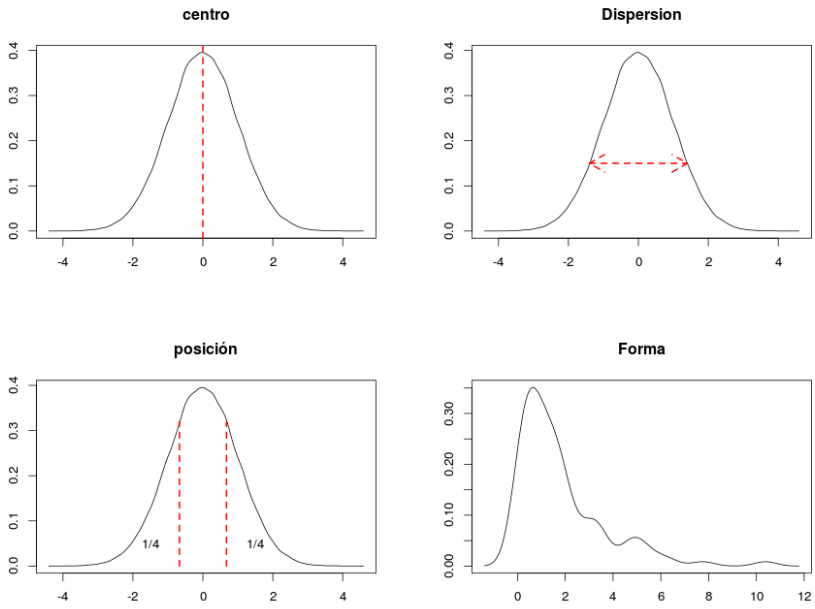


Figura 1.11: Tipos de estadísticos

Medidas de posición. Dividen un conjunto ordenado de datos en grupos con la misma cantidad de individuos. Las más populares: Cuantiles, percentiles, cuartiles, deciles...

- Se define el cuantil de orden α como un valor de la variable por debajo del cual se encuentra una frecuencia acumulada α.
- El percentil de orden k es el cuantil de orden $\frac{k}{100}$.

Ejemplo: El 5 % de los recién nacidos tiene un peso demasiado bajo. ¿Qué peso se considera "demasiado bajo"? Percentil 5 o cuantil 0,05.

```
pesos <- rnorm(1000, 3, 0.8)
hist(pesos)
```

```
quantile(pesos, 0.05)
```

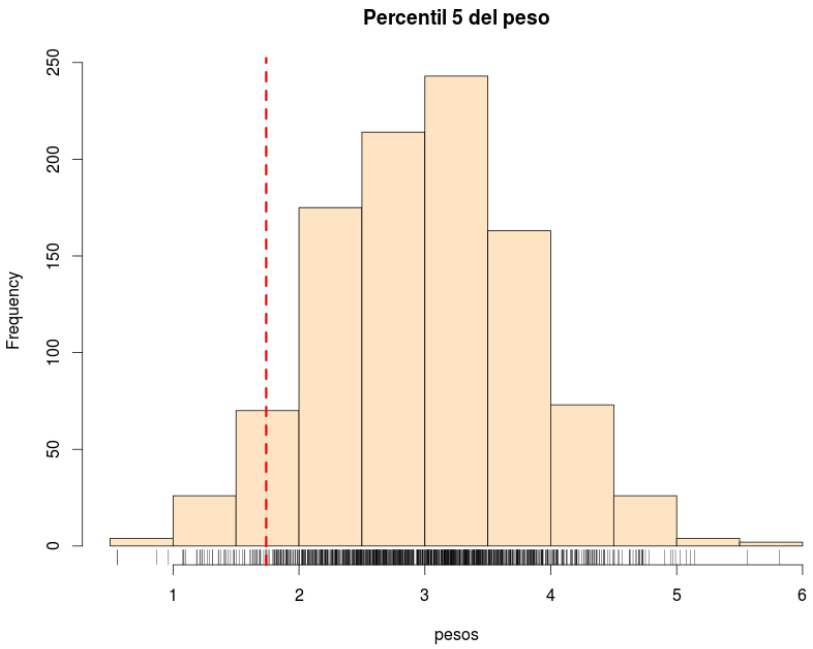


Figura 1.11: Histograma con percentil

```
##     5%
## 1.677
```

Ejemplo: ¿Qué peso es superado sólo por el 25 % de los individuos? Percentil 75, tercer cuartil o cuantil 0.75.

```
quantile(pesos, 0.75)
```

```
##    75%
## 3.526
```

Medidas de centralización o tendencia central: Indican valores con respecto a los que los datos "parecen" agruparse.

- Media, mediana, moda...

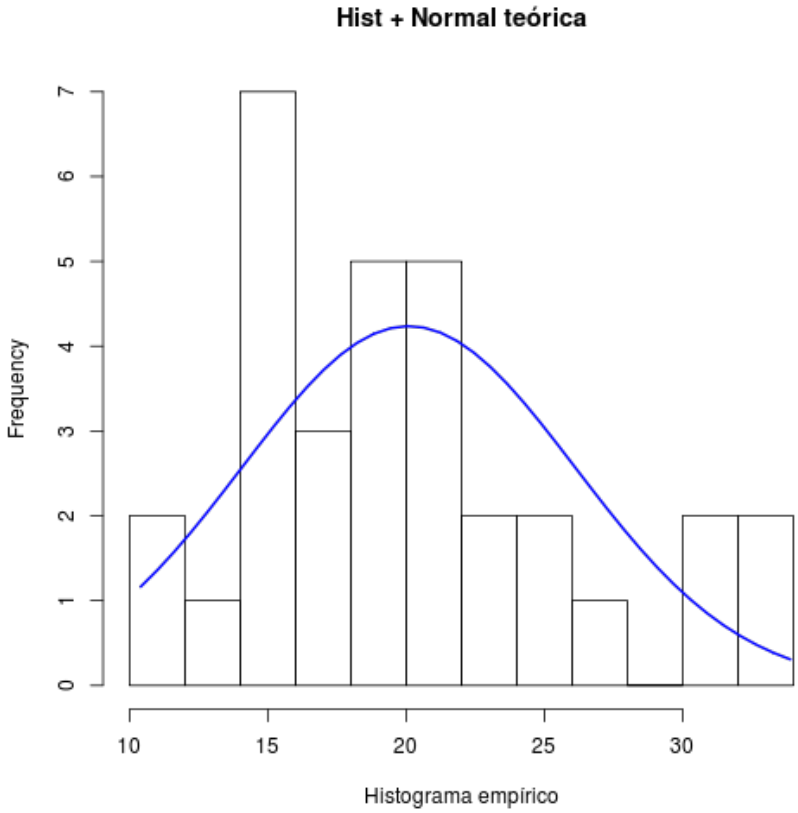


Figura 1.12: Histograma y curva de densidad de la normal teórica

Media("mean"): Es la media aritmética (promedio) de los valores de una variable. Suma de los valores dividido por el tamaño muestral.

```
mean(pesos)
```

```
## [1] 2.994
```

```
x <- c(1, 2, 3, 4, 5)
mean(x)
```

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

```
media <- (1 + 2 + 3 + 4 + 5)/5
media
```

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

- Conveniente cuando los datos se concentran simétricamente con respecto a ese valor. Muy sensible a valores extremos (en estos casos hay otras 'medias', menos intuitivas, pero que pueden ser útiles: media aritmética, geométrica, ponderada...)
- Centro de gravedad de los datos.

Mediana("median"): Es un valor que divide a las observaciones en dos grupos con el mismo número de individuos (percentil 50). Si el número de datos es par, se elige la media de los dos datos centrales.

- Mediana de 1,2,4,**5**,6,6,8 es 5.
- Mediana de 1,2,4,**5,6**,6,8,9 es $\frac{5+6}{2} = 5{,}5$
- Es conveniente cuando los datos son asimétricos. No es sensible a valores extremos.
- Mediana de 1,2,4,5,6,6,800 es 5. (¡La media es 117,7!)

```
median(pesos)
```

```
## [1] 3.021
```

```
x <- c(1, 2, 4, 5, 6, 6, 8)
median(x)
```

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

```
y <- c(1, 2, 4, 5, 6, 6, 8, 9)
z <- c(1, 2, 4, 5, 6, 6, 800)
median(z)
```

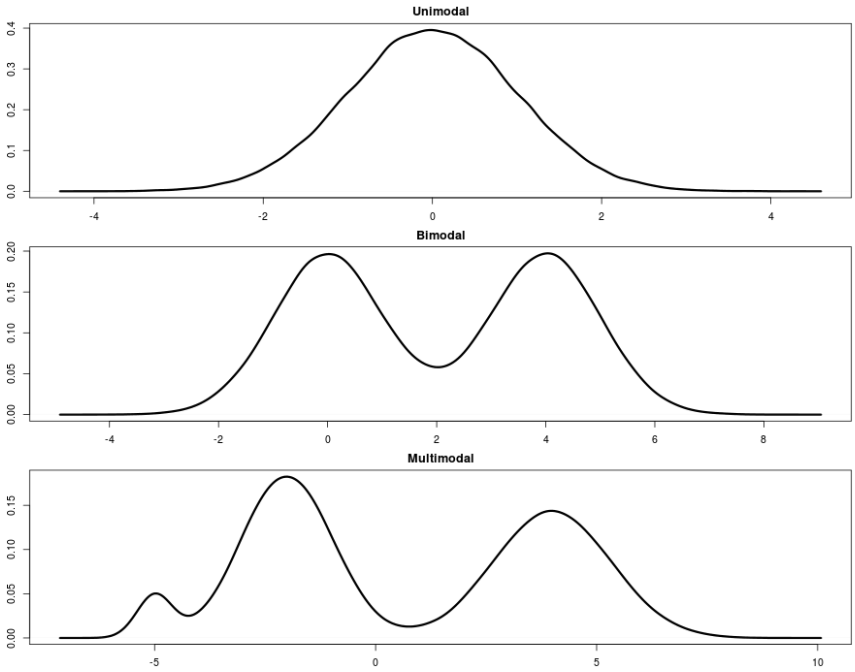


Figura 1.13: Moda

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

Moda:Es el/los valor/es donde la distribución de frecuencia alcanza un máximo. Puede haber más de uno (2 bimodal...)

```
# paquete "PrettyR" o crear una función install.packages('prettyR',
# dependencies=T)
library("prettyR")
help(package = "prettyR")
# Otra forma, crearnos una función x discreta
modad <- function(x) as.numeric(names(which.max(table(x))))
# ¡Ojo!: no funciona si hay más de una moda o si es constante

modad(c(1, 1, 3, 4, 5, 6, 6, 6))
modac <- function(x) {
    dd <- density(x)
    dd$x[which.max(dd$y)]
}
modad(z)
```

Nota: Un inciso sobre el cálculo de medias

- Datos sin agrupar: $x_1, x_2, \ldots x_n : \overline{x} = \frac{\sum_i x_i}{n}$
- Datos organizados en tabla: Si está en intervalos usar como x_i las marcas de clase. Si no ignorar la columna de intervalos: $\overline{x} = \frac{\sum_i x_i n_i}{n}$

Variable		fr.	fr. ac.
$L_0 - L_1$	x_1	n_1	N_1
$L_1 - L_2$	x_2	n_2	N_2
...			
$L_{k-1} - L_k$	x_k	n_k	N_k
n			

Figura 1.14: Datos organizados en tabla

Medidas de dispersión: Indican la mayor o menor concentración de los datos con respecto a las medidas de centralización.
Los más usuales son: Desviación típica, coeficiente de variación, rango, varianza...

Fuentes de variabilidad.
Imaginemos que los estudiantes de Bioestadística reciben diferentes calificaciones en la asignatura (variabilidad). ¿A qué puede deberse?.

Lo que vemos es que hay diferencias individuales en el conocimiento de la materia. ¿Podría haber otras razones (fuentes de variabilidad)?
Supongamos, por ejemplo, que todos los alumnos poseen el mismo nivel de conocimiento. ¿Las notas serían las mismas en todos? Seguramente no.

Causas:

- Dormir poco el día del examen.
- Diferencias individuales en la habilidad para hacer un examen.

- El examen no es una medida perfecta del conocimiento.
- Variabilidad por error de medida.
- En alguna pregunta difícil, se duda entre varias opciones, y al azar se elige la mala.
- Variabilidad por azar, aleatoriedad.

Amplitud o Rango ('range')
Diferencia entre observaciones extremas.

Rango intercuartílico ('interquartile range'=IQR):
Es la distancia entre primer y tercer cuartil.
Rango intercuartílico = P75 - P25 = Q3 - Q1

Varianza S^2 ('Variance'):
Mide el promedio de las desviaciones (al cuadrado) de las observaciones con respecto a la media.

- Es sensible a valores extremos (alejados de la media).
- Sus unidades son el cuadrado de las de la variable.

$$S^2 = \frac{1}{n}\left(x_i - \overline{x}\right)^2$$

$$S^2 = \frac{\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2}{n}$$

$$\widehat{S}^2 = \frac{\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2}{n-1}$$

Desviación típica ('standard deviation', SD, sd...)
Es la raíz cuadrada de la varianza.
Tiene las misma dimensionalidad (unidades) que la variable. Versión 'estética' de la varianza.

$$S = \sqrt{S^2}$$

Coeficiente de variación
Es la razón entre la desviación típica y la media.

- Mide la desviación típica en forma de "qué tamaño tiene con respecto a la media"

- También se la denomina variabilidad relativa.
- Es una cantidad adimensional. Interesante para comparar la variabilidad de diferentes variables.
- Si el peso tiene CV=30 % y la altura tiene CV=10 %, los individuos presentan más dispersión en peso que en altura.

$$CV = \frac{S}{\overline{x}}$$

```
# Dispresión
pesos <- rnorm(1000, 3, 0.8)
range(pesos)
```

```
## [1] 0.7249 5.3832
```

```
IQR <- (quantile(pesos, 0.75, names = F) - quantile(pesos,
0.25, names = F))
IQR
```

```
## [1] 1.021
```

```
var(pesos)
```

```
## [1] 0.6233
```

```
sd(pesos)
```

```
## [1] 0.7895
```

```
CV <- sd(pesos)/mean(pesos)
CV
```

```
## [1] 0.2637
```

Medidas de forma. Las medidas de forma son la asimetría y la curtosis.

Asimetría

- Las discrepancias entre las medidas de centralización son indicación de asimetría.
- Coeficiente de asimetría (positiva o negativa).
- Distribución simétrica (asimetría nula).

Curtosis o apuntamiento
La curtosis nos indica el grado de apuntamiento (aplastamiento) de una distribución con respecto a la distribución normal o gaussiana. Es una medida adimensional.

- Platicúrtica (aplanada): curtosis < 0
- Mesocúrtica (como la normal): curtosis = 0
- Leptocúrtica (apuntada): curtosis > 0

Regla aproximativa (para ambos estadísticos).

- Curtosis y/o coeficiente de asimetría entre -1 y 1, es generalmente considerada una muy ligera desviación de la normalidad.
- Entre -2 y 2 tampoco es malo del todo, según el caso.

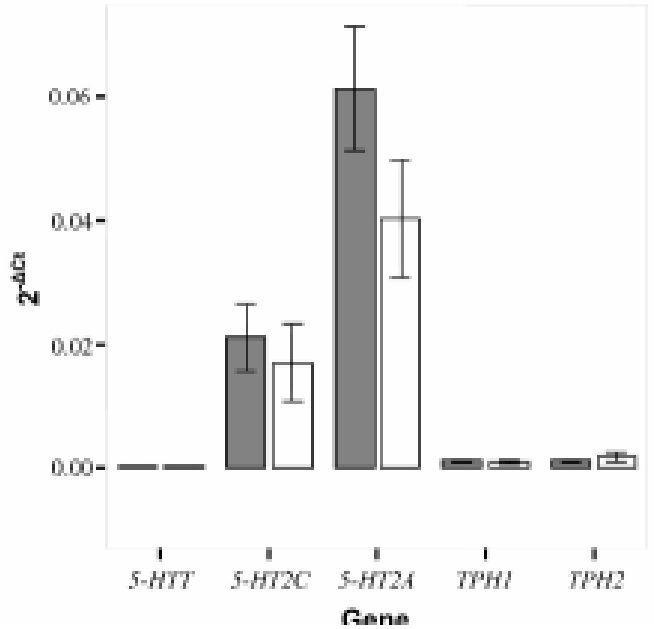
Una nota al dorso. Error típico de la media (SEM) Error típico de la media: Cuando estimamos la media a partir de una muestra de un determinado tamaño (n) los valores que toma la media en las diferentes muestras varía. A la desviación típica de los valores que toma el estadístico se le denomina error típico de la media.

Da una idea de la variabilidad del estadístico.
¡Ojo!: No de la distribución de la variable.

$$\sigma_{\overline{x}} = \frac{\sigma}{\sqrt{n}}$$

$$S_{\overline{x}} = \frac{\widehat{S}}{\sqrt{n}}$$

Nota: Un 'error' muy típico de los investigadores es mostrar medias y errores típicos de la media en lugar de la desviación típica y además poner bar plots.
¿Es un error o es intencionado?
¿Por qué solemos mostrar gráficos de barras y errores típicos de la media (SEM)?



(a) Data represents the mean +SEM for the individual cells studied (N=7 for both groups). Current trace calibration bar represents 1000 pA...

(b) El Coyote. Corporación Acme / Típica figura con barras y errores típicos de la media

Figura 1.15: Histogramas con barra de error

6.2 Algunos gráficos útiles (muy por encima)

Boxplot Un diagrama de caja, John Tukey (1977), es un gráfico, basado en cuartiles, mediante el cual se visualiza un conjunto de datos. Está compuesto por un rectángulo (la caja) y dos brazos (los bigotes).
También llamados 'diagramas de cajas y bigotes'.

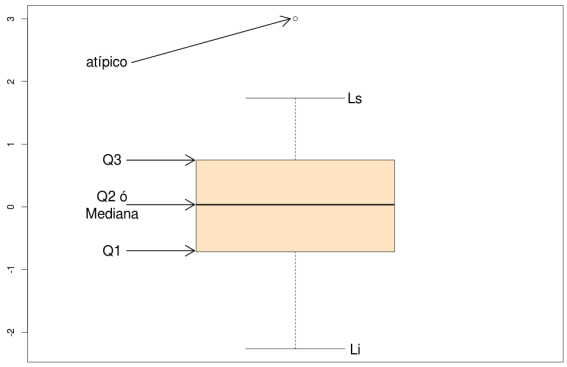


Figura 1.16: Boxplot o diagrama de caja y bigotes

Es un gráfico que suministra información sobre los valores mínimo y máximo, los cuartiles Q1, Q2 o mediana y Q3, y sobre la existencia de valores atípicos y simetría de la distribución.

- $L_i = Q_1 - 1{,}5IQR$
- $L_s = Q_3 + 1{,}5IQR$
- $IQR = Q_3 - Q_1$

Los valores atípicos son los inferiores a L_i y los superiores a L_s

```
boxplot(pesos)
```

```
p1 <- rnorm(1000, 3, 0.8)
p2 <- rnorm(1000, 2, 0.5)
```

```
p <- c(p1, p2)
grupo <- c(rep("M", 1000), rep("H", 1000))
df <- data.frame(p, grupo)
str(df)
```

```
## 'data.frame':    2000 obs. of  2 variables:
##  $ p    : num  1.68 1.76 3.65 3.77 3.62 ...
##  $ grupo: Factor w/ 2 levels "H","M": 2 2 2 2 2 ...
```

```
head(df)
```

```
##       p grupo
## 1 1.684     M
## 2 1.762     M
## 3 3.650     M
## 4 3.774     M
## 5 3.619     M
## 6 4.487     M
```

```
boxplot(df$p ~ df$grupo)
```

- Proporcionan una visión general de la simetría de la distribución de los datos, si la media no está en el centro del rectángulo, la distribución no es simétrica.
- Son útiles para ver la presencia de valores atípicos.
- Muy útiles para comparar distribuciones.

Histograma Un histograma es una representación gráfica de una variable en forma de barras, donde la superficie de cada barra es proporcional a la frecuencia de los valores representados. Representar histogramas en R es tan sencillo como crear un objeto hist, con la función hist().

Probar el siguiente código e ir observando qué cosas ocurren.

```
# histogramas

head(df)
```

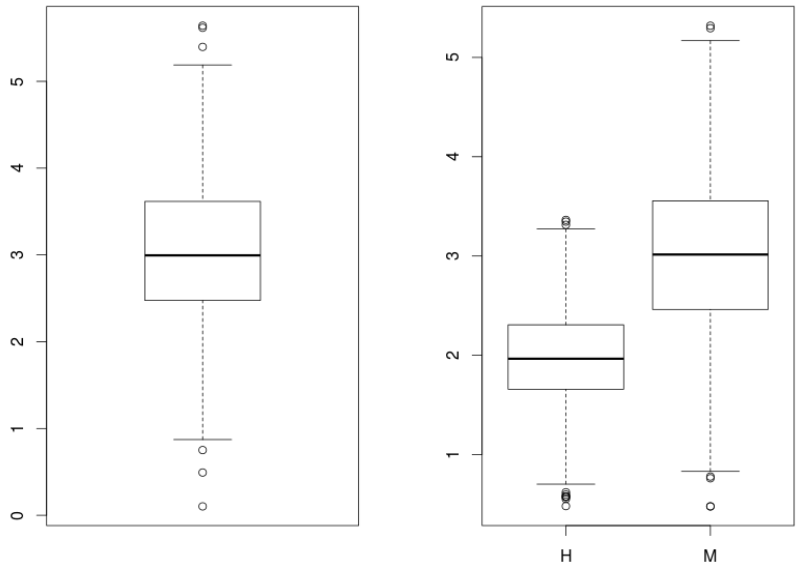


Figura 1.17: Boxplot simple y de dos variables. Comparación

```
##       p grupo
## 1 1.684     M
## 2 1.762     M
## 3 3.650     M
## 4 3.774     M
## 5 3.619     M
## 6 4.487     M
```

```
str(df)
```

```
## 'data.frame':    2000 obs. of  2 variables:
##  $ p    : num  1.68 1.76 3.65 3.77 3.62 ...
##  $ grupo: Factor w/ 2 levels "H","M": 2 2 2 2 ...
```

```
hist(df$p)   #por defecto
```

```
hist(df$p, breaks = 20)
```

```
hist(df$p, breaks = 20, probability = TRUE)

rug(jitter(p))  # aspecto: añadimos marcas en el eje OX de
observaviones, con un poco de ruido
```

```
hist(df$p, breaks = 20, probability = TRUE, main =
"Histograma de los pesos")
```

```
hist(df$p, breaks = 20, probability = TRUE, main =
"Histograma de los pesos", col = c("red", "yellow", "purple"))
```

```
colores <- c("red", "yellow", "purple")

hist(df$p, breaks = 20, probability = TRUE, main = "Mi
Histograma", xlab = "peso en kg",
    ylab = "frecuencia", col = colores)

# le añadimos su curva de densidad
lines(density(df$p))

# añadimos más cosas a la curva
lines(density(df$p), col = "blue", lty = 2, ps = 20)
```

Nota: Los histogramas fueron inventados por Florence Nightingale (1820-1910), Orden del Mérito del Reino Unido, británica, es considerada una de las pioneras en la práctica de la enfermería. Se la considera la madre de la enfermería moderna y creadora del primer modelo conceptual de enfermería. Destacó desde muy joven en la matemática, aplicando después sus conocimientos de estadística a la epidemiología y a la estadística sanitaria. Inventó los gráficos de sectores o histogramas para exponer los resultados de sus reformas. En 1858, fue la primera mujer miembro de la Statistical Society. También fue miembro honorario de la American Statistical Association.
Durante la guerra de Secesión en 1861 fue llamada por el gobierno de la Unión para que organizara sus hospitales de campaña durante la guerra, la cual redujo del 44 % al 2.2 % los heridos que fallecian en los hospitales.

6.3 Algunas funciones útiles para estadísticos descriptivos

Summary Hemos presentado a lo largo del texto muchas funciones para obtener descriptivos, la más recurrida es summary(), pero hay muchas más y algunas no están en los paquetes por defecto. Veamos algunos ejemplos.

```
p1 <- rnorm(1000, 3, 0.8)
p2 <- rnorm(1000, 2, 0.5)
p <- c(p1, p2)
altura <- c(rnorm(1000, 87, 7), rnorm(1000, 97, 6))
# length(p); hist(p)
grupo <- c(rep("M", 1000), rep("H", 1000))
df <- data.frame(p, altura, grupo)
head(df)
```

```
##       p altura grupo
## 1 3.961  89.30     M
## 2 3.892  93.15     M
## 3 1.136  85.80     M
## 4 2.885  82.27     M
## 5 2.368  81.68     M
## 6 2.380  99.89     M
```

```
str(df)
```

```
## 'data.frame':    2000 obs. of  3 variables:
##  $ p     : num  3.96 3.89 1.14 2.89 2.37 ...
##  $ altura: num  89.3 93.2 85.8 82.3 81.7 ...
##  $ grupo : Factor w/ 2 levels "H","M": 2 2 2 ...
```

```
summary(df)
```

```
##        p             altura       grupo
##  Min.   :0.35   Min.   : 64.6   H:1000
##  1st Qu.:1.90   1st Qu.: 86.5   M:1000
##  Median :2.37   Median : 92.6
##  Mean   :2.51   Mean   : 92.1
##  3rd Qu.:3.08   3rd Qu.: 97.6
##  Max.   :5.50   Max.   :122.2
```

```
summary(df[which(df$grupo == "M"), ])
```

```
##        p             altura      grupo
##  Min.   :0.35   Min.   : 64.6   H:   0
##  1st Qu.:2.49   1st Qu.: 82.5   M:1000
##  Median :3.04   Median : 87.3
##  Mean   :3.05   Mean   : 87.4
##  3rd Qu.:3.60   3rd Qu.: 92.4
##  Max.   :5.50   Max.   :110.9
```

```
# también podemos asignar dataframes a cada grupo para
# simplificar la sintaxis
df.M <- df[which(df$grupo == "M"), ]
summary(df.M)
```

```
##        p             altura      grupo
##  Min.   :0.35   Min.   : 64.6   H:   0
##  1st Qu.:2.49   1st Qu.: 82.5   M:1000
##  Median :3.04   Median : 87.3
##  Mean   :3.05   Mean   : 87.4
##  3rd Qu.:3.60   3rd Qu.: 92.4
##  Max.   :5.50   Max.   :110.9
```

```
df.H <- df[which(df$grupo == "H"), ]
summary(df.H)
```

```
##        p              altura      grupo
##  Min.   :0.394   Min.   : 79.3   H:1000
##  1st Qu.:1.651   1st Qu.: 92.8   M:   0
##  Median :1.988   Median : 96.7
##  Mean   :1.978   Mean   : 96.8
##  3rd Qu.:2.302   3rd Qu.:100.6
##  Max.   :3.614   Max.   :122.2
```

stat.desc() del paquete pastecs La función stat.desc() del paquete pastecs, tiene varias opciones muy interesantes:

stat.desc(x,basic=TRUE,desc=TRUE,norm=FALSE,p=0.95)

- `basic`: nº de valores, nº de nulls, nº de missing, min, max, rango, suma...
- `desc`: mediana, media, SEM, IC(95 %), var, sd, CV,
- `norm`: (OJO: no fijado en TRUE por defecto), curtosis, asimetría, Shapiro-Wilk

```
# del paquete 'pastecs', la función stat.desc()
# install.packages('pastecs')
library("pastecs")
```

```
## Loading required package: boot
```

```
stat.desc(df)
```

```
##                      p    altura grupo
## nbr.val      2.000e+03 2.000e+03    NA
## nbr.null     0.000e+00 0.000e+00    NA
## nbr.na       0.000e+00 0.000e+00    NA
## min          3.497e-01 6.460e+01    NA
## max          5.502e+00 1.222e+02    NA
## range        5.152e+00 5.760e+01    NA
## sum          5.025e+03 1.841e+05    NA
## median       2.369e+00 9.256e+01    NA
## mean         2.513e+00 9.207e+01    NA
## SE.mean      1.925e-02 1.798e-01    NA
## CI.mean.0.95 3.775e-02 3.525e-01    NA
## var          7.411e-01 6.463e+01    NA
## std.dev      8.609e-01 8.039e+00    NA
## coef.var     3.426e-01 8.732e-02    NA
```

Ojo: Sin incluir la variable 'grupo' y con la opción norm=T.

```
stat.desc(df[-3], norm = TRUE)
```

```
##                     p     altura
## nbr.val      2.000e+03  2.000e+03
## nbr.null     0.000e+00  0.000e+00
## nbr.na       0.000e+00  0.000e+00
```

```
## min           3.497e-01  6.460e+01
## max           5.502e+00  1.222e+02
## range         5.152e+00  5.760e+01
## sum           5.025e+03  1.841e+05
## median        2.369e+00  9.256e+01
## mean          2.513e+00  9.207e+01
## SE.mean       1.925e-02  1.798e-01
## CI.mean.0.95  3.775e-02  3.525e-01
## var           7.411e-01  6.463e+01
## std.dev       8.609e-01  8.039e+00
## coef.var      3.426e-01  8.732e-02
## skewness      5.259e-01 -2.050e-01
## skew.2SE      4.805e+00 -1.873e+00
## kurtosis     -8.171e-02 -7.713e-02
## kurt.2SE     -3.734e-01 -3.525e-01
## normtest.W    9.776e-01  9.964e-01
## normtest.p    3.623e-17  1.085e-04
```

```
stat.desc(df.M[-3], basic = FALSE, norm = TRUE)
```

```
##                      p    altura
## median        3.043262 87.269280
## mean          3.047432 87.382058
## SE.mean       0.025825  0.227225
## CI.mean.0.95  0.050677  0.445894
## var           0.666927 51.631337
## std.dev       0.816656  7.185495
## coef.var      0.267982  0.082231
## skewness     -0.004771  0.005607
## skew.2SE     -0.030840  0.036250
## kurtosis     -0.094020  0.040453
## kurt.2SE     -0.304205  0.130887
## normtest.W    0.999174  0.998500
## normtest.p    0.946942  0.554355
```

descrive de Hmisc Hay más funciones que ofrecen descriptivos, por ejemplo Hmisc (y muchas más).

```
# install.packages('Hmisc')
library("Hmisc")
describe(df$p)
```

```
## df$p
##       n missing  unique    Mean     .05     .10
##    2000       0    2000   2.513   1.309   1.517
##      .25     .50     .75     .90     .95
##    1.896   2.369   3.082   3.764   4.098
##
## lowest : 0.3497 0.3943 0.4479 0.5127 0.5854
## highest: 5.1446 5.1709 5.4146 5.4535 5.5020
```

```
head(df$p)
```

```
## [1] 3.961 3.892 1.136 2.885 2.368 2.380
```

```
describe(df$altura)
```

```
## df$altura
##       n missing  unique    Mean     .05     .10
##    2000       0    2000   92.07   78.69   81.29
##     .25     .50     .75     .90     .95
##   86.49   92.56   97.63  102.08  104.60
##
## lowest :  64.60  66.14  66.81  67.19  67.97
## highest: 112.13 112.73 113.30 115.87 122.20
```

```
head(df$altura, 25)
```

```
##  [1]  89.30  93.15  85.80  82.27  81.68  99.89 100.47
##       93.28  80.33 100.45
## [11]  73.27  87.38  92.07  95.30  78.63  90.17  81.02
##       80.72  83.06  91.63
## [21] 101.00  90.80  82.23  95.31  84.19
```

Función tapply() Con la función tapply nos podemos construir fácilmente nuestras tablas de descriptivos de una forma muy elegante.

```
# tapply
tapply(df$p, df$g, mean)
```

```
##     H     M
## 1.978 3.047
```

```
m <- tapply(df$p, df$g, mean)
s <- tapply(df$p, df$g, sd)
m2 <- tapply(df$p, df$g, median)
n <- tapply(df$p, df$g, length)
cbind(media = m, sd = s, mediana = m2, n)
```

```
##   media     sd mediana    n
## H 1.978 0.4935   1.988 1000
## M 3.047 0.8167   3.043 1000
```

Tablas de frecuencias y probabilidades cruzadas. Esto lo veremos más extensamente cuando hablemos del contraste χ^2

```
pais <- c("ES", "ES", "ES", "US", "US")
sexo <- c("F", "F", "M", "F", "M")

t <- table(pais, sexo)  # tabla de frecuencias absolutas
t
```

```
##     sexo
## pais F M
##   ES 2 1
##   US 1 1
```

```
# frec relativas
prop.table(t)  # porcentajes totales
```

```
##     sexo
## pais   F   M
##   ES 0.4 0.2
##   US 0.2 0.2
```

```
prop.table(t) * 100
```

```
##      sexo
## pais  F  M
##   ES 40 20
##   US 20 20
```

```
# porcentajes por filas
prop.table(t, 1)
```

```
##      sexo
## pais      F      M
##   ES 0.6667 0.3333
##   US 0.5000 0.5000
```

```
# porcentajes por columnas
prop.table(t, 2)
```

```
##      sexo
## pais      F      M
##   ES 0.6667 0.5000
##   US 0.3333 0.5000
```

Capítulo 2

Introducción a los contrastes.

1. Población y Muestra

La meta más simple a la hora de analizar datos es ”sacar las conclusiones más fuertes con la cantidad limitada de datos de los que disponemos”.

Dos problemas:

- Diferencias importantes pueden ser oscurecidas por variabilidad biológica e imprecisión experimental. Se hace complicado distinguir entre *diferencias reales y variabilidad aleatoria.*
- El ser humano es capaz de encontrar modelos. Nuestra inclinación natural (especialmente con nuestros propios datos) es concluir que las diferencias observadas son REALES, tendemos a minimizar los efectos de la variabilidad aleatoria.
- El rigor estadístico nos previene de cometer estos errores. (“si lo empleamos bien”).

Un ejemplo: Los estudiantes de Bioestadística reciben diferentes calificaciones en la asignatura (variabilidad). ¿A qué puede deberse?

Diferencias individuales en el conocimiento de la materia. ¿Podría haber otras razones (fuentes de variabilidad)?

Por ejemplo supongamos que todos los alumnos poseen el mismo nivel de conocimiento. ¿Las notas serían las mismas en todos? Seguramente No.

- Dormir poco el día del examen.

- Diferencias individuales en la habilidad para hacer un examen.
- El examen no es una medida perfecta del conocimiento.
- Variabilidad por error de medida.
- En alguna pregunta difícil, se duda entre varias opciones, y al azar se elige la mala.
- Variabilidad por azar, aleatoriedad.

La idea básica de las estadística es extrapolar, desde los datos recogidos, para llegar a conclusiones más generales sobre la población de la que se han recogido los datos.

Los estadísticos han desarrollado métodos basados en un modelo simple:

- Si razonablemente asumimos que los datos han sido obtenidos mediante un muestreo aleatorio de una población infinita. Analizamos estos datos y hacemos inferencias sobre la población.

Pero no siempre es *tan ideal.* En un experimento típico no siempre tomamos una muestra de una población, pero queremos extrapolar desde nuestra muestra a una situación más general. En esta situación aún podemos usar el concepto de población y muestra si definimos la muestra como los 'datos recogidos' y la población como los datos que habríamos recogido si repitiéramos el experimento un número infinito de veces.

No sólo es necesario que los datos provengan de una población. También es necesario que cada sujeto, (cada observación) sea 'escogido' independientemente del resto.

Ejemplos:

- Si realizas un experimento biomédico 3 veces, y cada vez por triplicado, no tenemos 9 valores independientes. Si promediamos los triplicados, entonces tenemos 3 valores medios independientes.
- Si en un estudio clínico muestreamos 10 pacientes de una clínica y otros 10 de un Hospital: no hemos muestreado 20 individuos independientes de la población. Probablemente hemos muestreado dos poblaciones distintas.

Hay tres enfoques básicos.

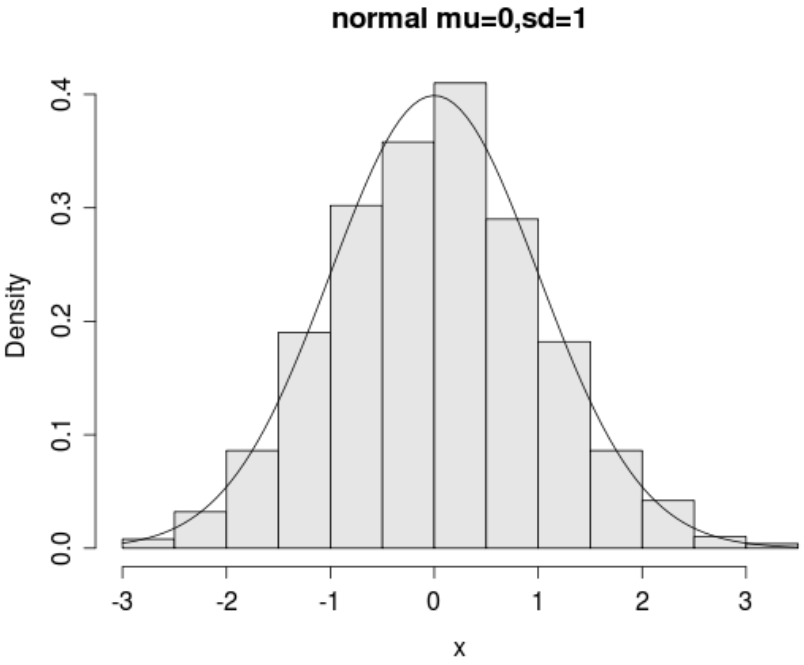
- El primer método consiste en asumir que la población sigue un distribución especial conocida como por ejemplo la Normal o Gaussiana (campana de Gauss).

- Los tests te permiten hacer inferencias sobre la media (y también sobre otras propiedades).
- Los tests más conocidos pertenecen a este enfoque.
- También se conoce como enfoque **paramétrico**.
- El segundo enfoque consiste en ordenar los valores y ordenarlos de mayor a menor (rangos) y comparar distribuciones de rangos.
- Es el principio básico de los **tests no-paramétricos.**
- El tercer enfoque es conocido como **'Resampling'** (se escapa de los objetivos de este curso, aquí también inlcuiríamos los métodos conocidos como *bootstraping*).
- Incluso hay un cuarto enfoque: **Métodos Bayesianos** (que también se escapan de los objetivos de este curso).

Ya hemos dicho que la idea básica de las estadística es extrapolar desde los datos recogidos para llegar a conclusiones más generales sobre la población de la que proceden los datos. El problema es que sólo podemos aplicar las inferencias a la población de la que hemos obtenido las muestras y *'generalmente'* queremos ir más lejos. Desafortunadamente la estadística no nos puede ayudar con estas extrapolaciones. En este caso se necesita **juicio científico y sentido común** para hacer inferencias más allá de las limitaciones de la estadística. Así el análisis estadístico es sólo parte de la interpretación de nuestros datos.

2. La distribución Normal

La distribución normal tiene características matemáticas especiales que hacen la base de la mayoría de los test estadísticos. La razón: el TCL (Teorema Central del Límite) que veremos más adelante.



$$\int_{-\infty}^{x} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\frac{(t-\mu)^2}{\sigma^2}} dt$$

¿Qué cosas observamos en esta distribución?

- La distribución de probabilidad normal es simétrica alrededor de su media. **(Coef simetría=0)**
- **media=mediana=moda** (todas las medidas de tendencia central coinciden).
- La curva normal desciende suavemente en ambas direcciones a partir del valor central.
- Es **asintótica**, lo que quiere decir que la curva se acerca cada vez más al eje X pero jamás llega a tocarlo. Es decir, las "colas" de la curva se extienden de manera indefinida en ambas direcciones.
- **Coef. Kurtosis = 0**
- El área total encerrada por f(x) vale 1: $\int_{-\infty}^{+\infty} f(x)dx = P\left(-\infty < X < +\infty\right) = 1$
- Si X es una v.a. que se distribuye según una normal de media μ y sd σ, $Z = \dfrac{x-\mu}{\sigma}$ se distribuye como una normal tipificada, es decir, de media 0 y sd 1.

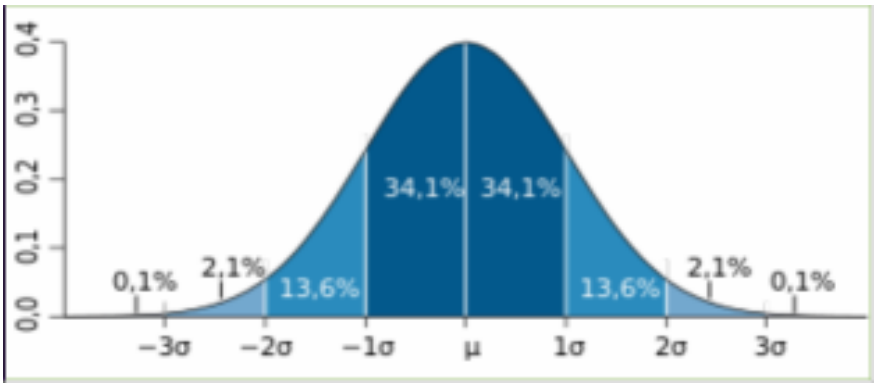


Figura 2.1: Tabulación de una normal

2.1 TCL: Teorema Central del Límite

"Si tus muestras son suficientemente grandes, la distribución de las medias seguirá una distribución Normal con la misma media que la distribución original".

De una forma más sencilla...

1. Dada una población con una distribución cualquiera.
2. Aleatoriamente obtenemos varias muestras de esa población. Calculamos sus medias.
3. Construimos un histograma de la distribución de frecuencias de las medias.
4. Esta distribución de medias sigue una 'Normal'.

¿Qué significa suficientemente grandes? Depende de cuán distinta sea la distribución de una normal.

Así pues, el teorema del límite central garantiza una distribución normal cuando n es suficientemente grande.

Existen diferentes versiones del teorema, en función de las condiciones utilizadas para asegurar la convergencia. Una de las más simples establece que es suficiente que las variables que se suman sean independientes, idénticamente distribuidas, con valor esperado y varianza finitas.

Nota: La aproximación entre las dos distribuciones es, en general, mayor en el centro de las mismas que en sus extremos o colas, motivo por el cual se prefiere el nombre "teorema del límite central" ("central" califica al límite, más que al teorema).

```
# simulación del TCL (aml!)  b<-rbinom(1000,10,0.25);hist(b)
# b<-rnorm(1000) b<-runif(100,0,1);hist(b) b<-rexp(100,1/10)
# hist(b)
b <- rchisq(100, 5)
hist(b)
plot(density(b))

N1 <- 1000  # el número de repeticiones debe ser alto
N2 <- 15  # tamaño muestras. Probar con 2, 5, 15, 30, 50
vectordemedias <- NA
for (i in 1:N1) {
    vectordemedias[i] = mean(sample(b, N2, replace = T))
}

# vectordemedias
hist(vectordemedias, breaks = 20)  # observamos si se parece
a una normal
SEM <- sd(vectordemedias)
SEM  # indicador de la calidad de la estimación de la media
mean(vectordemedias)
mean(b)  #observad que difieren poco
```

El TCL es el que nos permite calcular intervalos de confianza, p-valores (contrastes) y tamaños muestrales.

Ver el vídeo del profesor F. J. Barón López de la Universidad de Málaga: http://www.youtube.com/watch?v=FcDcJnw00hk

¿Cómo evaluamos la normalidad? Hay muchas maneras:

- **Gráficamente:** mediante histogramas, Q-Q plots...
- **Analíticamente:** con contrastes de hipótesis sobre normalidad: test de Shapiro-Wilk, de Kolmogorov–Smirnov, de Lilliefors, de Jaques-Bera... (los veremos luego)

2.2 Gráficos Q-Q

La manera de evaluarla mediante histogramas es muy 'rustica' pues simplemente lo que se hace es comparar el histograma frente a la densidad teórica de una normal con esos parámetros. Los Q-Q plots son más informativos, "Q" viene de cuantil. El método se emplea para el diagnóstico de diferencias entre la distribución de probabilidad de una población de la que se ha extraído una muestra aleatoria y una distribución usada para la comparación, es decir, no sólo para contrastar normalidad.

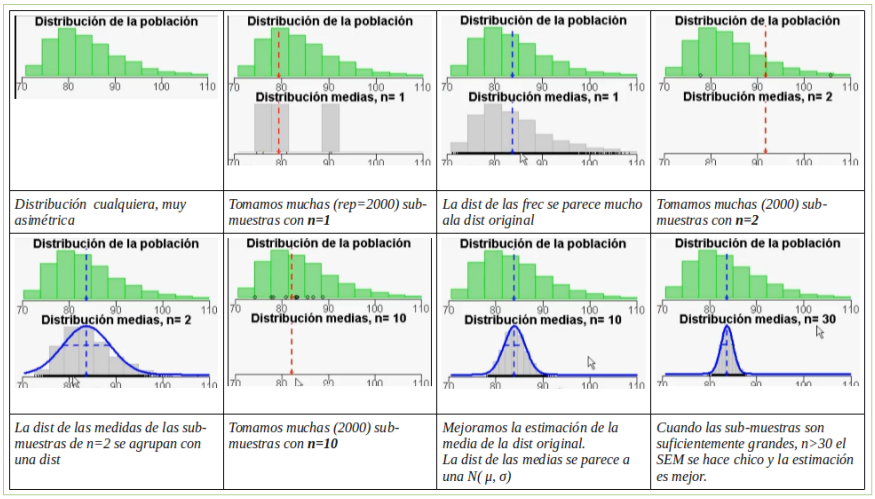


Figura 2.2: TCL

Cuando queremos contrastar normalidad para una muestra de tamaño n, se dibujan n puntos con los (n+1)-cuantiles de la distribución de comparación, la distribución normal, en el eje horizontal y el estadístico de k-ésimo orden (para k = 1,2,. . . n) de la muestra en el eje vertical. Si la distribución de la variable es la misma que la distribución de comparación se obtendrá, aproximadamente, una línea recta, especialmente cerca de su centro.

En R los podemos crear con la función `qqnorm()`.

```
attach(mtcars)
qqnorm(mpg)
qqline(mpg)
```

Ejercicio: Compara los siguientes Q-Q plots.

```
x <- rnorm(100, 0, 1)
qqnorm(x, main = "normal(0,1)")
qqline(x)

x <- rnorm(100, 10, 15)
qqnorm(x, main = "normal(10,15)")
qqline(x)

x <- rexp(100, 1/10)
```

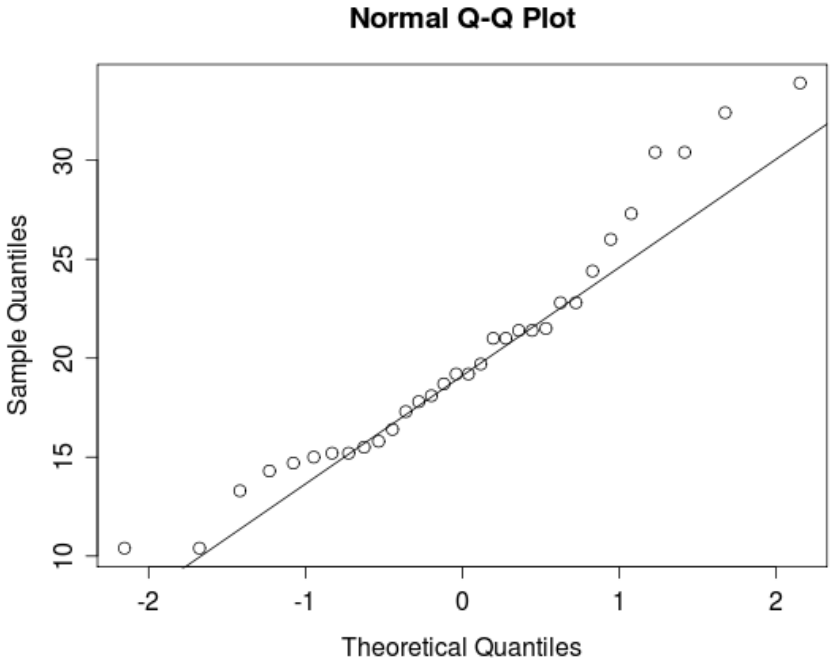


Figura 2.3: Q-Q Plot

```
qqnorm(x, main = "exponencial mu=10")
qqline(x)

x <- runif(100, 0, 1)
qqnorm(x, main = "unif(0,1)")
qqline(x)
```

3. Intervalos de confianza

Ya hemos visto que la media que calculamos de una muestra probablemente no sea igual a la media de la población. El tamaño de la diferencia dependerá del tamaño y de la variabilidad de la muestra (n, σ). Basándonos en el TCL, combinamos estos dos factores: **tamaño de la muestra** (n) y **variabilidad** (σ), para calcular intervalos de confianza a un determinado nivel de confianza; generalmente 95 %.

IC(95 %) ~ n, σ

Si asumimos que tenemos una muestra aleatoria de una población. Podemos estar seguros al 95 % de que el IC (95 %) contiene a la media poblacional. Dicho de otra manera: si generamos 100 IC(95 %) de una población, se espera que contengan la media poblacional en 95 casos y no lo haga en 5. No sabemos cuál es la media poblacional, así que no sabemos cuándo ocurre esto.

Cómo los calculamos:

$$(\overline{x} - z_{\alpha/2}\frac{S}{\sqrt{n}}, \overline{x} + z_{\alpha/2}\frac{S}{\sqrt{n}})$$

$z_{\sigma/2}$ para la normal es muy próximo a 1.96.

```r
# Intervalos de confianza para una Normal.
media <- 5
sd <- 2
n <- 20
inf <- NA
sup <- NA
ic <- data.frame(inf, sup)
error <- qnorm(0.975) * sd/sqrt(n)
ic[1, 1] <- media - error
ic[1, 2] <- media + error
ic

# Intervalos de confianza para una distribución T
media <- 5
sd <- 2
n <- 20
inf <- NA
sup <- NA
ic <- data.frame(inf, sup)
error <- qt(0.975, df = n - 1) * sd/sqrt(n)
ic[1, 1] <- media - error
ic[1, 2] <- media + error
ic
```

Veremos qué funciones como `t.test()` nos ofrecen estos IC.

En R para crear intervalos de confianza para proporciones empleamos la función `prop.test()`.

```r
# IC para una media de una muestra
x <- rnorm(10)
t.test(x)
```

```
##
##  One Sample t-test
##
## data:  x
## t = 1.275, df = 9, p-value = 0.2343
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  -0.2909  1.0421
## sample estimates:
## mean of x
##    0.3756
```

```
# IC para una proporción 342 de 100 dijeron que sí
# ¿cuál es el IC para la prop?
prop.test(42, 100)
```

```
##
##  1-sample proportions test with continuity correction
##
## data:  42 out of 100, null probability 0.5
## X-squared = 2.25, df = 1, p-value = 0.1336
## alternative hypothesis: true p is not equal to 0.5
## 95 percent confidence interval:
##  0.3233 0.5229
## sample estimates:
##    p
## 0.42
```

Para la mediana podemos calcularlo empleando la función wilcox.test().

```
x <- c(110, 12, 2.5, 98, 1017, 540, 54, 4.2, 150, 432)
mean(x)
```

```
## [1] 242
```

```
median(x)
```

```
## [1] 104
```

```
wilcox.test(x, conf.int = T)
```

```
##
##  Wilcoxon signed rank test
##
## data:  x
## V = 55, p-value = 0.001953
## alternative hypothesis: true location is not equal
## to 0
## 95 percent confidence interval:
##   33.0 514.5
## sample estimates:
## (pseudo)median
##            150
```

¿Qué asumimos cuando interpretamos un IC para una media?:

- Muestreo aleatorio de una población.
- La población se distribuye, aproximadamente, como una Normal.

- Si n es grande no es tan importante está condición.
- Existe independencia de las observaciones.

Nota: Si se viola alguna de estas condiciones, el IC real es más 'ancho' que el calculado.

Así pues un intervalo de confianza es un rango de valores (calculado en una muestra) en el cual se encuentra el verdadero valor del parámetro, con una probabilidad determinada.

- Nivel de confianza $1-\alpha$: probabilidad de que el verdadero valor del parámetro se encuentre en el intervalo.
- Nivel de significación α: probabilidad de equivocarnos.
- Normalmente $1-\alpha = 95\,\%$ ($\alpha = 5\,\%$)

4. P-valor. Contrastes de hipótesis

Observar medias maestrales diferentes no es suficiente evidencia de que sean diferentes las medias poblacionales. Es posible que sean iguales y que la diferencia

observada se deba a una coincidencia, nunca se puede estar seguro. Lo único que se puede hacer es **calcular las probabilidades de equivocarnos**.

Si las poblaciones tienen la misma media realmente: ¿cuál es la probabilidad de observar una diferencia tan grande o mayor entre las medias maestrales?

- El p-valor es una probabilidad de 0 a 1.
- Si p es pequeño, podemos concluir que la diferencia entre muestras ('probablemente') no es debida al azar.
- Concluiríamos que tiene distintas medias.

Otra forma de ver lo mismo.

- Los estadísticos hablan de hipótesis nula (H_0).
- La hipótesis nula dice que "no hay diferencia entre las medias".
- La hipótesis nula es lo contrario que la hipótesis experimental.
- Así podemos definir **p-valor** como la ***probabilidad de observar una diferencia tan grande o mayor que la observada si la hipótesis nula fuera cierta***.

Así pues:

- Se dice que rechazamos la hipótesis nula si $p < 0{,}05$ y la diferencia es estadísticamente significativa (ss).
- Si $p > 0{,}05$, **no rechazamos la hipótesis nula** y decimos que la diferencia no es estadísticamente significativa (ns).
- No podemos decir que la H_0 sea verdad, simplemente "no la rechazamos", es decir, no tenemos suficiente evidencia para rechazar la hipótesis de igualdad (la H_0).

¿En qué se basa el cálculo del p-valor? Pues en el TCL y en las propiedades de la distribución normal, la cual tenemos tabulada.

Se puede también abordar el tema de los contrastes de hipótesis y cálculo del p-valor desde una perspectiva integradora con el concepto de intervalo de confianza.

Supongamos que deseamos hacer un contraste acerca de **un parámetro q, de la población**. Para llevarlo a cabo consideraremos la distribución de algún estadístico muestral que de alguna manera se corresponda y se relacione con el parámetro q. Designemos en general a este estadístico como T. **Si con los datos muestrales obtenemos un valor concreto para T tal que pertenezca a una determinada región del campo de variación de T optaremos por no rechazar la hipótesis y en caso contrario por rechazarla**. Obviamente la clave del problema será delimitar la región del campo de variación de T que consideraremos como zona de aceptación

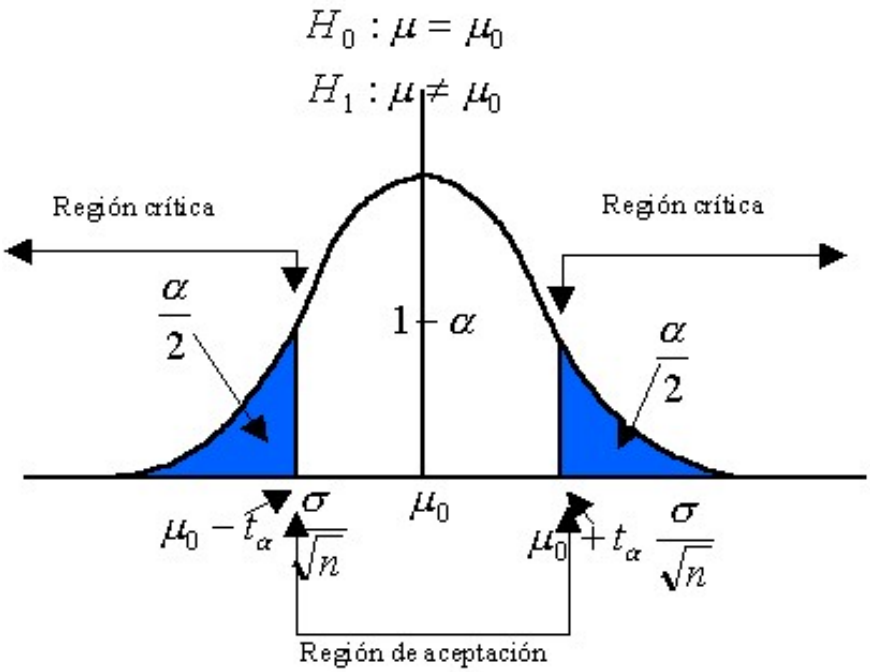


Figura 2.4: Regiones

de la hipótesis. **Esto se resolverá por un criterio probabílistico partiendo de la distribución muestral de T.**

Región crítica: Región del campo de variación del estadístico tal que si contiene al valor evaluado del mismo con los datos muestrales nos llevará a rechazar la hipótesis. La designaremos por R_1.

Región de aceptación: Es la región complementaria a la región crítica. Si el valor evaluado del estadístico pertenece a ella: no rechazamos la hipótesis. La designaremos por R_0. Ambos conjuntos/regiones son disjuntos.

NOTA: La región de aceptación no es más que un intervalo de confianza al 95 % (si la significación es 0.05) del estadístico T.

Un ejemplo intuitivo: *Supongamos que un fabricante de coches dice que un determinado vehículo recorre 25m por galón de gasolina, 25mpg. El consumidor pregunta a 10 usuarios su consumo y en media le responden que gasta 22mpg con una sd de 1,5. ¿Nos engaña el fabricante?*

$$H_0 : \mu = 25$$

$$H_a : \mu < 25$$

(la función `t.test()` no funciona porque los datos vienen resumidos, ¡no tenemos los datos crudos!, así que construimos nosotros el estimador y el p-valor).

```
# t test a mano (aml!)  asumimos mu=25 (H0)
mua <- 22
sda <- 1.5
n <- 10
t <- (mua - 25)/(sda/(sqrt(n)))
t
```

```
## [1] -6.325
```

```
# ahora con la función 'pt', porque asumimos que el
# estadístico 't' sigue una dist t (TCL) calculamos
# la probabilidad de que ese valor se dé
pt(t, df = n - 1)
```

```
## [1] 6.847e-05
```

```
# ¡Ojo!¡Hay que mosquearse...!
```

5. Potencia Estadística

Cuando un experimento concluye diciendo que no se ha encontrado una 'diferencia significativa', no implica que no haya diferencia; simplemente no la hemos encontrado. Decimos entonces que no tenemos *suficiente evidencia para rechazar la hipótesis nula (la igualdad), así que la asumimos*.

Posibles razones para un p-valor no significativo ($p > 0{,}05$):

- En verdad no hay diferencia, es decir, es correcto nuestro test (nuestro p-valor) y no hay diferencias reales poblacionales.
- Sí hay diferencias poblacionales (no lo podemos saber con seguridad porque no conocemos el parámetro, sólo el estadístico/estimador).
  Algunas causas posibles:
- Tamaño de la muestra.
- Alta variabilidad.

Si éstas son las razones estamos cometiendo **error de tipo II**.

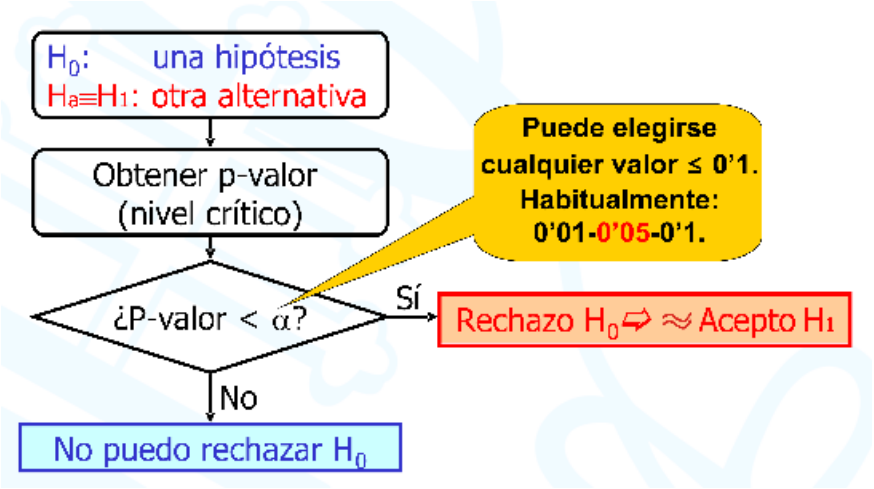


Figura 2.5: Contraste y significación

5.1 Error de tipo I y error de tipo II

- Cometemos error de tipo II o β, cuando 'afirmamos' que no hay diferencias ($p > 0{,}05$) y en verdad sí las hay.
- Cometemos error de tipo I o α, cuando 'afirmamos' que sí hay diferencias ($p < 0{,}05$) y en verdad no las hay.

Llamamos **potencia estadística** de un test a la capacidad de un test para revelar diferencias que realmente existen.

¿Cuánta potencia tiene nuestro análisis para encontrar hipotéticas diferencias en el caso de existir éstas? La potencia depende del tamaño (n) y de la cantidad de variación de la muestra (d, desviación estándar o típica). También influye el tamaño de lo que es para nosotros una diferencia, o ¿cuánto han de diferir para considerarlos distintos?
$n = (\frac{\sigma \times 1{,}96}{d})^2$, $z_{1-\alpha/2} = 1{,}96$, $\alpha = 0{,}05$

(De esto hablaremos más adelante en el curso)

5.2 Múltiples comparaciones

Interpretar un p-valor es sencillo, "si la hipótesis nula (*igualdad*) es cierta hay un 5 % de posibilidades de que una selección aleatoria de sujetos muestre una diferencia tan grande (o mayor) como la que muestran". Pero interpretar muchos p-valores a la vez puede no ser tan sencillo. Si testeamos diferentes hipótesis nulas

		Condición de la población	
		H0 verdadera	Ha verdadera
	Aceptar H0	Conclusión correcta	Error de tipo II
Conclusión			
	Rechazar H0	Error de tipo I	Conclusión correcta

Figura 2.6: Tipos de errores

(independientes) a la vez, con un nivel de significación del 0.05 hay más de un 5 % de probabilidades obtener un resultado significativo por azar.

Ejemplo: Si hacemos 3 tests con $\alpha = 0{,}05$. La probabilidad de no cometer error de tipo I es **0.95 (95 %) para cada test**. Suponemos que los tests son independientes.

- La probabilidad general de **NO** cometer error de tipo I es: (0.95)^3 = 0.875.
- Así la probabilidad general de cometer error de tipo I es: 1 - (0.95)^3 = 1 - 0.875 = 0.143, es decir, un **14,3 %**. ¡¡Se ha incrementado de 5 % a 14,3 %!! con sólo 3 comparaciones.

Cuantas más comparaciones hagamos más crece el error de tipo I.

Nº hip nulas Independientes	Prob de obtener al menos un p valor<0.05 por azar	Significación para mantener error tipo I=0.05
1	5%	0,05
2	10%	0.0253
3	14%	0.0170
4	19%	0.0127
....	...	....
100	99%	0.0005
N	$100(1-0.95^N)$	$1-0.95^{(N)}$

Figura 2.7: Comparaciones múltiples

Veremos más sobre este problema y de cómo resolverlo (correcciones de la significación y comparaciones planificadas) en el tema 5 de comparación de más de dos grupos.

6. Contrastes de normalidad

Cuando contrastamos lo primero que debemos tener en mente es cuál es la hipótesis nula. En los contrastes de normalidad la hipótesis nula es también conocida como la hipótesis de normalidad, es decir, *"no hay diferencias entre nuestra distribución y una distribución normal con esa media y esa sd"*.

Recordemos que hay muchas maneras de evaluar la normalidad:

- **Gráficamente:** histogramas, Q-Q plots...
- **Analíticamente:** test de Shapiro-Wilk, de Kolmogorov–Smirnov, de Lilliefors, de Jarque-Bera... (los veremos luego)

6.1 Métodos Gráficos

Habíamos visto los Gráficos Q-Q que quizás son siempre la primera aproximación al problema junto con los histogramas que repercutan la distribución *'empírica'* (real de nuestros datos), frente a la curva de densidad teórica, es decir la curva que generamos sabiendo la media y la sd que pensamos que ha de tener.

```
x <- mtcars$mpg
h <- hist(x, breaks = 10, xlab = "Histograma empírico",
main = "Hist + Normal teórica")

xfit <- seq(min(x), max(x), length = 40)
yfit <- dnorm(xfit, mean = mean(x), sd = sd(x))
yfit <- yfit * diff(h$mids[1:2]) * length(x)

lines(xfit, yfit, col = "blue", lwd = 2)
```

En R los podemos crear con la función qqnorm().

```
qqnorm(mtcars$mpg)
qqline(mtcars$mpg)
```

6.2 Métodos analíticos

i) Regla aproximativa Regla aproximativa de evaluación de normalidad según la asimetría y la curtosis.

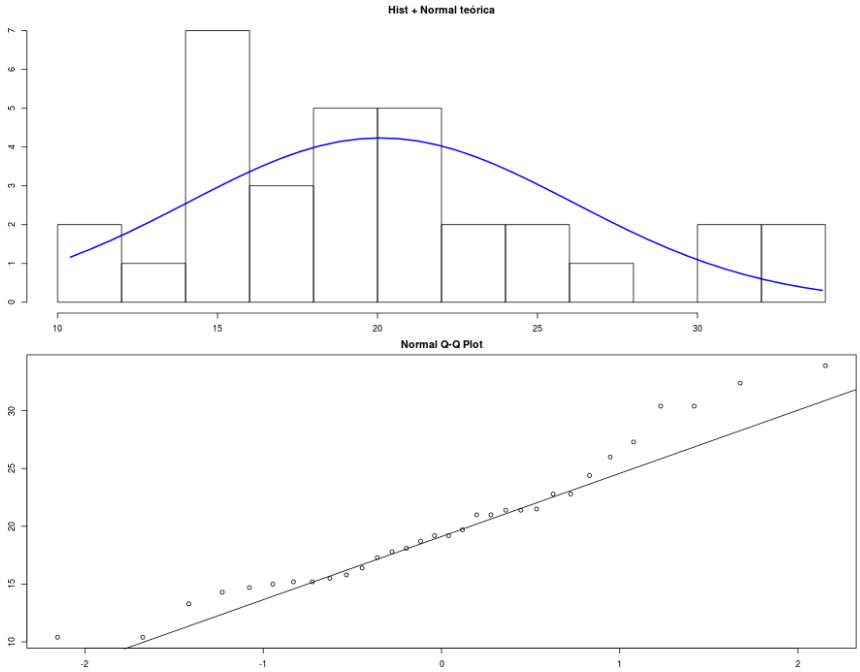


Figura 2.8: Histograma y normal teórica

Curtosis y/o coeficiente de asimetría entre -1 y 1, es generalmente considerada una muy ligera desviación de la normalidad.

Entre -2 y 2 tampoco es malo del todo, según el caso.

```
# cálculo de la curtosis y lel coef de asimetría en R
g3 <- mean((x - mean(x))^3)/(sd(x)^3)
g4 <- mean((x - mean(x))^4)/(sd(x)^4)
```

ii) Contraste sobre asimetría y curtosis. Se trata siemplemente de ver si son atípicos. Primero normalizamos los coeficientes:

$$Z_s = \frac{S}{S \times E_s}$$

$$Z_k = \frac{K}{S \times E_k}$$

¡¡Ojo, para ambas la media es 0!!

Si alguno de los valores es mayor en valor absoluto que 1.96 se considera significativo ($ |Z_s|>1.96$ = significativo). Y consideramos que hay demasiada asimetría (en su caso curtosis).

Nota: En muestras grandes es más interesante mirar a la distribución de forma gráfica (histograma).

iii) Contraste de normalidad de Shapiro-Wilk. El test de Shapiro-Wilk funciona bien con muestras pequeñas (menores de 50), para muestras grandes es equivalente al test de kolmogorov-smirnov que veremos luego. Se ejecuta con la función shapiro.test(x).

```
shapiro.test(x)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  x
## W = 0.9476, p-value = 0.1229
```

iv) Contraste de normalidad de Kolmogorov-Smirnov. El test de Kolmogorov-Smirnov puede testar si nuestra distribución viene de una distribución arbitraria, no sólo de la normal. Se ejecuta con la función `ks.test()`. Es preferible a Shapiro-Wilk si las muestras son grandes (mayores a 100).
`ks.test(x,y,... alternative=c("two.sided", "less", "greater"), excat=NULL)`

```
ks.test(x, pnorm)
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  x
## D = 1, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

v) Test de normalidad de Jarque-Bera. El test de Jarque-Bera no requiere estimaciones de los parámetros que caracterizan la normal.

```
# install.packages('tseries')
library(tseries)
x <- mtcars$mpg
jarque.bera.test(x)
```

```
##
##  Jarque Bera Test
##
## data:  x
## X-squared = 2.241, df = 2, p-value = 0.3261
```

La falta de normalidad influye en el modelo en:

- Los estimadores mínimo-cuadráticos no son eficientes (de mínima varianza).
- Los intervalos de confianza de los parámetros del modelo y los contrastes de significación son solamente aproximados y no exactos.

En otras palabras: Se pierde precisión.
EL TCL, nos permite 'saltarnos' esta hipótesis para muestras suficientemente grandes.

Causas que originan falta de normalidad:

- Existen observaciones heterogéneas
- Errores en la recogida de datos.
- El modelo especificado no es correcto (por ejemplo, no se ha tenido en cuenta una variable de clasificación cuando las observaciones proceden de diferentes poblaciones).
- Hay observaciones atípicas (estimadores robustos).
- Existe asimetría en la distribución (transformación de Box-Cox, HEV).

7. Contrastes de homogeneidad de varianza

El supuesto de homogeneidad de varianzas también se conoce como supuesto de homocedasticidad, nosotros habitualmente lo abreviamos como HOV, y dice que "la varianza es constante (no varía) en los diferentes niveles del factor". La falta de homocedasticidad se denomina heterocedasticidad (HEV).

Nota: Si el diseño es balanceado ($n_i = n_j$, para todo $i, j \in \{1, .., I\}$), la heterocedasticidad no afecta tanto a la calidad de los contrastes, a no ser que la varianza de la respuesta para algún grupo particular sea considerablemente mayor que para otros.

Reglilla: Balanceadas y HEV: ok si $\frac{\widehat{S}^2_{Max}}{\widehat{S}^2_{Min}} < 3$

(Si no hay balanceo esta regla puede usarse con un 2)

Si los tamaños muestrales son muy distintos se verifica que: si los grupos **con tamaños muestrales pequeños tienen mayor varianza** la probabilidad de cometer un error de tipo I en las pruebas de hipótesis será menor de lo que se obtiene y los niveles de confianza de los intervalos serán inferiores a lo que se cree (**conservador**).

Si por el contrario son los tratamientos con **tamaños muestrales grandes tienen mayor varianza** entonces se tendrá el efecto contrario y las pruebas serán más **liberales**.

i) Test de Levene El test de Levene del paquete 'car', función `levene.test()`. El test de Levene se resuelve con un ANOVA (ya veremos lo que es) de los valores absolutos de las desviaciones de los valores muestrales respecto a un estadístico de centralidad (media, mediana o media truncada) para cada grupo. La elección del estadístico de centralidad de los grupos determina la robustez y la potencia del test. Por robustez se entiende la habilidad del test para no detectar erróneamente varianzas distintas, cuando la distribución no es normal y las varianzas son realmente iguales. La potencia significa la habilidad del test para señalar varianzas distintas, cuando efectivamente lo son. Permite comparar más de dos grupos.

```
#install.packages("car");
library(car)

> levene.test(df$q1,df$gender, center="mean")
  Levene's Test for Homogeneity of Variance (center = "mean")
        Df F value  Pr(>F)
  group  1      15 0.03047 *
         3
```

```
> leveneTest(df$q1,df$gender, center="median")
  Levene's Test for Homogeneity of Variance (center = "median")
        Df F value Pr(>F)
  group  1     2.4 0.2191
         3
```

ii) F test para la evaluación de homocedasticidad En R podemos empelar la función `var.test()` para hacer un F test y comparar las varianzas de dos poblaciones.

```
>var.test(df$q1,df$q2)

            F test to compare two variances

    data:  df$q1 and df$q2
    F = 0.7059, num df = 4, denom df = 4, p-value = 0.7439
    alternative hypothesis: true ratio of variances is not
    equal to 1
    95 percent confidence interval:
     0.07349473 6.77966815
    sample estimates:
    ratio of variances
              0.7058824
```

O si tenemos un factor que me determina dos grupos.

```
>var.test(df$q1~df$gender)

            F test to compare two variances

    data:  df$q1 by df$gender
    F = 0.1667, num df = 2, denom df = 1, p-value = 0.2679
    alternative hypothesis: true ratio of variances is not
    equal to 1
    95 percent confidence interval:
     0.0002084636 6.4177215190
    sample estimates:
    ratio of variances
              0.1666667
```

iii) Test de Bartlett para testar más de dos grupos. Empleamos la función `bartlett.test()` para testar la homocedasticidad de más de dos muestras. El

test de Levene es menos sensible a la falta de normalidad que el de Bartlett. Sin embargo, si estamos seguros de que los datos provienen de una distribución normal, entonces el test de Bartlett es el mejor.

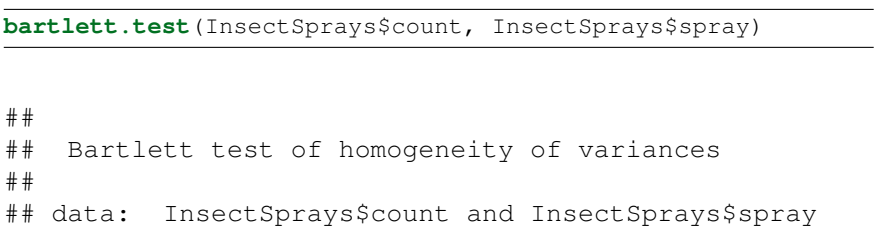
```
require(graphics)
str(InsectSprays)
```

```
## 'data.frame':    72 obs. of  2 variables:
##  $ count: num  10 7 20 14 14 12 10 23 17 20 ...
##  $ spray: Factor w/ 6 levels "A","B","C","D",..:
##  1 1 1 1 1 1 ...
```

```
head(InsectSprays)
```

```
##   count spray
## 1    10     A
## 2     7     A
## 3    20     A
## 4    14     A
## 5    14     A
## 6    12     A
```

```
plot(count ~ spray, data = InsectSprays)
```



```
bartlett.test(InsectSprays$count, InsectSprays$spray)
```

```
##
##  Bartlett test of homogeneity of variances
##
## data:  InsectSprays$count and InsectSprays$spray
## Bartlett's K-squared = 25.96, df = 5,
## p-value = 9.085e-05
```

iv) Test de Brown-Forsyth La función `plot.hov()` (paquete 'HH'), nos ofrece un test gráfico de HOV basado en Brown-Forsyth.

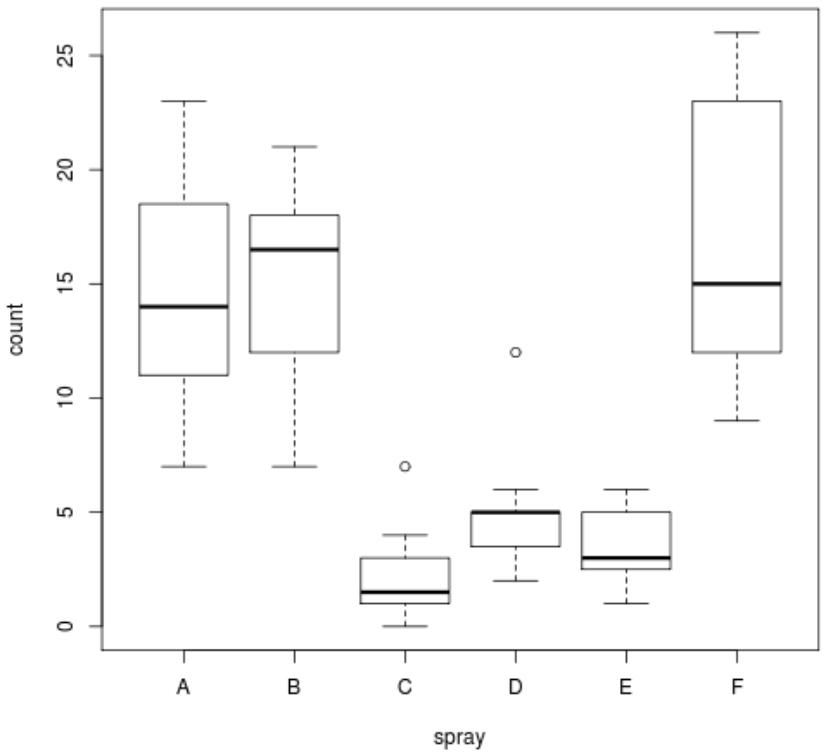


Figura 2.9: Bartlett

```
> #install.packages("HH")
> library(HH)
> hov(df$q1~df$gender)

      hov: Brown-Forsyth

    data:  df$q1
    F = 2.4, df:df$gender = 1, df:Residuals = 3, p-value =
     0.2191
    alternative hypothesis: variances are not identical

    Mensajes de aviso perdidos
    In model.matrix.default(mt, mf, contrasts) :
```

```
      variable 'group' converted to a factor

> plot.hov(df$q1~df$gender)
```

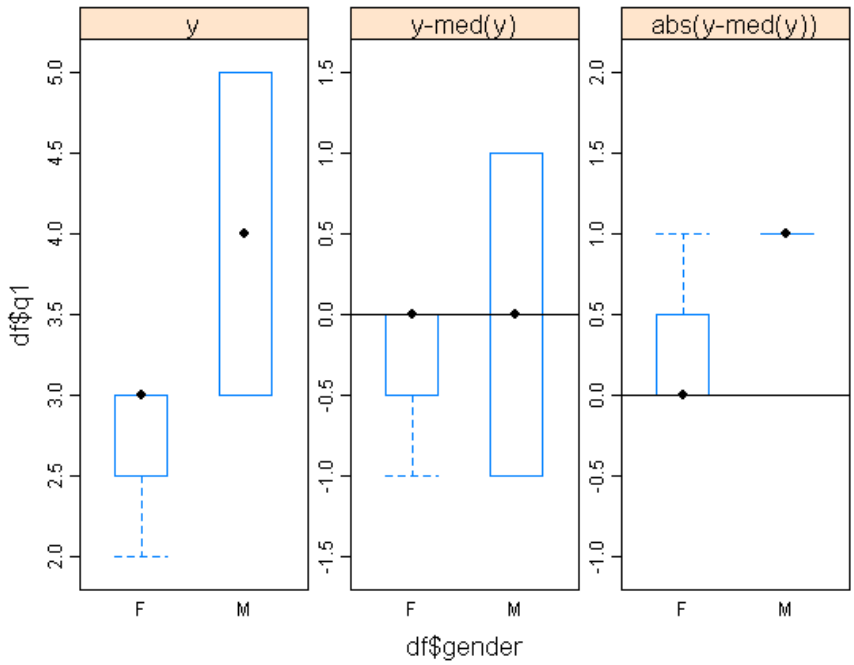


Figura 2.10: Brown-Forsyth

v) Test de Fligner-Killeen Podemos también comparar varianzas empleando un test no paramétrico con la función `Fligner.test()` que se basa en la mediana.

```
> fligner.test(InsectSprays$count, InsectSprays$spray)

            Fligner-Killeen test of homogeneity of variances

    data:  InsectSprays$count and InsectSprays$spray
    Fligner-Killeen:med chi-squared = 14.4828, df = 5
    p-value = 0.01282
```

Notas generales sobre la elección del test: El artículo original de Levene proponía la media como estadístico de centralidad. Brown y Forsythe (1974) extendieron este

test al utilizar la mediana e incluso la media truncada al 10 %. Sus estudios de Monte Carlo mostraron que la utilización de la media truncada mejoraba el test cuando los datos seguían una distribución de Cauchy (colas grandes) y la mediana conseguía mejorarlo cuando los datos seguían una (distribución asimétrica). Con la media se consigue el mejor test para distribuciones simétricas y con colas moderadas. Así pues, aunque la elección óptima depende de la distribución de los datos, la definición del test basada en la mediana es la recomendación general, ya que, proporciona una buena robustez para la mayoría de distribuciones no normales y, al mismo tiempo, una aceptable potencia. Si conocemos la distribución de los datos, podemos optar por alguna otra de las opciones.

Bibliografía

1. John Verzani (2001). *SimpleR-Using R for Introductory Statistics*. CRAN-R-Project, http://cran.r-project.org/doc/contrib/Verzani-SimpleR.pdf
2. Emmanuel Paradis (2003). *R para principiantes.2003*. CRAN-R-Project, http://cran.r-project.org/doc/contrib/rdebuts_es.pdf
3. Francesc Carmona (2003). Apuntes *Curso Básico de R*. Fecha de consulta 13:29, marzo 5, 2013, desde http://www.ub.es/stat/docencia/EADB/Curso%20basico%20de%20R.pdf
4. Joseph Adler (2009). *R in a nutshell, a desktop quick refrence*. O'Reilly Media, Inc.
5. Robert I. Kabacoff (2011). *R in acction. Data analysis and graphics with R*. Manning Publications Co.
6. Harvey Motulsky (1995). *Analyzing Data with Graphpad Prism*. GraphPad Software Inc.
7. Harvey Motulsky (1995). *Intuitive Biostatistics*. Oxford University Press.
8. Julían de la Horra Navarro (2003), *Estadística Aplicada* . Ediciones Dias de Santos.
9. Emilio Torres Manzanera (2010), *"Curso Avanzado del Paquete Estadístico R. Introducción a la modelización Estadística"*, Fecha de consulta: 10:42, abril 8, 2012 desde uce.uniovi.es/cursolineal/ModelizacionR.pdf
10. Variable aleatoria. (2012, 2 de abril). Wikipedia, La enciclopedia libre. Fecha de consulta: 10:42, abril 8, 2012 desde http://es.wikipedia.org/w/index.php?title=Variable_aleatoria&oldid=55047215
11. Teorema del límite central. (2012, 21 de abril).Wikipedia, La enciclopedia libre. Fecha de consulta: 20:34, mayo 8, 2012 desde http://es.wikipedia.org/w/index.php?title=Teorema_del_l%C3%ADmite_central&oldid=55555487
12. Francisco Javier Barón López (2012). *Apuntes y vídeos de Bioestadística*, Fecha de consulta 13:29, marzo 5, 2013, desde http://www.bioestadistica.uma.es/baron/apuntes/ http://video.google.es/videoplay?docid=-4793135283571454469&hl=es
13. Distribución normal. (2012, 23 de abril). Wikipedia, La enciclopedia libre. Fecha de consulta: 09:17, mayo 9, 2012 desde http://es.wikipedia.org/w/

index.php?title=Distribuci%C3%B3n_normal&oldid=55624567.

14. Brown, M. B. and Forsythe, A. B. (1974), *Journal of the American Statistical Association*, 69, 364-367.
15. R Core Team (2012). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, http://www.R-project.org/

Este libro tiene la intención de ayudar a investigadores de todas las áreas de conocimiento, algunos ya iniciados en estadística, que pretenden dar el salto a R y la vez repasar conceptos de una forma intuitiva, evitando excesivos formalismos. Pero, ¿todas las áreas?, ¿hay algún área en la que no sea necesario el análisis de datos o la estadística? En realidad, hoy día, desde la lingüística en el análisis computacional de textos, a la simple representación gráfica de la información, pasando por los estudios epidemiológicos, la biotecnología, la genética, la ecología, las ciencias de la educación, psicometría, econometría, finanzas, ciencias sociales, medicina, matemáticas, bellas artes, etc. requieren antes o después de análisis de datos. R no es solo un programa, es una forma de trabajar y manera de pensar, además admite sabores y versiones personales, pero sobre todo es un lenguaje, más concretamente es la lingua franca actual del análisis de datos.

ISBN 978-84-686-3628-3