

---

---

# Curso de R

---

---

Por  
Pascal



**UNIVERSIDAD COMPLUTENSE  
MADRID**

Grado en Ingeniería Informática  
FACULTAD DE INFORMÁTICA

Para dominio público  
**Curso de R**

MADRID, 2018–2019



# Antes de empezar...

## Sobre el curso

Este documento se ha realizado gracias a los cursos de **Universitat de les Illes Balears (UIB)** se agradece el conocimiento aportado.

## Sobre la autoria

Se ha procedido a grabar la realización de todo el curso en  $\text{\LaTeX}$ : [https://drive.google.com/drive/folders/1eB\\_R1zuqLIMibD7Jy0-sTREpGS2OJrwn?usp=sharing](https://drive.google.com/drive/folders/1eB_R1zuqLIMibD7Jy0-sTREpGS2OJrwn?usp=sharing)



# Sobre TEF<sub>L</sub>ON

TEFLON(CC0 1.0(DOCUMENTACIÓN) MIT(CÓDIGO))ES UNA PLANTILLA DE L<sup>A</sup>T<sub>E</sub>X CREADA POR DAVID PACIOS IZQUIERDO CON FECHA DE ENERO DE 2018. CON ATRIBUCIONES DE USO CC0.

Esta plantilla fue desarrollada para facilitar la creación de documentación profesional para Trabajos de Fin de Grado o Trabajos de Fin de Máster. La versión usada es la 1.3.

V:1.3 OVERLEAF V2 WITH PDFL<sup>A</sup>T<sub>E</sub>X, MARGIN 1IN, NO-BIB

## Contacto

**Autor:** DAVID PACIOS IZQUIERO

**Correo:** dpacios@ucm.es

**ASCII:** asciifdi@gmail.com

DESPACHO 110 - FACULTAD DE INFORMÁTICA



# Índice general

	Página
<b>1. Instalación y uso básico de R y de RStudio</b>	<b>1</b>
1.1. Instalación . . . . .	1
1.1.1. Instalar R . . . . .	1
1.1.2. Instalación de RStudio . . . . .	3
1.2. Paneles de RStudio . . . . .	6
1.3. Como guardar el trabajo . . . . .	8
1.4. Paquetes . . . . .	11
<b>2. Matemáticas básicas</b>	<b>15</b>
2.1. Operaciones y funciones reales básicas . . . . .	15
2.2. Cifras significativas y redondeo . . . . .	17
2.3. Definición de variables y funciones . . . . .	18
2.4. Números complejos . . . . .	20
<b>3. Representación gráfica y lectura de datos</b>	<b>23</b>
3.1. Data frames y plots . . . . .	23
3.2. Regresión lineal . . . . .	25
3.3. Formas básicas de lectura de datos externos . . . . .	27
3.4. Rectas de regresión y transformaciones logarítmicas . . . . .	30
<b>4. Vectores, entradas vacías, factores y listas generalizadas</b>	<b>41</b>
4.1. Vectores . . . . .	41
4.1.1. Construcción de vectores . . . . .	41
4.1.2. Operaciones con vectores . . . . .	45
4.1.3. Entradas y trozos de vectores . . . . .	48
4.2. Entradas vacías de un vector . . . . .	50
4.3. Factores . . . . .	51
4.4. Listas generalizadas . . . . .	52
<b>5. Matrices y vectores propios</b>	<b>57</b>
5.1. Matrices . . . . .	57
5.1.1. Construcción de matrices . . . . .	57
5.1.2. Entradas y trozos de matrices . . . . .	59
5.1.3. Funciones para matrices . . . . .	60
5.1.4. Álgebra matricial . . . . .	62
5.2. Cálculo de valores y vectores propios . . . . .	65

<b>6. Data frame</b>	<b>69</b>
6.1. Introducción . . . . .	69
6.2. Estructura . . . . .	69
6.3. Importar y exportar data frame . . . . .	75
6.4. Crear data frame . . . . .	77
6.5. Cómo modificar un data frame . . . . .	79
6.6. Cómo añadir filas y columnas a un data frame . . . . .	81
6.7. Cómo seleccionar trozos de un data frame . . . . .	83
6.8. Cómo aplicar una función a un data frame . . . . .	87
6.9. Cómo añadir las variables de un data frame al entorno global . . . . .	89
<b>7. Función plot</b>	<b>91</b>
7.1. Parámetros . . . . .	93
7.1.1. Parámetros de aspecto exterior . . . . .	93
7.1.2. Parámetros de aspecto de los puntos . . . . .	94
7.1.3. Parámetros de tipo de gráfico . . . . .	96
7.1.4. Parámetros de ejes de coordenadas . . . . .	102
7.2. Cómo añadir elementos a un gráfico . . . . .	103
7.2.1. Añadir puntos . . . . .	103
7.2.2. Añadir rectas . . . . .	104
7.2.3. Añadir curvas . . . . .	105
7.2.4. Añadir texto . . . . .	106
7.2.5. Añadir una leyenda . . . . .	107
<b>8. Tablas de frecuencias, diagramas de barras, otros gráficos básicos para datos cualitativos y un ejemplo completo</b>	<b>109</b>
8.1. Tablas de frecuencias . . . . .	109
8.1.1. Tablas unidimensionales de frecuencias . . . . .	109
8.1.2. Tablas bidimensionales de frecuencias . . . . .	110
8.1.3. Tablas multidimensionales de frecuencias . . . . .	112
8.2. Tablas de frecuencias a partir de data frames . . . . .	115
8.3. Diagramas de barras . . . . .	117
8.4. Otros gráficos básicos para datos cualitativos . . . . .	119
8.4.1. Diagrama circular . . . . .	119
8.4.2. Gráficos de mosaico . . . . .	120
8.5. Un ejemplo completo . . . . .	122
<b>9. Datos ordinales</b>	<b>129</b>
<b>10. Frecuencias, medidas de tendencia central y de posición, medidas de dispersión y diagrama de cajas</b>	<b>131</b>
10.1. Frecuencias . . . . .	131
10.2. Medidas de tendencia central y de posición . . . . .	132
10.2.1. Medidas de tendencia central . . . . .	132
10.2.2. Medidas de posición . . . . .	133
10.2.3. Cálculo del cuantil . . . . .	133
10.3. Medidas de dispersión . . . . .	134
10.3.1. Resumen estadístico . . . . .	135
10.4. Diagramas de cajas . . . . .	136



<b>11. Matrices de datos cuantitativos, covarianzas y correlaciones, y representación gráfica de datos multidimensionales</b>	<b>141</b>
11.1. Matrices de datos cuantitativos . . . . .	141
11.1.1. Estadísticos multidimensionales . . . . .	141
11.2. Covarianzas y correlaciones . . . . .	143
11.2.1. Covarianza entre dos variables . . . . .	143
11.2.2. Matrices covarianzas y covarianzas muestrales . . . . .	144
11.2.3. Correlación entre dos variables . . . . .	145
11.2.4. Propiedades de la correlación entre dos variables . . . . .	145
11.2.5. Matrices de correlaciones . . . . .	146
11.3. Representación gráfica de datos multidimensionales . . . . .	146
11.3.1. La función plot . . . . .	146
11.3.2. La función scatterplot3d . . . . .	147
11.3.3. Diagramas de dispersión con plot . . . . .	148
11.3.4. Diagramas de dispersión con la función pairs . . . . .	150
11.3.5. Diagramas de dispersión con spm . . . . .	151
<b>12. Agrupamiento de datos con R, estadísticos para datos agrupados e histogramas</b>	<b>153</b>
12.1. Agrupamiento de datos con R . . . . .	153
12.1.1. Agrupamiento de datos . . . . .	153
12.1.2. Agrupamiento de datos con R . . . . .	155
12.2. Estadísticos para datos agrupados . . . . .	157
12.2.1. Fórmulas de los estadísticos . . . . .	157
12.2.2. Cuantiles de datos agrupados . . . . .	159
12.3. Histogramas . . . . .	160
12.3.1. Histogramas en R . . . . .	161
12.3.2. Densidad de una variable . . . . .	164



# Capítulo 1

## Instalación y uso básico de R y de RStudio

R realiza los cálculos y RStudio es el entorno donde se van a realizar esos cálculos. Primero explicaremos cómo instalar R y después como instalar Rstudio.

### 1.1. Instalación

#### 1.1.1. Instalar R

Para instalar R primero nos dirigiremos a la web <https://www.r-project.org/>. Una vez que la hemos introducido en el buscador veremos lo siguiente:

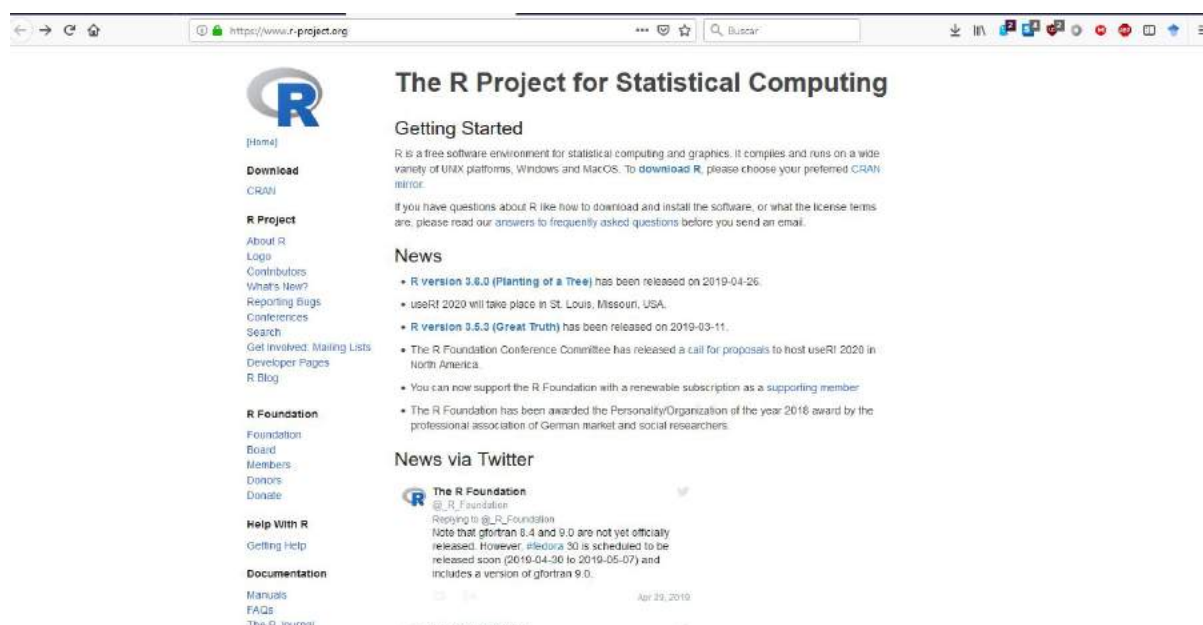


Figura 1.1: Página web RProject

Una vez que estemos en la página seleccionaremos [CRAN](#) en la esquina superior izquierda, en [Download](#). Cuando entremos, veremos muchas CRAN de muchos países, en nuestro caso buscaremos la de [Spain](#), que es la siguiente:



Figura 1.2: CRAN de España

Seleccionamos cualquiera de las direcciones que vemos. Una vez seleccionadas, nos dejará instalar R en Linux, Mac o Windows. Cliqueamos el sistema operativo que tengamos y procederemos a la instalación.

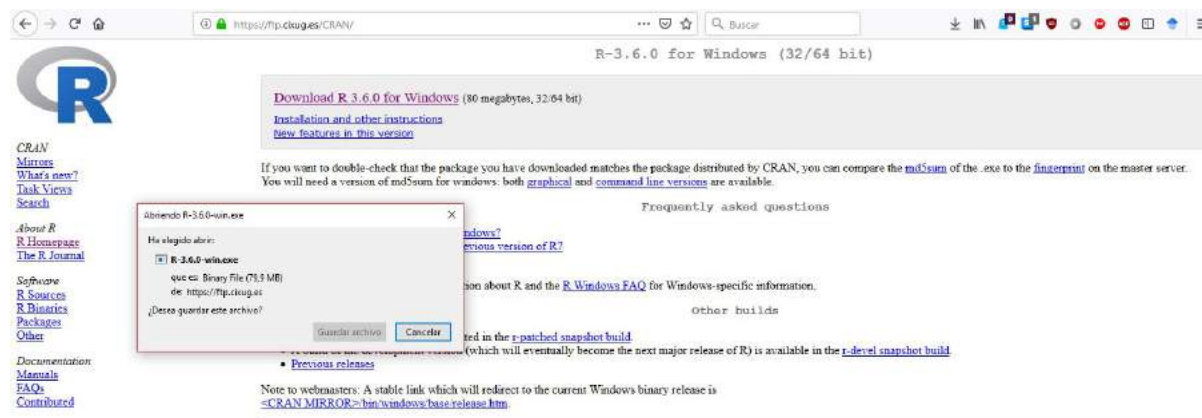


Figura 1.3: Instalación de R

Iniciamos la instalación y lo primero que nos pedirá será el idioma a utilizar durante la instalación.

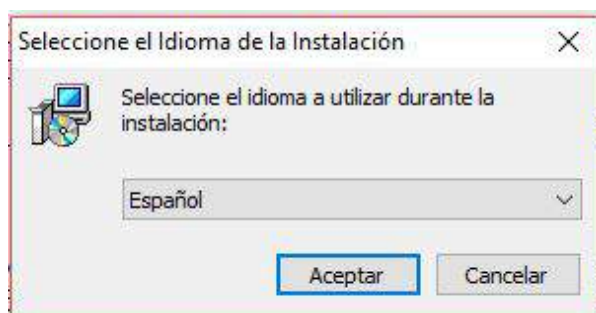


Figura 1.4: Idioma instalación R

Le damos a siguiente en el acuerdo de términos y condiciones, colocamos donde queremos instalar R.

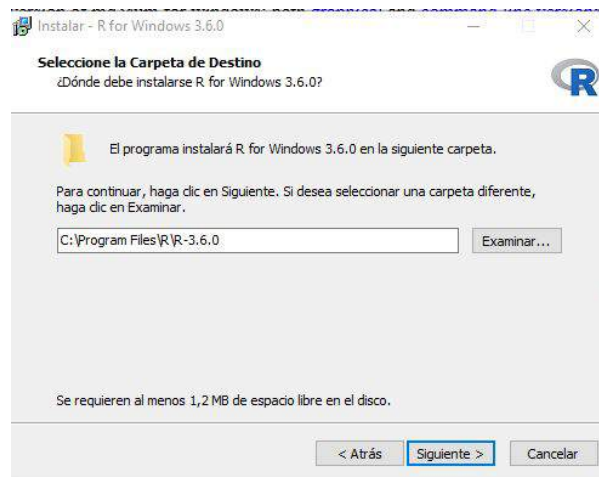


Figura 1.5: Localización de R

Seguidamente, seleccionamos que no deseamos utilizar las opciones de configuración y también seleccionamos las tareas adicionales.

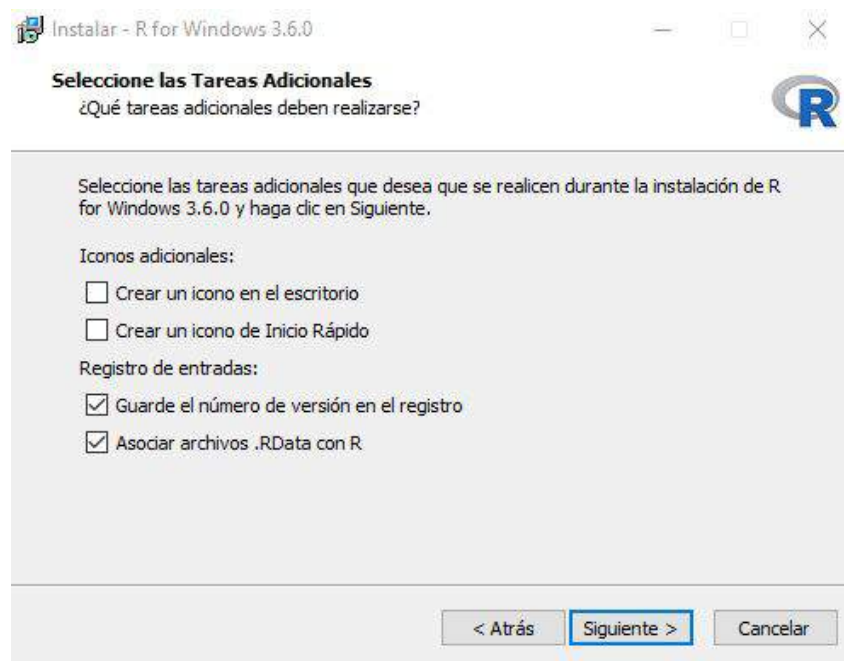


Figura 1.6: Tareas de R

Finalmente, seguimos dando a Siguiete hasta que nos permita instalarlo. El proceso tardará como máximo 2 minutos.

### 1.1.2. Instalación de RStudio

Como hemos dicho antes, RStudio es el entorno gráfico que nos permite utilizar de una manera más cómoda a R. Nos permite autocompletar el código, detectar errores en la sintaxis e incluye un sistema de menús de ayuda. Para instalarlo iremos a la web <https://www.rstudio.com/>, y una vez la pongamos en nuestro buscador veremos lo siguiente:

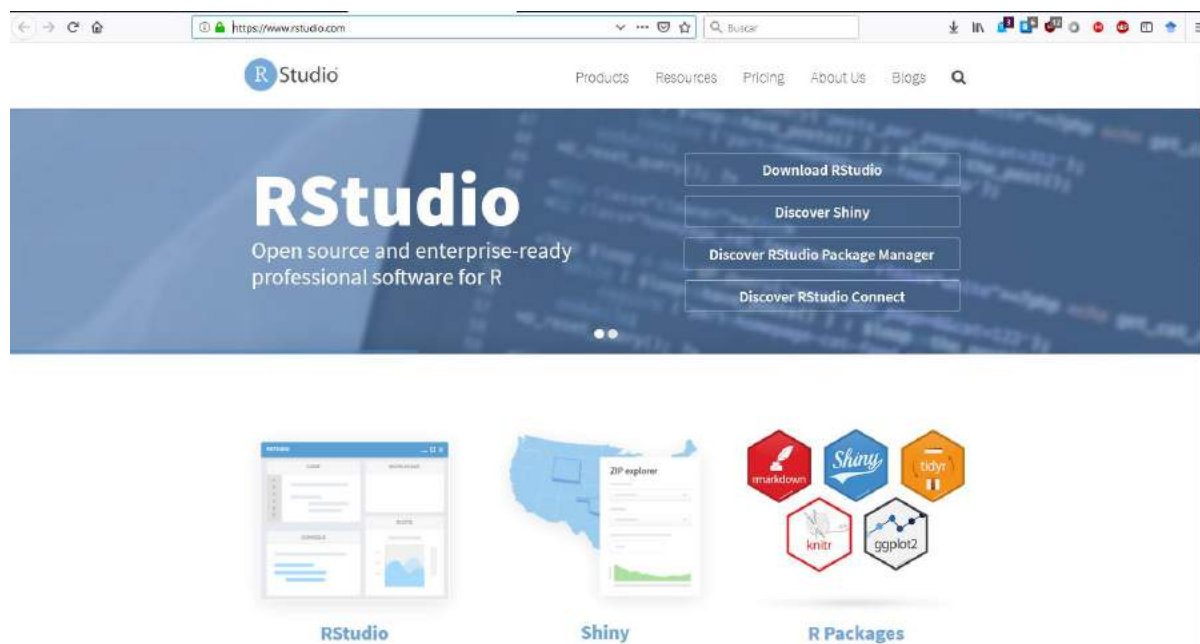


Figura 1.7: Web RStudio

Una vez dentro elegimos la opción de **Dowload Rstudio** y seleccionamos la versión **RStudio Desktop**, que es la versión gratuita.

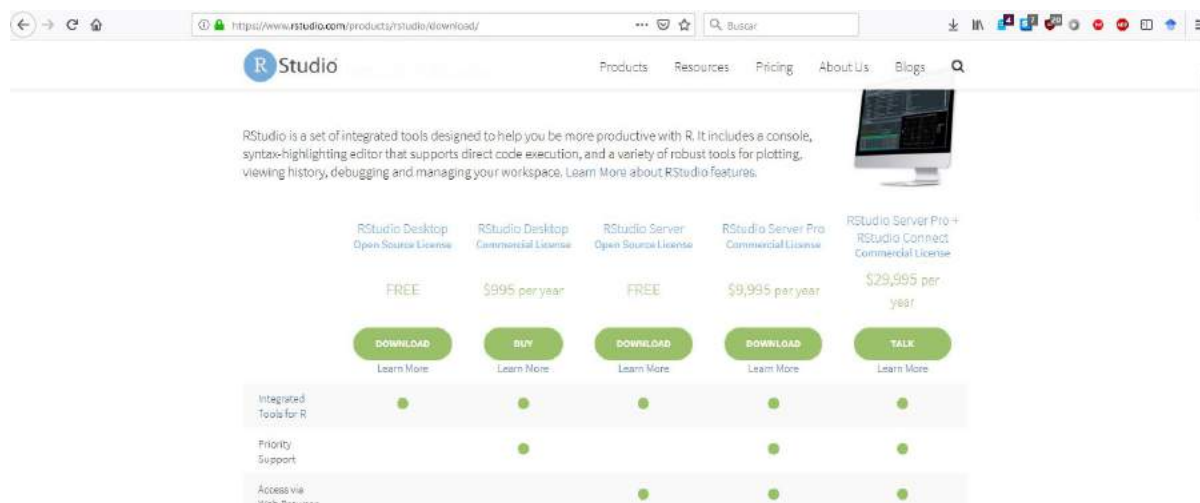


Figura 1.8: RStudio Desktop

La seleccionamos y nos dejará instalarla para las siguientes plataformas:

- Windows.
- Mac.
- Ubuntu.
- Fedora.
- Debian.

- OpenSUSE.
- SLES.

### Installers for Supported Platforms

Installers	Size	Date	MD5
RStudio 1.2.1335 - Windows 7+ (64-bit)	126.9 MB	2019-04-08	d0e2470f1f8ef4cd35a669aa323a2136
RStudio 1.2.1335 - Mac OS X 10.12+ (64-bit)	121.1 MB	2019-04-08	6c570b0e2144583f7c48c284ce299eef
RStudio 1.2.1335 - Ubuntu 14/Debian 8 (64-bit)	92.2 MB	2019-04-08	c1b07d0511469abfe582919b183eee83
RStudio 1.2.1335 - Ubuntu 16 (64-bit)	99.3 MB	2019-04-08	c142d69c210257fb10d18c045fff13c7
RStudio 1.2.1335 - Ubuntu 18 (64-bit)	100.4 MB	2019-04-08	71a8d1990c0d97939804b46cfb0aea75
RStudio 1.2.1335 - Fedora 19+/RedHat 7+ (64-bit)	114.1 MB	2019-04-08	296b6ef88969a91297fab6545f256a7a
RStudio 1.2.1335 - Debian 9+ (64-bit)	100.6 MB	2019-04-08	1e32d4d6f6e216f086a81ca82ef65a91
RStudio 1.2.1335 - OpenSUSE 15+ (64-bit)	101.6 MB	2019-04-08	2795a63c7efd8e2aa2dae86ba09a81e5
RStudio 1.2.1335 - SLES/OpenSUSE 12+ (64-bit)	94.4 MB	2019-04-08	c65424b06ef6737279d982db9eefcae1

Figura 1.9: Plataformas disponibles

Hacemos una instalación como si fuera un programa normal, le vamos dando a Siguiente en todos los pasos hasta que se instale. Una vez instalado veremos lo siguiente:

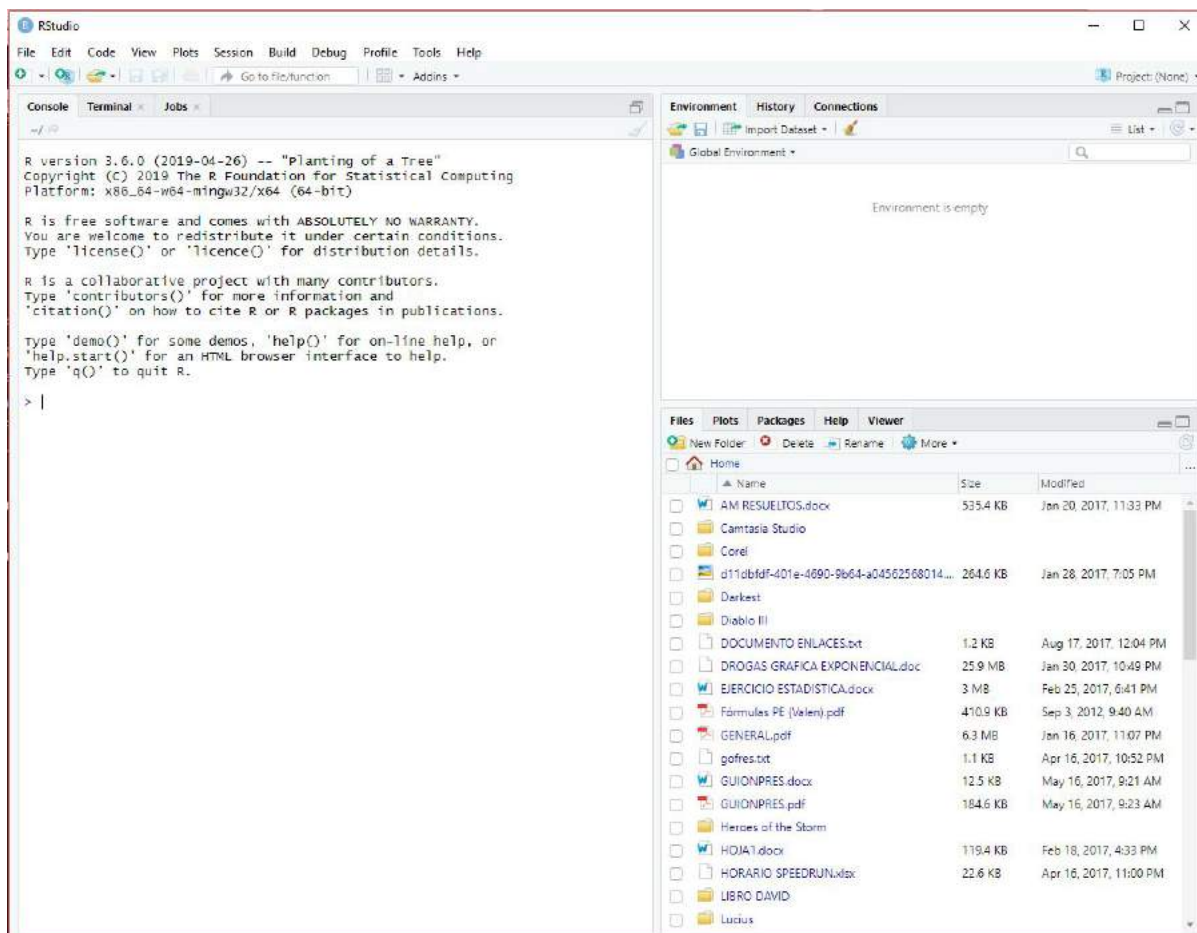


Figura 1.10: Interfaz RStudio

- A la izquierda está la consola donde se ejecutan los comandos R.
- A la derecha, en la parte superior tenemos una ventana que nos muestra nuestro entorno de trabajo, donde se verán las variables y funciones que se van cargando.
- A la derecha, en la parte inferior tenemos el directorio home donde R arranca por defecto. Esta ventana contiene varias pestañas:
  - a). **Files:** Archivos en el directorio actual.
  - b). **Plots:** Gráficos generados por el programa.
  - c). **Packages:** Librerías instaladas.
  - d). **Help:** Sistema de ayuda.
  - e). **Viewer:** Contenido web local.

## 1.2. Paneles de RStudio

Tenemos tres paneles, el primer panel es la consola, donde daremos nombres a nuestras variables, el segundo es el editor donde guardaremos nuestros scripts y por último, tenemos el script que nos permite crear nuestro programa y guardarlo.

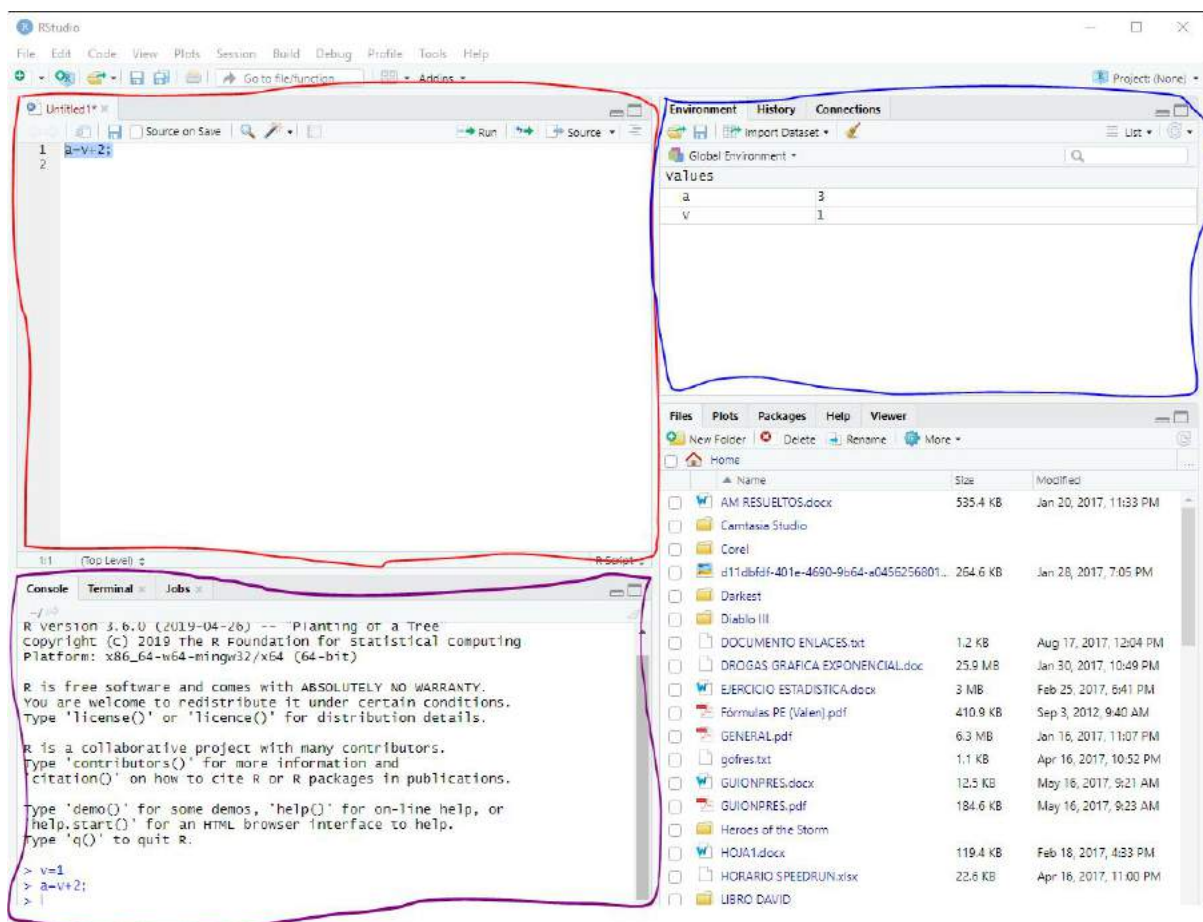


Figura 1.11: Paneles



En este caso, tendremos la **Consola** de color morado, el **Entorno** de color azul y el **Editor** de color rojo.

Además, en el Entorno podremos ver los valores de nuestras funciones, el historial de las funciones y volver a ejecutarlas.

Finalmente, tendremos en un cuarto panel la presentación de nuestros gráficos, los paquetes utilizados y la herramienta más importante, la ayuda donde podremos buscar los nombres de las distintas funciones por si se nos olvida algún nombre.

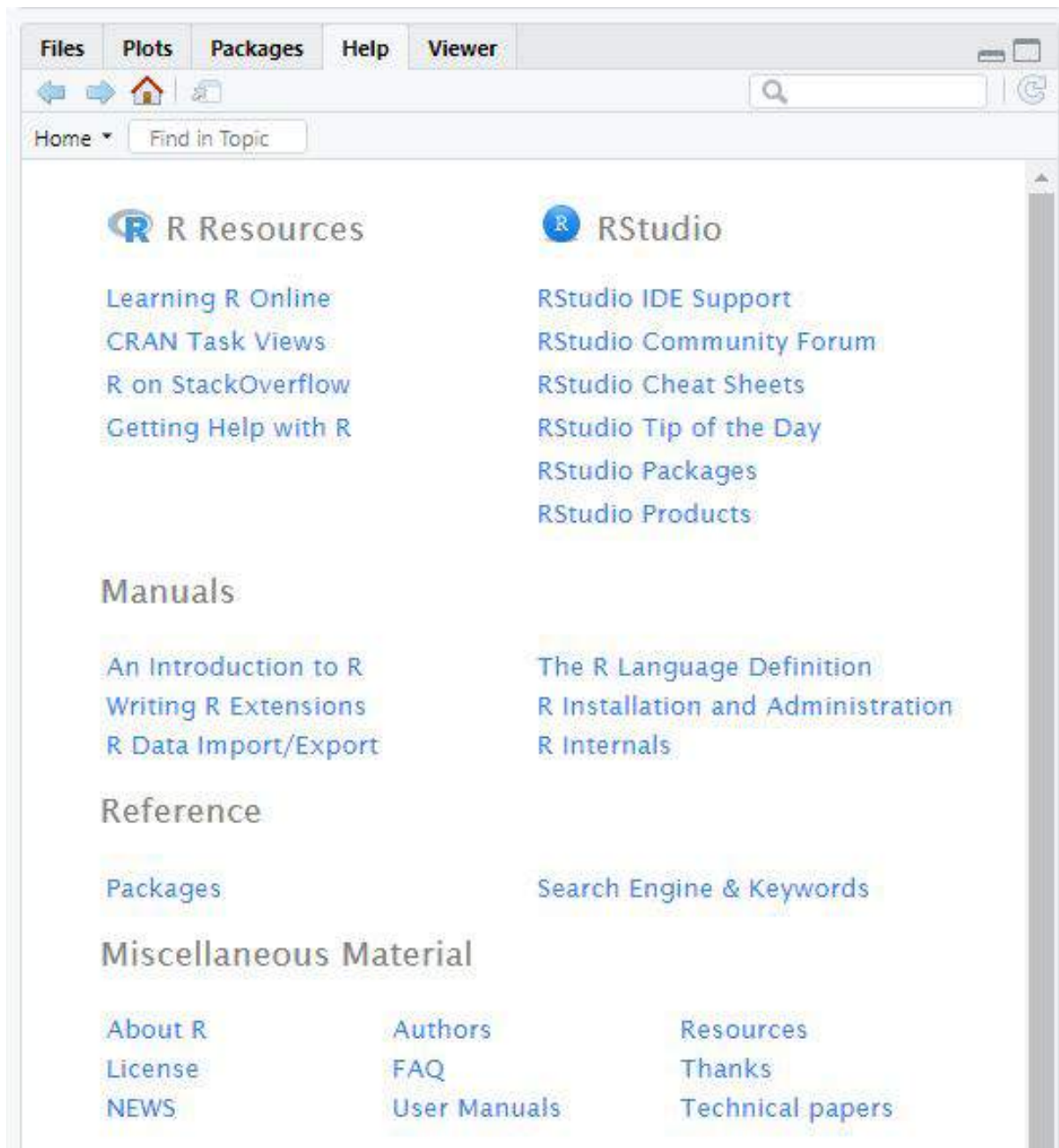


Figura 1.12: Pestaña ayuda

En la zona de búsqueda superior podremos buscar el nombre de la función directamente y en la zona inferior un tema o una palabra en concreto de alguna de las funciones.

### 1.3. Como guardar el trabajo

Para abrir el editor, vamos al icono superior izquierdo y seleccionamos nuevo Script.

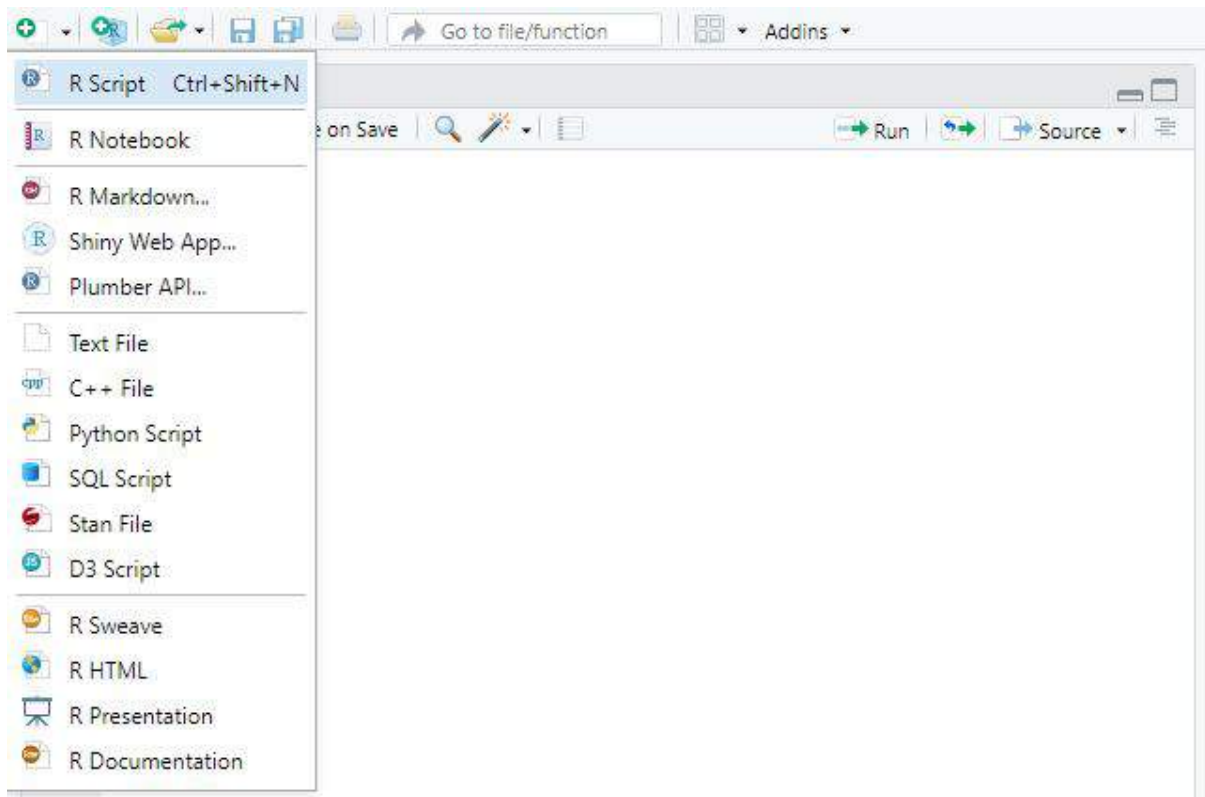


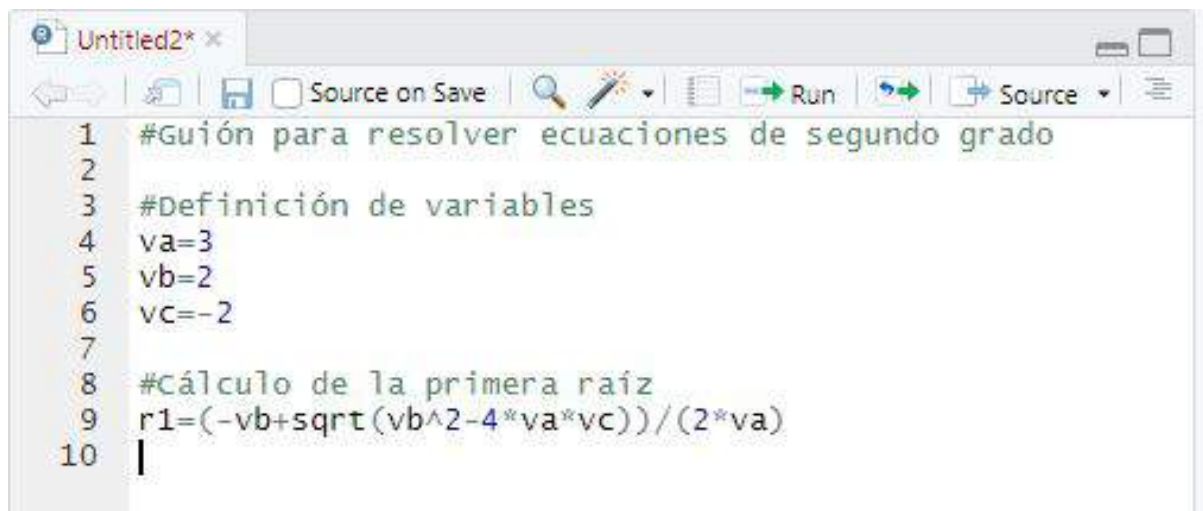
Figura 1.13: Editor RStudio

Vamos a poner un programa sencillo que nos haga la raíz cuadrada, para ello, vamos a poner mediante comentarios los distintos pasos que hacemos. Para ello, utilizamos la almohadilla (`#`), todo el texto que pongamos después de ella será comentado y no se ejecutará. El programa sencillo a ejecutar es el siguiente:

Listing 1.1: Código programa a ejecutar

```
1 #Guión para resolver ecuaciones de segundo grado
2
3 #Definición de variables
4 va=3
5 vb=2
6 vc=-2
7
8 #Cálculo de la primera raíz
9 r1=(-vb+sqrt(vb^2-4*va*vc))/(2*va)
```

Este programa lo veremos en el editor de la siguiente manera:



```
1 #Guión para resolver ecuaciones de segundo grado
2
3 #Definición de variables
4 va=3
5 vb=2
6 vc=-2
7
8 #Cálculo de la primera raíz
9 r1=(-vb+sqrt(vb^2-4*va*vc))/(2*va)
10 |
```

Figura 1.14: Programa sencillo a ejecutar

Tenemos varias opciones, darle al botón de Run para que se ejecute nuestro programa o podremos guardarlo seleccionando el icono del disco. Una vez seleccionado, como hay acentos en nuestro programa seleccionamos la codificación UTF-8, elegimos el nombre del archivo y nos lo guardará como un [Programa.R](#).

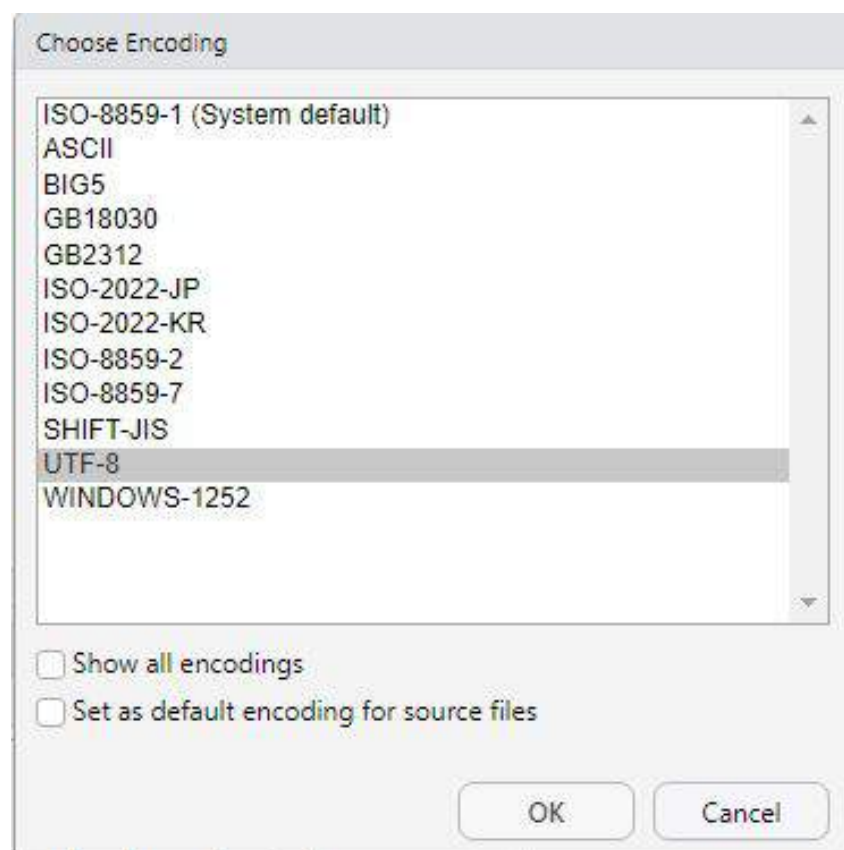
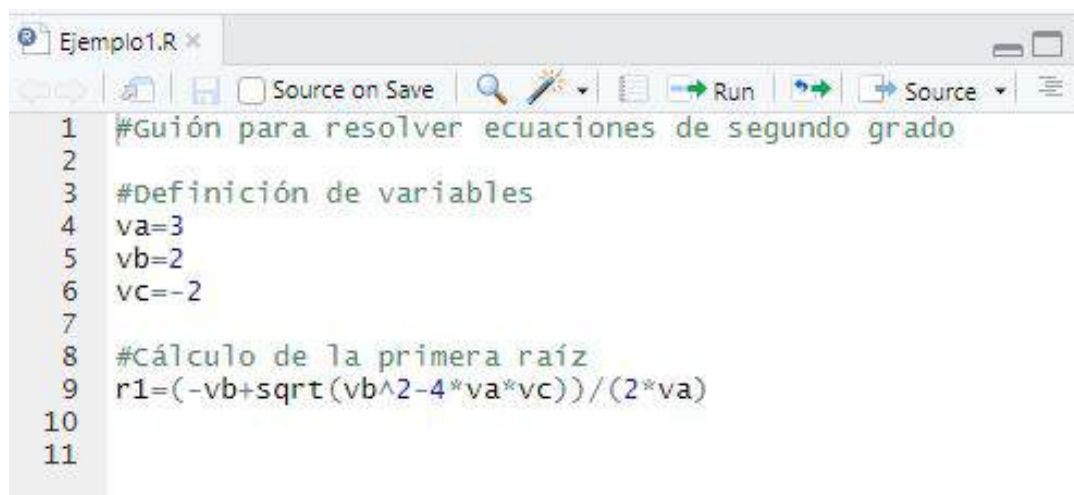


Figura 1.15: Codificación programa

Una vez que hemos guardado el programa, cuando volvamos a ejecutar RStudio nos encontraremos lo siguiente en el Editor:



```
1 #Guión para resolver ecuaciones de segundo grado
2
3 #Definición de variables
4 va=3
5 vb=2
6 vc=-2
7
8 #Cálculo de la primera raíz
9 r1=(-vb+sqrt(vb^2-4*va*vc))/(2*va)
10
11
```

Figura 1.16: Nombre programa

Además, no solo nos permite salvar editor, también nos permite salvar las variables en el Workspace. Cuando salgamos del RStudio nos indicará si queremos salvar el Workspace, le damos que si y al volvernos a meter tendremos nuestras variables guardadas.

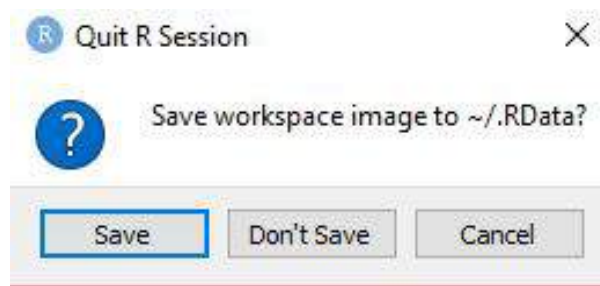


Figura 1.17: Salvar Workspace

En el caso del salvar el Workspace de este programa, al volver a entrar nos encontraremos lo siguiente:

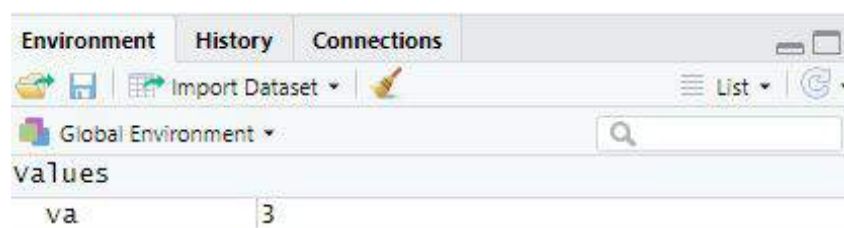


Figura 1.18: Variables salvadas

## 1.4. Paquetes

Una de las mejoras características de R es que podemos ampliarlo mediante el uso de paquetes. En este capítulo vamos a ver como instalar y guardar paquetes.

Para instalarlo primero nos tendremos que bajar el paquete de algún repositorio y lo segundo que tendremos que hacer es cargar el paquete. Para cargar e instalar paquetes utilizaremos la pestaña superior derecha, con nombre [Packages](#).

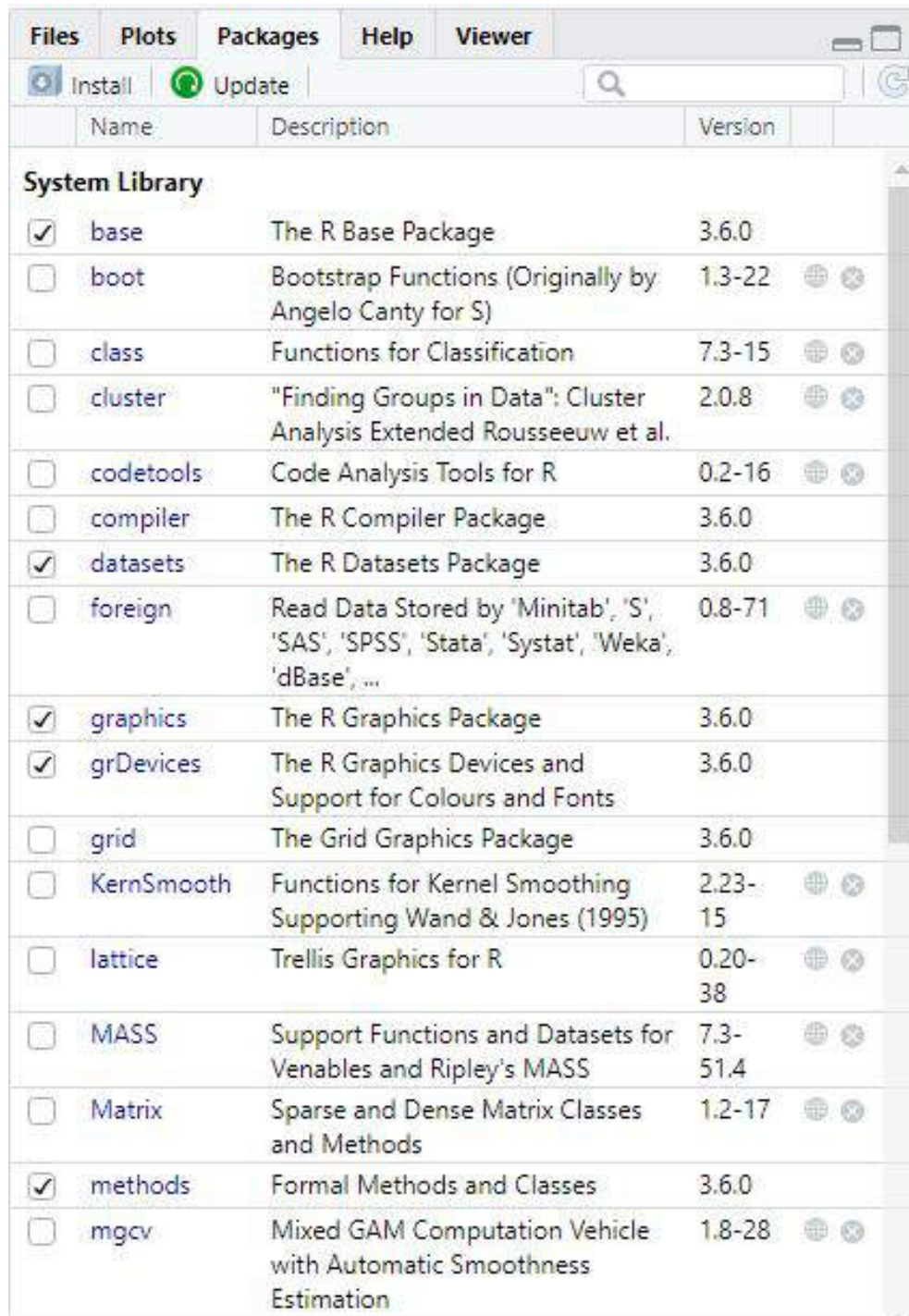


Figura 1.19: Paquetes pendientes de cargar

Por otro lado, hemos encontrado un paquete que nos interesa instalar. Este paquete se

llama fun y para descargarlo e instalarlo vamos a seguir los siguientes pasos:

- Seleccione Install.
- Le indico que busque en el repositorio CRAN.
- Le indico que intale todas las dependencias y que lo instale.

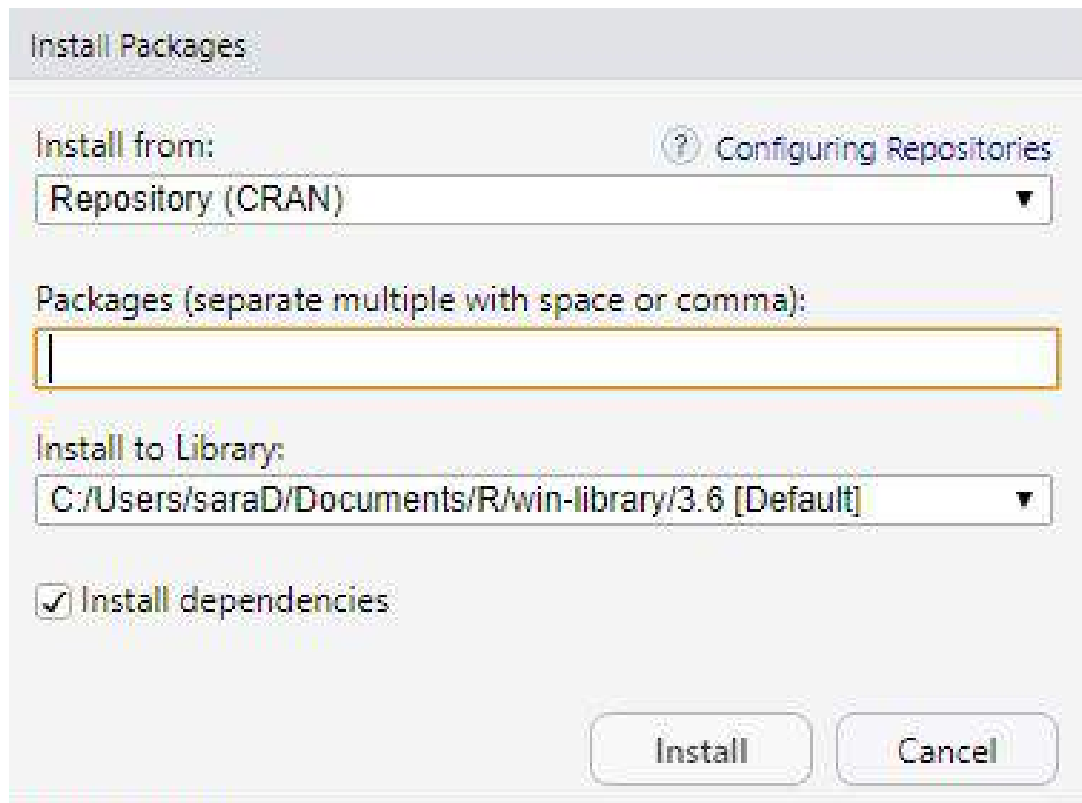


Figura 1.20: Instalar paquetes

Una vez hemos realizado estos pasos, se nos mostrará lo siguiente en la ventana de comandos y en la de los paquetes.

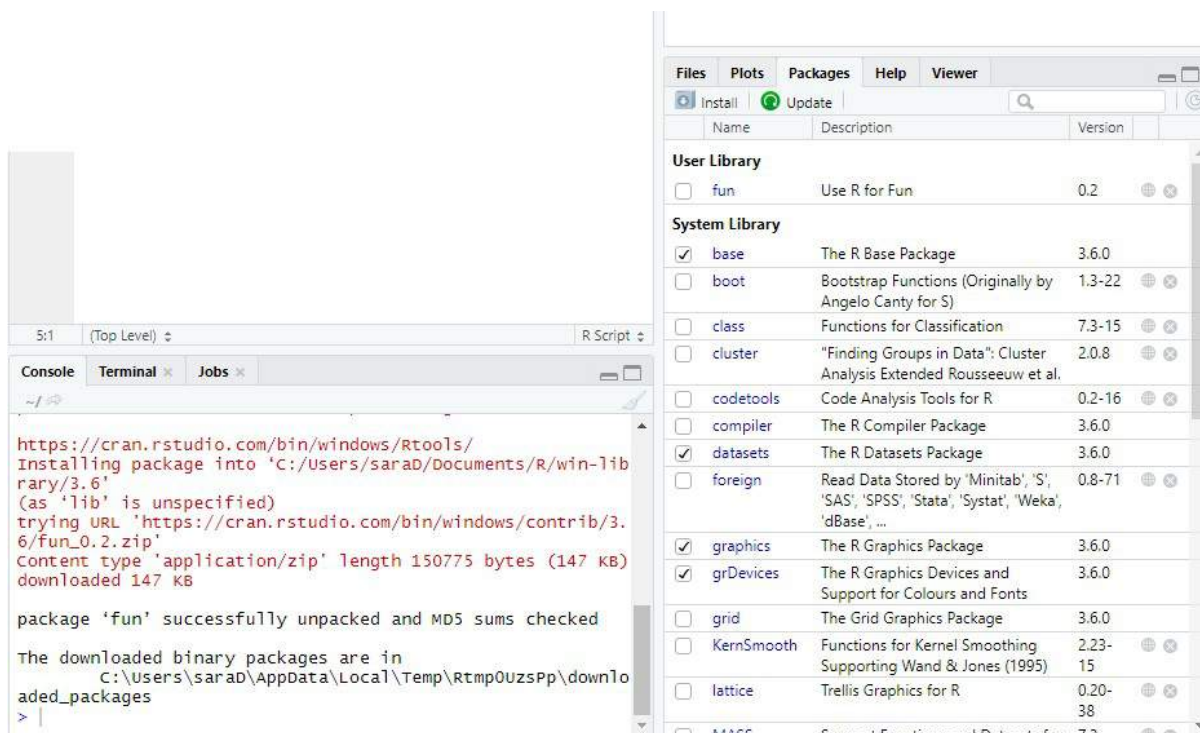


Figura 1.21: Instalar paquetes- Consola

Para cargar el paquete tenemos dos opciones, pinchar al cuadrado de los paquetes para que le ponga un tick o utilizar el comando `library("Paquete a cargar")`, en nuestro caso sería el paquete `fun`.

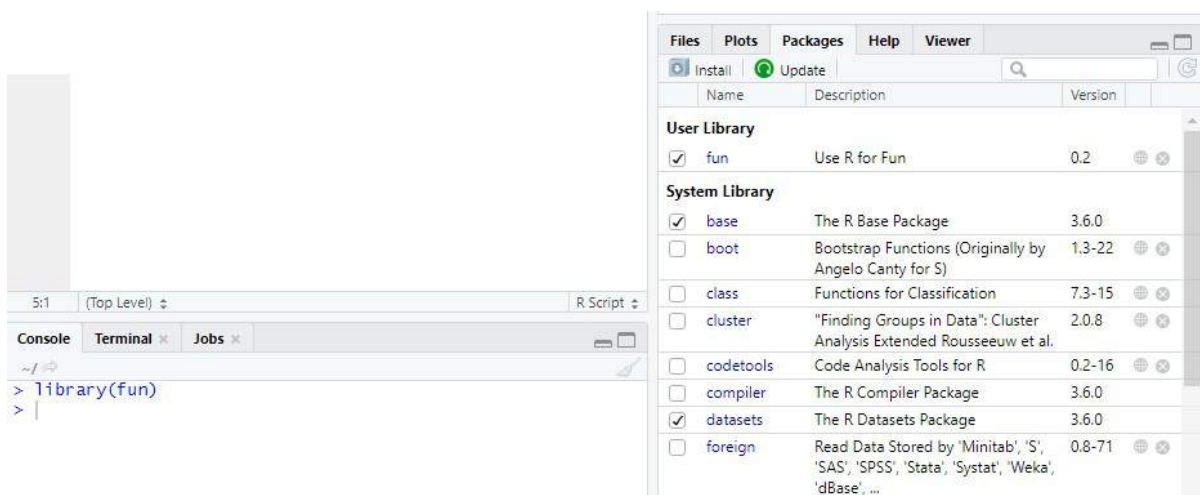


Figura 1.22: Paquete ya cargado

Finalmente, con esto tendremos ya instalado y cargado nuestro paquete, y podremos utilizar las funciones que tiene instaladas.





# Capítulo 2

## Matemáticas básicas

### 2.1. Operaciones y funciones reales básicas

Es importante recordar que podemos utilizar R como una calculadora, en ella podemos sumar resta y multiplicar. También es importante saber que los decimales se separan por puntos, no por comas. Una vez dicho esto, podemos realizar una suma muy simple en la ventana de comandos, para ello solo tenemos que poner la suma y darle a [Enter](#) y nos saldrá el resultado en la misma ventana de comandos.



```
Console Terminal x Jobs x
~/
> 2+3
[1] 5
> |
```

Figura 2.1: Suma básica

Vista esta operación básica, vamos a poner en una tabla las operaciones con sus comandos:

Cuadro 2.1: Tabla de operaciones básicas con comandos

Operación	<i>Suma</i>	<i>Resta</i>	<i>Multiplicación</i>	<i>División</i>	<i>Potencia</i>	<i>Cociente entero</i>	<i>Resto</i>
Comando	+	-	*	/	^	%/%	%%

Para los paréntesis tenemos la misma prioridad que en las operaciones básicas vistas en otros cursos como en el caso de Octave. No es lo mismo una operación realizada con paréntesis que sin él, vamos a poner un ejemplo sencillo donde el paréntesis varía mucho el resultado de la operación:

```

> 5^2*3
[1] 75
> 5^(2*3)
[1] 15625

```

Figura 2.2: Operación con y sin paréntesis

Por otro el número pi se indica con el comando `pi` y se opera como un número más. Seguidamente, vamos a ver las funciones numéricas más comunes con su respectivo comando:

Cuadro 2.2: Funciones básicas

Función	Comando	Función	Comando
$\sqrt{x}$	<code>sqrt</code>	$e^x$	<code>exp</code>
$\ln(x)$	<code>log</code>	$\log_{10}(x)$	<code>log10</code>
$\log_a(x)$	<code>log(,a)</code>	$\sin(x)$	<code>sin</code>
$\cos(x)$	<code>cos</code>	$\tan(x)$	<code>tan</code>
$\arcsin(x)$	<code>asin</code>	$\arccos(x)$	<code>acos</code>
$\arctan(x)$	<code>atan</code>	$ x $	<code>abs</code>
$n!$	<code>factorial</code>	$\binom{n}{m}$	<code>choose</code>

Vamos a ver algunos ejemplos de estas operaciones:

Listing 2.1: Código operaciones básicas

```

1 > 2+3
2 [1] 5
3 > 5^2*3
4 [1] 75
5 > 5^(2*3)
6 [1] 15625
7 > sqrt(2)
8 [1] 1.414214
9 > exp(2)
10 [1] 7.389056
11 > log10(100)
12 [1] 2
13 > log(100)
14 [1] 4.60517
15 > log(100, pi)
16 [1] 4.022932
17 > exp(1)
18 [1] 2.718282

```

También es muy importante tener en cuenta de que los argumentos de R en las funciones trigonométricas han de estar en radianes y los resultados de las funciones trigonométricas inversas también se dan en radianes. Por lo que, tendremos que tener en cuenta la relación entre grados y radianes según lo que estemos operando.

#### Relación entre radianes y grados:

$$x \text{ grados} = \frac{x \cdot \pi}{180} \text{ radianes}$$

$$x \text{ radianes} = \frac{180x}{\pi} \text{ grados}$$

Finalmente, vamos a ver un ejemplo de cómo cambia realizar un coseno de 60 radianes respecto de 60°.

Listing 2.2: Código de operaciones en grados y radianes

```
1 > cos(60)
2 [1] -0.952413
3 > cos(60*pi/180)
4 [1] 0.5
```

## 2.2. Cifras significativas y redondeo

R utiliza la notificación científica para expresar números muy grandes o muy pequeños mediante la base 10 como podemos ver en el siguiente ejemplo:

Listing 2.3: Notación científica por defecto de R

```
1 > 2^50
2 [1] 1.1259e+15
3 > 2^(-50)
4 [1] 8.881784e-16
```

Si queremos conocer más cifras significativas de un número utilizaremos la función `print(x,n)`, donde  $n$  son las cifras significativas que queremos mostrar. Vamos a ver un ejemplo con esta misma función:

Listing 2.4: Código para cifras significativas

```
1 > sqrt(5)
2 [1] 2.236068
3 > print(sqrt(5), 14)
4 [1] 2.2360679774998
```

**IMPORTANTE:** R sólo trabaja con 16 cifras significativas. Pedir que una función haga más de esta nos puede llevar a errores.

Por otro lado, para redondear un número a  $n$  cifras decimales utilizaremos la función `round(x,n)`. Para ver cómo funciona vamos a poner un ejemplo:

Listing 2.5: Código función round

```
1 > round(exp(3),4)
2 [1] 20.0855
3 > round(exp(3),2)
4 [1] 20.09
5 > round(exp(3))
6 [1] 20
```

Si no le indicamos a cuántas cifras queremos redondear, redondeará al número entero más cercano.

Por otro lado, tenemos la función `floor(x)` que redondea a un entero por defecto, la función `ceiling(x)` que redondea a un entero por exceso y con la función `trunc(x)` que trunca el número a su parte entera. Vamos a ver un ejemplo redondeado el número pi de cada una de estas funciones para ver cómo varían:

Listing 2.6: Código funciones redondeo

```
1 > pi
2 [1] 3.141593
3 > round(pi)
4 [1] 3
5 > floor(pi)
6 [1] 3
7 > ceiling(pi)
8 [1] 4
9 > trunc(pi)
10 [1] 3
```

## 2.3. Definición de variables y funciones

Una variable es un objeto que sirve para guardar datos, y para asignarle un valor podemos utilizar lo siguiente:

- `variable = valor.`
- `variable <- valor.`
- `valor -> variable.`

Primero vamos a definir una variable, la llamaremos *a* y que sea igual a la  $\sqrt{5}$ . Como podemos ver en el momento de realizarla, R no nos da el valor, pero si indicamos el nombre de la variable nos mostrará su valor.

Listing 2.7: Código de definición de variable

```
1 > a=sqrt(5)
2 > a
3 [1] 2.236068
```

Una vez que hemos definido la variable podemos realizar todo tipo de operaciones con ella. Estas variables pueden cambiar de valor a lo largo de la sesión si las cambiamos el número, por ejemplo, vamos a cambiar el valor a esta variable y vamos a realizar una operación sencilla con esta:

Listing 2.8: Código de cambio de valor de variable

```
1 > a=sqrt(5)
2 > a
3 [1] 2.236068
4 > a=20
5 > a
6 [1] 20
7 > a/5
8 [1] 4
```

Seguidamente, para definir una función utilizaremos la construcción **nombre de la función=function(variable){definición}**. Como este proceso es muy abstracto vamos a definir una función sencilla:

Listing 2.9: Código de función sencilla

```
1 > f=function(a){a^pi}
2 > f(2)
3 [1] 8.824978
4 > f(a)
5 [1] 12226.62
```

No solo podremos definir una función de un solo valor, podremos definir más de una variable en la misma función separándola entre comas, por ejemplo, vamos a añadir una función *g* con dos variables:

Listing 2.10: Código de función de dos variables

```
1 > g=function(a,b){(a-b)^pi}
2 > g(5,2)
3 [1] 31.54428
4 > g(0,1)
5 [1] NaN
```

Como podemos ver nos ha devuelto un valor **NaN**, lo que significa que no existe un valor entero.

También podemos definir la misma función de dos maneras distintas separando las definiciones entre punto y coma, para verlo vamos a definir una función *h*:

Listing 2.11: Código de función con dos definiciones

```
1 > h=function(a){b=sqrt(a); g(a,b)}
2 > h(5)
3 [1] 24.38367
```

Posteriormente, los nombres de las variables y las funciones pueden contener letras, números, símbolos, pero es muy importante que comiencen con una letra o un punto. También se puede borrar algo de la memoria mediante el comando `rm()`, con este comando podremos borrar alguna variable o función que no nos guste.

Listing 2.12: Código borrar

```
1 > rm(f)
```

Una vez que hemos hecho todo este proceso nuestro entorno debería quedarnos así:

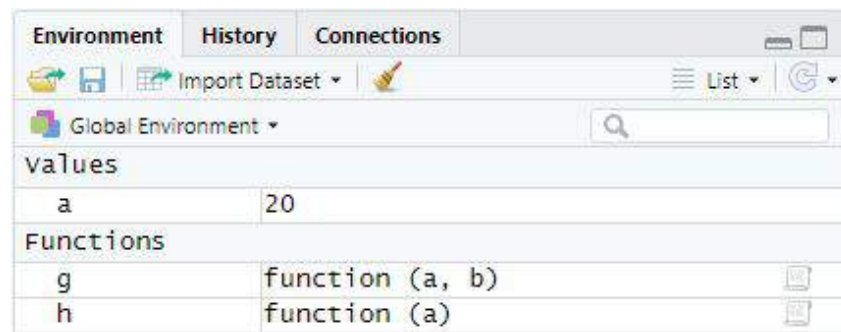


Figura 2.3: Entorno funciones

Y terminando, si queremos borrar todo el entorno sólo tendremos que pulsar el icono de la escoba y nos lo borrará.

## 2.4. Números complejos

Los números complejos se pueden escribir de dos formas, para ello tendremos que tener en mente la forma de los números complejos  $a + bi$ :

- Si  $b$  es racional escribiremos  $a+bi$ .
- En otros casos escribiremos `complex(real=a, imaginary=b)`. Y declaramos un número real como complejo con la función `as.complex`.

Ahora vamos a proceder a ver cómo lo tendríamos que definir en R:

Listing 2.13: Código números complejos

```
1 > 3+2.5 i
2 [1] 3+2.5 i
3 > 3+sqrt(2) i
4 Error: unexpected symbol in "3+sqrt(2) i"
5 > complex(real=3, imaginary=sqrt(2))
6 [1] 3+1.414214 i
7 > sqrt(-4)
8 [1] NaN
9 Warning message:
10 In sqrt(-4) : NaNs produced
```

```

11 > z=as.complex(-4)
12 > sqrt(z)
13 [1] 0+2i

```

Con este tipo de números podremos utilizar las mismas operaciones y funciones básicas que con los números reales. Vamos a ver unos ejemplos de operaciones con números complejos:

Listing 2.14: Código operaciones números complejos

```

1 > (2+3i)*(5+1i)
2 [1] 7+17i
3 > (2+3i)/(5+1i)
4 [1] 0.5+0.5i
5 > sqrt(2+3i)
6 [1] 1.674149+0.895977i
7 > sqrt(2+3i)^2
8 [1] 2+3i

```

Aparte de las funciones que tienen en común con los números reales, los números complejos tienen unas funciones específicas y son las siguientes:

$$z = a + bi$$

- Parte real e imaginario: Funciones [Re](#) e [Im](#).
- Conjugado: Función [Conj](#).
- Módulo: Función [Mod](#).
- Argumento: Función [Arg](#)

Con estas funciones estamos representando lo siguiente:

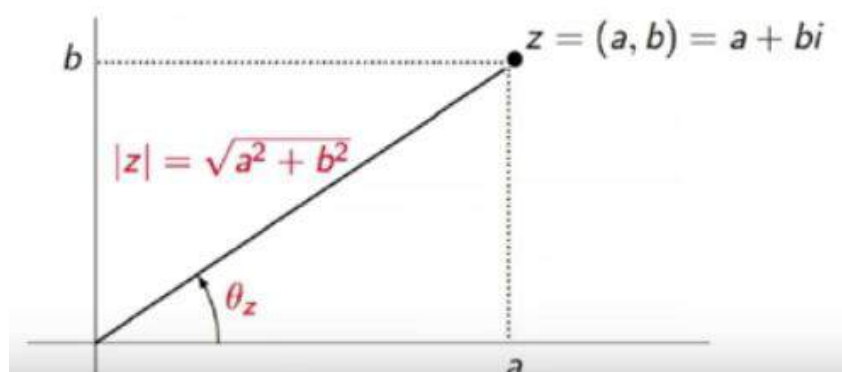


Figura 2.4: Funciones números complejos

Vamos a ver un ejemplo donde se calcule el conjugado, el módulo y el argumento de un número complejo:

Listing 2.15: Código del conjugado, módulo y argumento del número complejo

```
1 > z=2+3i
2 > Conj(z)
3 [1] 2-3i
4 > Mod(z)
5 [1] 3.605551
6 > Arg(z)
7 [1] 0.9827937
```

Es importante indicar que el argumento está en radianes, por lo que si queremos transformarlo a grados hay que seguir la transformación expuesta arriba.



# Capítulo 3

## Representación gráfica y lectura de datos

### 3.1. Data frames y plots

En este capítulo vamos a ver la representación estadística de una serie de puntos a través de una recta. Para ello, primero vamos a recoger los datos.

Listing 3.1: Código de recogida de datos en vectores

```
1 > edad=c(1,2,3,5,7,9,11,13)
2 > altura=c
   (76.11,86.45,95.27,109.18,122.03,133.73,143.73,156.41)
```

Como son muchos datos, vamos a crear una tabla de datos, que en R es un [data frame](#). Vamos a ver su estructura:

Listing 3.2: Código data.frame

```
1 > datos1=data.frame(edad, altura)
2 > datos1
3   edad altura
4 1     1  76.11
5 2     2  86.45
6 3     3  95.27
7 4     5 109.18
8 5     7 122.03
9 6     9 133.73
10 7    11 143.73
11 8    13 156.41
```

Como podemos ver la primera columna corresponde con la edad y la segunda columna corresponde a la altura. Podremos utilizar el símbolo `$` para acceder a los datos de una columna.

Listing 3.3: Código acceso datos columna edad

```
1 > datos1$edad
2 [1] 1 2 3 5 7 9 11 13
```

Ahora vamos a dibujar estos puntos, para ello vamos a utilizar el comando `plot(x,y)`. Es importante que  $x$  e  $y$  sean vectores de la misma longitud.

Listing 3.4: Código dibujar gráficas

```
1 > plot(datos1$edad, datos1$altura)
2 > plot(datos1)
```

Una vez hemos escrito cualquiera de los dos códigos veremos la siguiente gráfica:

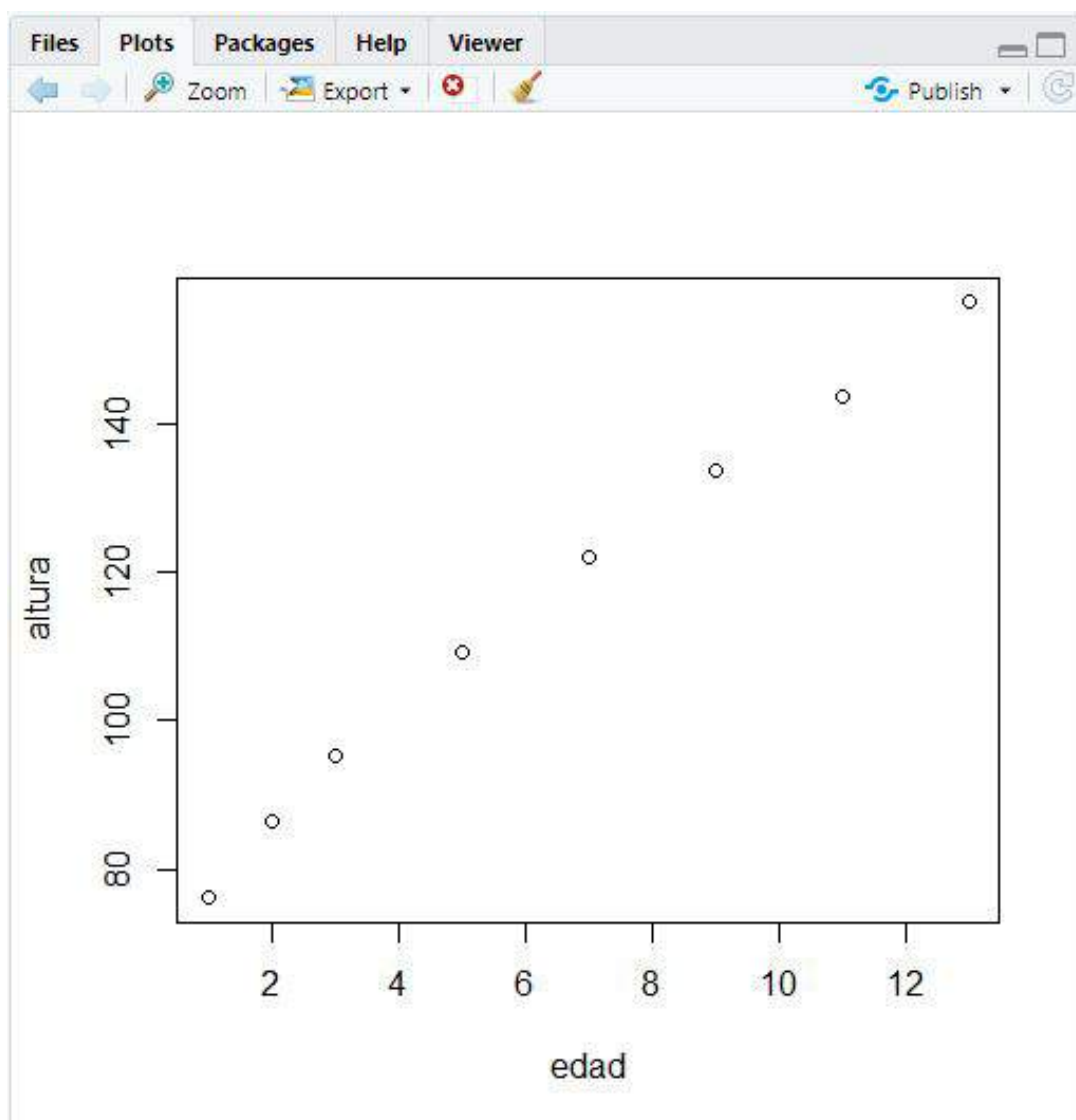


Figura 3.1: Representación gráfica

## 3.2. Regresión lineal

La recta de regresión es una recta que es la que mejor se adapta a los puntos. En este capítulo vamos a ver cómo se calcula y se realiza una recta de regresión.

Para calcular la recta de regresión utilizamos el comando `lm()`, que se utiliza para los métodos lineales. Para poder utilizarla haremos `lm(y~x)`, la tilde significa en función de, por lo que en este caso estaríamos indicando que realice una recta de regresión de la variable  $y$  en función de la variable  $x$ . Para tenerlo más claro, vamos a hacer el método lineal del ejemplo anterior:

Listing 3.5: Código método lineal

```

1 > edad=c(1,2,3,5,7,9,11,13)
2 > altura=c
  (76.11,86.45,95.27,109.18,122.03,133.73,143.73,156.41)
3 > datos1=data.frame(edad,altura)
4 > plot(datos1)
5 > regr_edad_altura=lm(altura~edad, data=datos1)
6 > lm(altura~edad, data=datos1)
7
8 Call:
9 lm(formula = altura ~ edad, data = datos1)
10
11 Coefficients:
12 (Intercept)      edad
13    73.968      6.493

```

En este caso nos está dando el valor de los coeficientes y con el intercept el valor de la ordenada en el origen. Si queremos más información sobre este resultado sólo tenemos que ejecutar el comando `summary(variable)` sobre la variable. En este caso nos daría los siguiente:

Listing 3.6: Código summary

```

1 > summary(regr_edad_altura)
2
3 Call:
4 lm(formula = altura ~ edad, data = datos1)
5
6 Residuals:
7   Min     1Q  Median     3Q    Max
8 -4.351 -1.743  0.408  2.018  2.745
9
10 Coefficients:
11             Estimate Std. Error t value
12 (Intercept)  73.9681     1.7979  41.14
13 edad         6.4934     0.2374  27.36
14             Pr(>|t|)
15 (Intercept) 1.38e-08 ***

```

```
16 edad          1.58e-07 ***
17 ---
18 Signif. codes:
19 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
20
21 Residual standard error: 2.746 on 6 degrees of freedom
22 Multiple R-squared:  0.992,    Adjusted R-squared:  0.9907
23 F-statistic: 748.4 on 1 and 6 DF,  p-value: 1.577e-07
```

En Coefficients podemos ver los distintos valores que toman la ordenada en el origen y la pendiente, y el Multiple R-squared es el coeficiente de ordenación  $R^2$ .

Finalmente, vamos a dibujar la recta de regresión sobre la gráfica que ya tenemos. Para ello, vamos a utilizar el comando `abline(variable)` que dibuja rectas.

Listing 3.7: Código cálculo recta de regresión completo

```
1 > edad=c(1,2,3,5,7,9,11,13)
2 > altura=c
  (76.11,86.45,95.27,109.18,122.03,133.73,143.73,156.41)
3 > datos1=data.frame(edad,altura)
4 > plot(datos1)
5 > regr_edad_altura=lm(altura~edad, data=datos1)
6 > lm(altura~edad, data=datos1)
7 > abline(regr_edad_altura, col="red")
```

Con este código obtendremos la siguiente gráfica:

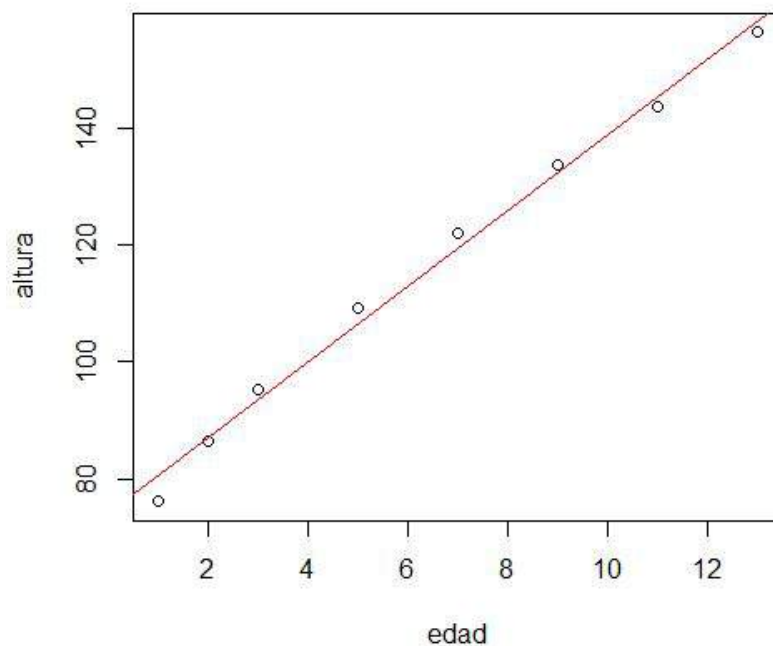


Figura 3.2: Recta regresión ejemplo

### 3.3. Formas básicas de lectura de datos externos

R nos permite compartir los datos vía internet, para que se pueda acceder a los mismos datos sin necesidad de cambiarlos de sitio y de una manera más cómoda. Estos datos los podremos importar a R vía [Import Dataset](#) en la pestaña del entorno. Haciendo clic sobre él nos da la opción de importarlo desde un fichero o desde una página web.

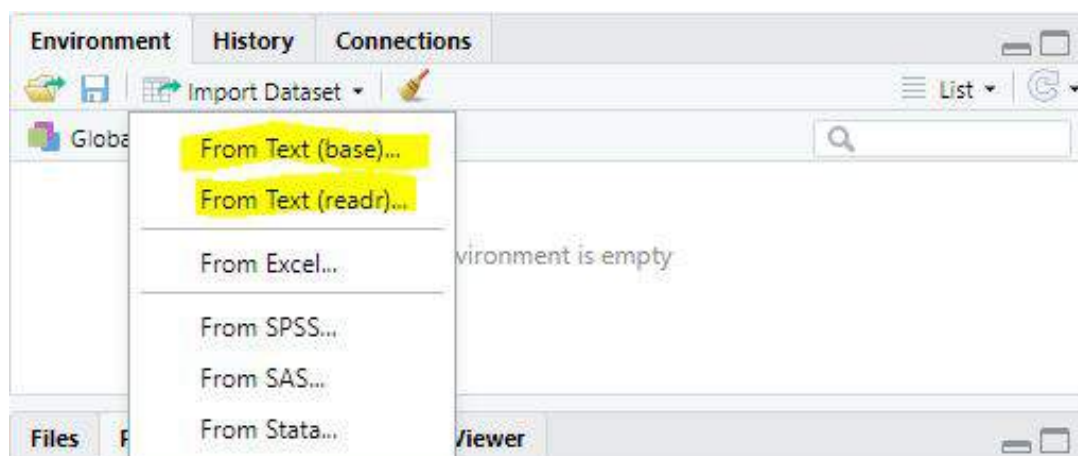


Figura 3.3: Importar datos

Ahora vamos a escoger subir los datos desde una web. La web escogida es <http://aprender.uib.es/Rdir/pearson.txt>, donde tendremos una serie de datos sobre padres e hijos.

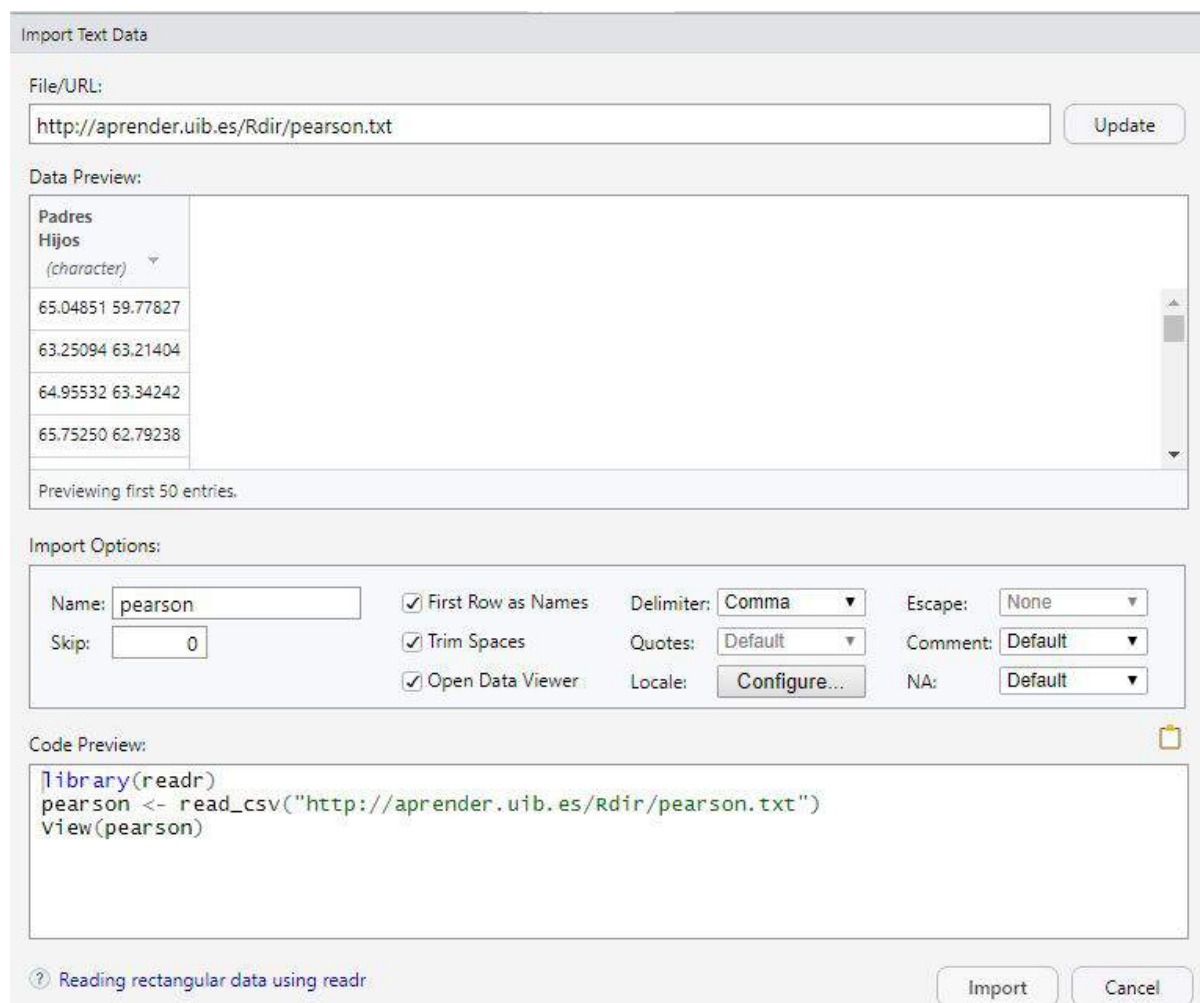


Figura 3.4: Importar datos web

En esta pestaña le indicamos que nos lo separe por un espacio en blanco y denominamos a nuestros datos como `df_pearson`. Y veremos nuestros datos así:

	Padres	Hijos
1	65.04851	59.77827
2	63.25094	63.21404
3	64.95532	63.34242
4	65.75250	62.79238
5	61.13723	64.26113
6	63.02254	64.24221
7	65.37053	64.08231
8	64.72398	63.99574
9	66.06509	64.61338
10	66.96738	63.97944
11	59.00800	65.24451
12	62.93203	65.35102
13	63.67063	65.67992

Figura 3.5: Datos ya importados

Otra manera de importar los datos es mediante el comando `read.table(nombre del fichero o URL, header=FALSE, sep=" ", dec=".")`. Primero colocamos el nombre de fichero o la dirección, lo segundo es el encabezado que en este caso no tiene, lo tercero es la separación que son los espacios en blanco y lo último los decimales los separamos por un punto. Vamos a importar los mismos datos con este método:

Listing 3.8: Código importar datos

```
1 > df_pearson2=read.table("http://aprender.uib.es/Rdir/pearson.txt", header = TRUE)
2 > View(df_pearson2)
```

Finalmente, vamos a hacer una recta de regresión con estos nuevos datos. Para ello, vamos a utilizar este nuevo código:

Listing 3.9: Código recta regresión

```
1 > plot(df_pearson)
2 > regHijosPadres=lm(Hijos~Padres, data=df_pearson)
3 > abline(regHijosPadres, col="red")
4 > summary(regHijosPadres)
5
6 Call:
7 lm(formula = Hijos ~ Padres, data = df_pearson)
8
9 Residuals:
10      Min       1Q   Median       3Q      Max
11 -8.8772 -1.5144 -0.0079  1.6285  8.9685
12
13 Coefficients:
14             Estimate Std. Error t value
15 (Intercept) 33.88660    1.83235   18.49
16 Padres       0.51409    0.02705   19.01
17             Pr(>|t|)
18 (Intercept) <2e-16 ***
19 Padres      <2e-16 ***
20 ---
21 Signif. codes:
22 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
23
24 Residual standard error: 2.437 on 1076 degrees of freedom
25 Multiple R-squared:  0.2513, Adjusted R-squared:  0.2506
26 F-statistic: 361.2 on 1 and 1076 DF, p-value: < 2.2e-16
```

Y este código nos dará la siguiente gráfica de regresión:

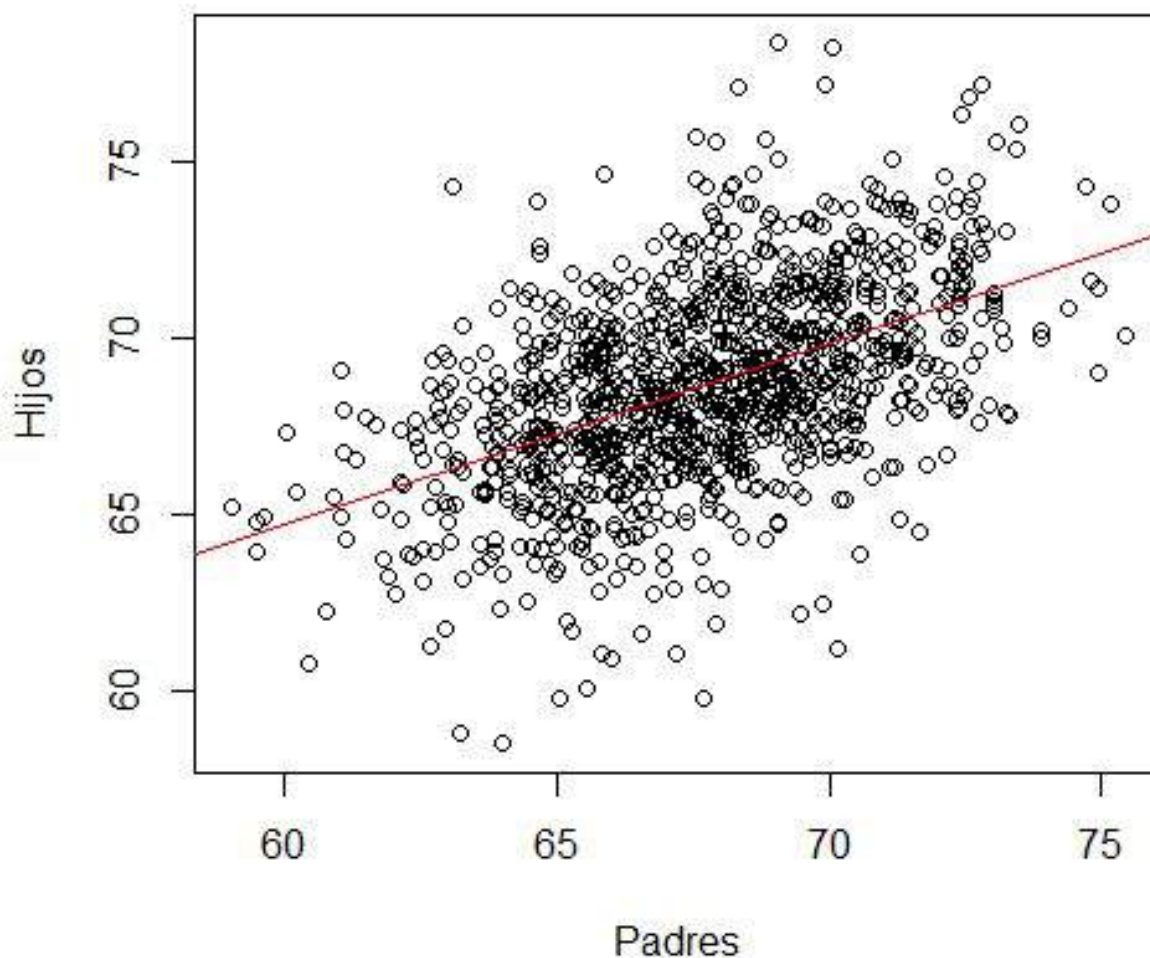


Figura 3.6: Regresión nueva

### 3.4. Rectas de regresión y transformaciones logarítmicas

En este capítulo vamos a ver cómo representar funciones logarítmicas o exponenciales. Para ello, primero vamos a ver cómo transformar una función exponencial en una lineal.

$$y = a \cdot 10^{bx}$$

$$\log y = \log(a \cdot 10^{bx}) = \log a + bx \log 10 = \log a + bx$$

Seguidamente, vamos a ver cómo se transforma una función exponencial en una lineal.

$$y = a \cdot x^b$$

$$\log y = \log(a \cdot x^b) = \log a + b \cdot \log x$$

Vamos a poner un ejemplo de una función logarítmica que representa la relación entre la inhibición y la serotonina. Primero vamos a ver su código:



Listing 3.10: Código función logarítmica

```
1 > inhibicion=c(19,36,60,84)
2 > serotonina=c(1.2,3.6,12,33)
3 > df_IS=data.frame(inhibicion , serotonina)
4 > plot(df_IS)
```

Este código nos dará como resultado una representación logarítmica que es la siguiente:

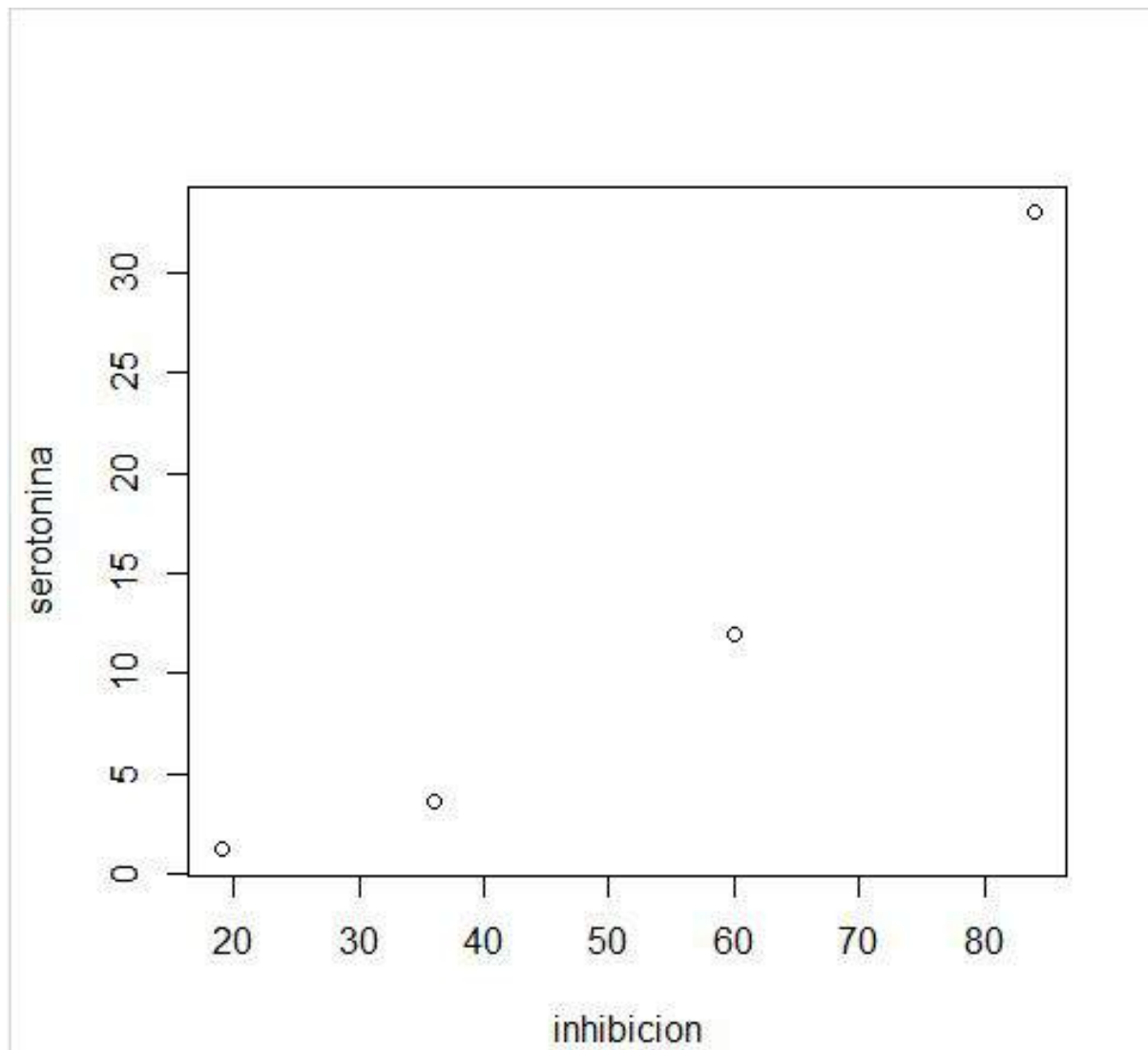


Figura 3.7: Gráfica función logarítmica

Pero si hacemos el logaritmo en base 10 de la serotonina nos dará una gráfica más lineal.

Listing 3.11: Código representación más lineal

```
1 > plot(inhibicion , log10(serotonina))
```

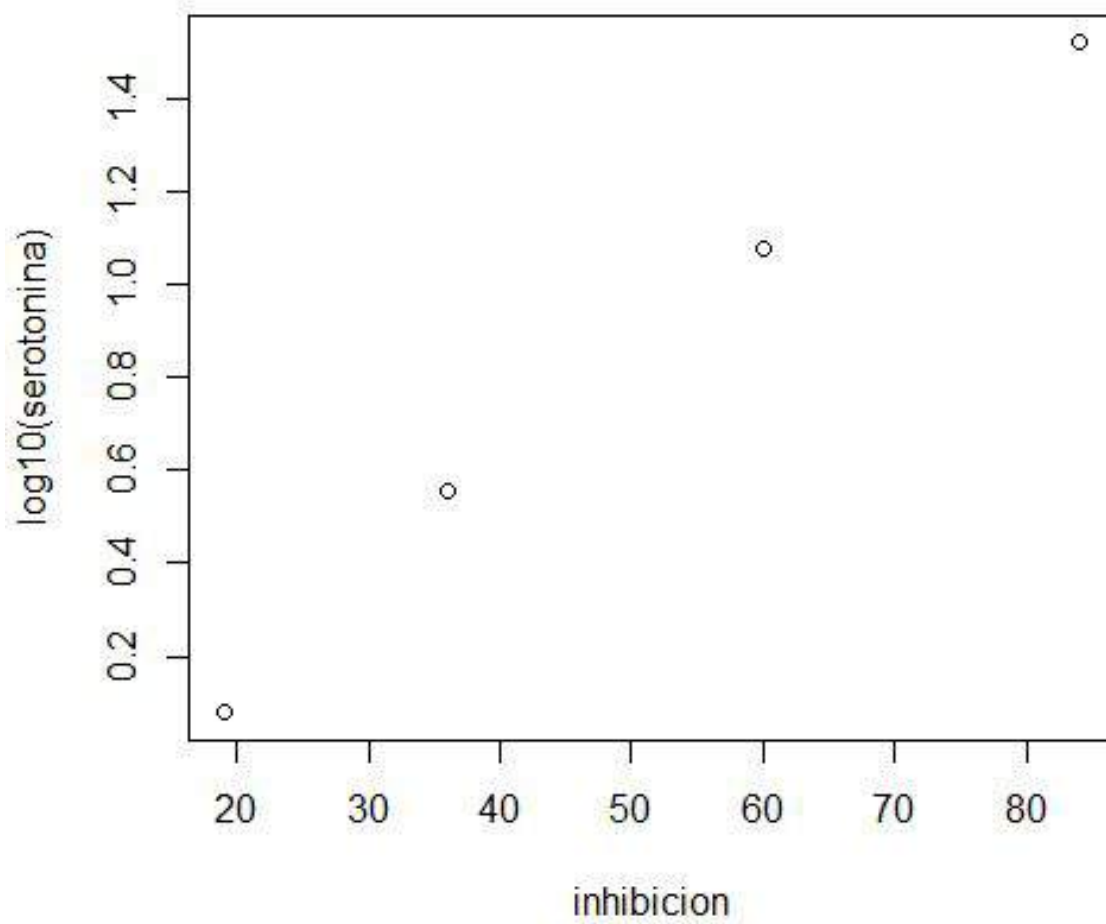


Figura 3.8: Representación más lineal

Esta gráfica la podemos hacer de otra manera, en el mismo plot sólo tendremos que indicar cual es la parte logarítmica de la función.

Listing 3.12: Código representación más lineal

```
1 > plot(df_IS, log="y")
```

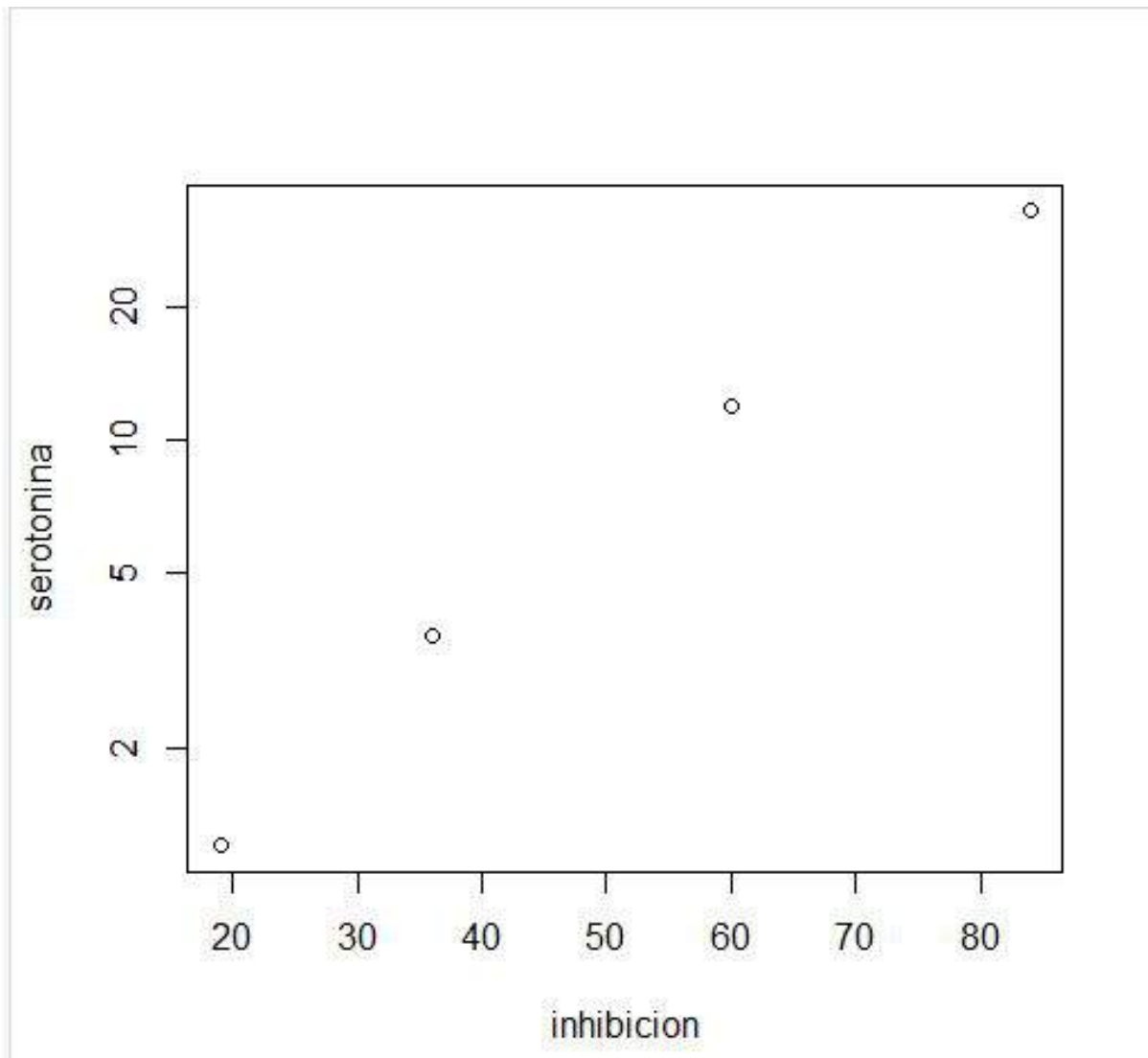


Figura 3.9: Representación semilogarítmica

Con el código anterior obtendremos una gráfica en escala semilogarítmica.

Como hemos obtenido estos resultados, que son más lineales, podremos realizar una regresión lineal de ellos. Vamos a proceder con el siguiente código:

Listing 3.13: Código de regresión lineal

```

1 > inhibicion=c(19,36,60,84)
2 > serotonina=c(1.2,3.6,12,33)
3 > df_IS=data.frame(inhibicion,serotonina)
4 > plot(df_IS)
5 > plot(inhibicion,log10(serotonina))
6 > plot(df_IS, log="y")
7 > regr_inh_logSer=lm(log10(serotonina)~inhibicion, data=df_IS)
8 > summary(regr_inh_logSer)
9
10 Call:
11 lm(formula = log10(serotonina) ~ inhibicion, data = df_IS)

```

```

12
13 Residuals :
14      1      2      3      4
15 -0.05381  0.04997  0.04578 -0.04195
16
17 Coefficients :
18             Estimate Std. Error t value
19 (Intercept) -0.284272  0.076811  -3.701
20 inhibicion  0.021961  0.001384  15.863
21             Pr(>|t|)
22 (Intercept)  0.06588 .
23 inhibicion  0.00395 **
24 ---
25 Signif. codes:
26 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
27
28 Residual standard error: 0.068 on 2 degrees of freedom
29 Multiple R-squared:  0.9921, Adjusted R-squared:  0.9882
30 F-statistic: 251.6 on 1 and 2 DF, p-value: 0.00395

```

Ahora vamos a reconvertir estos parámetros.

$$\log a = -0,284272 \rightarrow a = 10^{-0,284272} = 0,5196708$$

$$y = 0,5197 \cdot 10^{0,02196 \cdot x}$$

Esta reconversión la pasaremos a la curva para ver si se adaptan bien los datos de la regresión lineal. El código sería el siguiente:

Listing 3.14: Código curva

```

1 > plot(df_IS)
2 > curve(0.5196704*(10^0.021961)^x, add=TRUE, col="blue")

```

Que nos dará la siguiente gráfica:

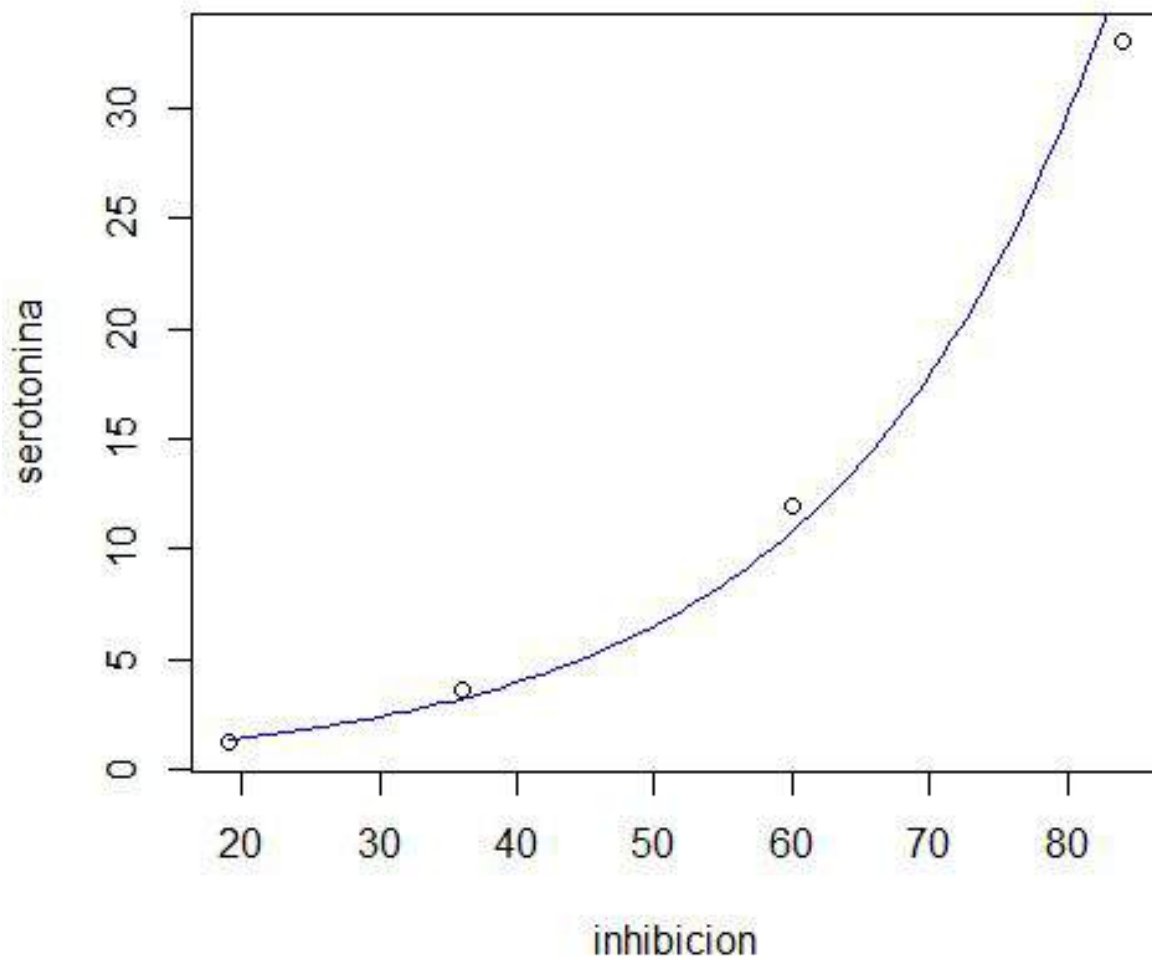


Figura 3.10: Representación de la curva

Seguidamente, vamos a ver cómo se realiza una función exponencial.

Listing 3.15: Código función exponencial

```
1 > tiempo=1:12
2 > SIDA_acum=c(97,709,2698,6928,15242,29944,52902,83903,120612,
3 161711,206247,257085)
4 > df_SIDA=data.frame(tiempo, SIDA_acum)
5 > plot(df_SIDA)
```

Este código nos da como resultado la siguiente gráfica:

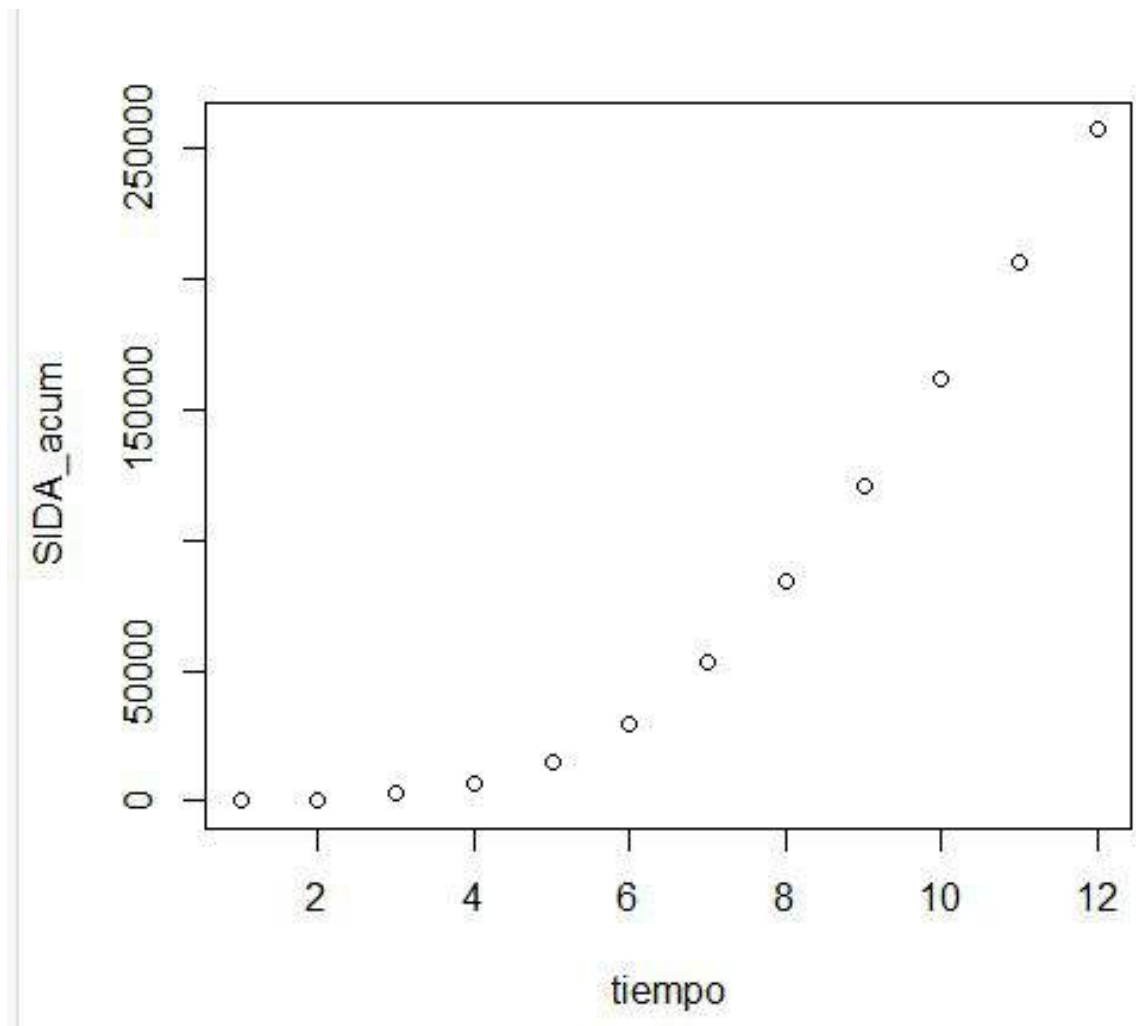


Figura 3.11: Función exponencial

Para hacerla más lineal, la tendremos que pasar dos veces por una logarítmica mediante este sencillo código:

Listing 3.16: Código recta más lineal

```
1 > plot(df_SIDA, log="xy")
```

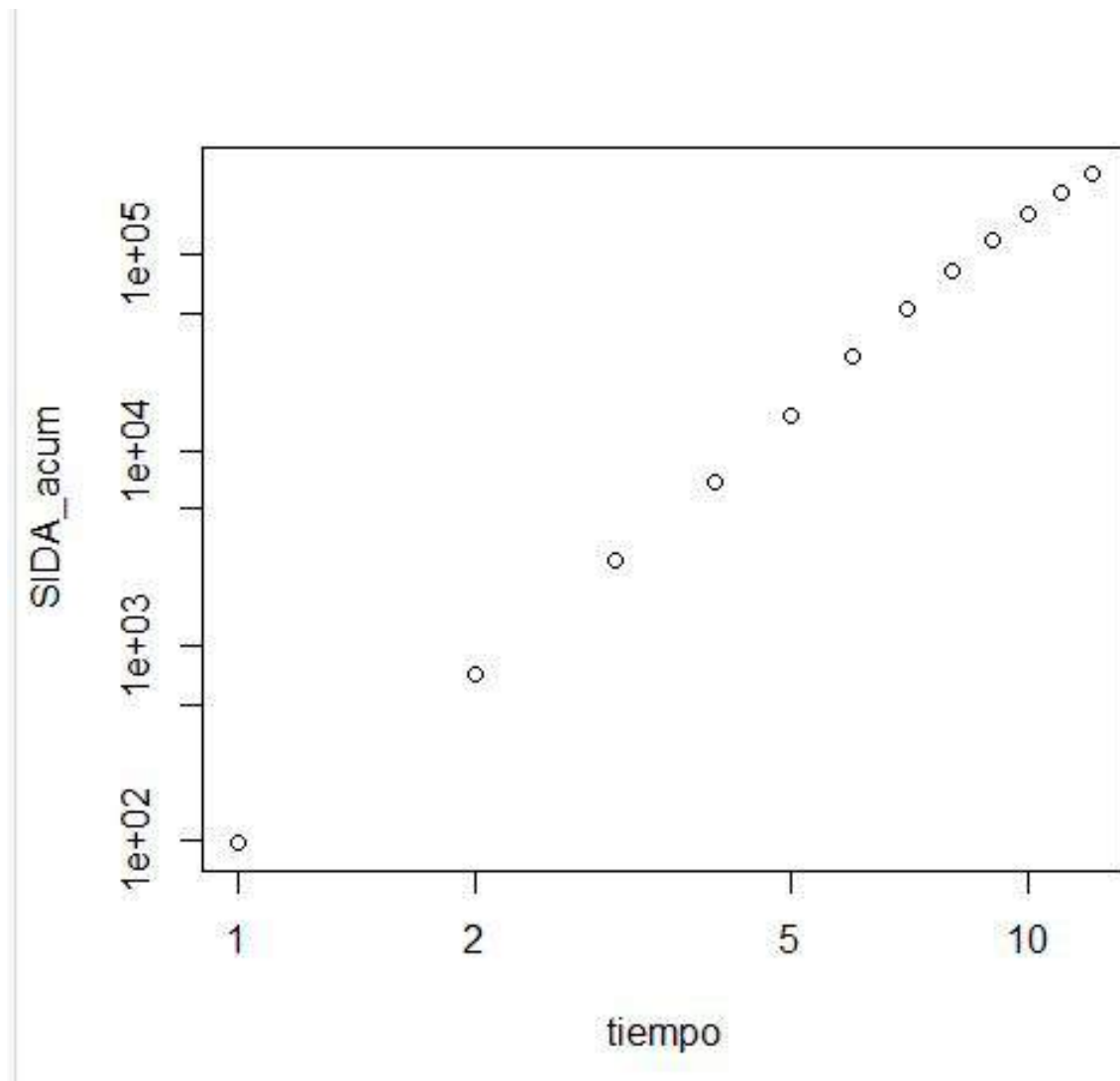


Figura 3.12: Representación más lineal

Con esta gráfica obtenida vamos a realizar la regresión lineal de la misma.

Listing 3.17: Código regresión lineal

```

1 > regr_logt_logSIDA=lm(log10(SIDA_acum)~log10(tiempo), data=df
  _SIDA)
2 > summary(regr_logt_logSIDA)
3
4 Call :
5 lm(formula = log10(SIDA_acum) ~ log10(tiempo), data = df_SIDA)
6
7 Residuals:
8      Min       1Q   Median       3Q      Max
9 -0.052459 -0.042597 -0.000831  0.039183
10
11 0.069265

```

```

13
14 Coefficients :
15             Estimate Std. Error t value
16 (Intercept)  1.91751    0.03283   58.41
17 log10(tiempo) 3.27409    0.04162   78.66
18             Pr(>|t|)
19 (Intercept)  5.25e-14 ***
20 log10(tiempo) 2.69e-15 ***
21 ---
22 Signif. codes:
23 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
24
25 Residual standard error: 0.04531 on 10 degrees of freedom
26 Multiple R-squared:  0.9984,    Adjusted R-squared:  0.9982
27 F-statistic: 6188 on 1 and 10 DF,  p-value: 2.692e-15

```

Y como en la anterior función, vamos a reconvertir los parámetros.

$$\log a = 1,917507 \rightarrow a = 10^{1,917507} = 82,7003$$

$$y = 82,7003 \cdot x^{3,27409}$$

Listing 3.18: Código curva tras reconversión de parámetros

```

1 > plot(df_SIDA)
2 > curve(10^1.91751*x^3.27409, add="TRUE", col="green")

```

Y este código nos dará como resultado la siguiente gráfica:



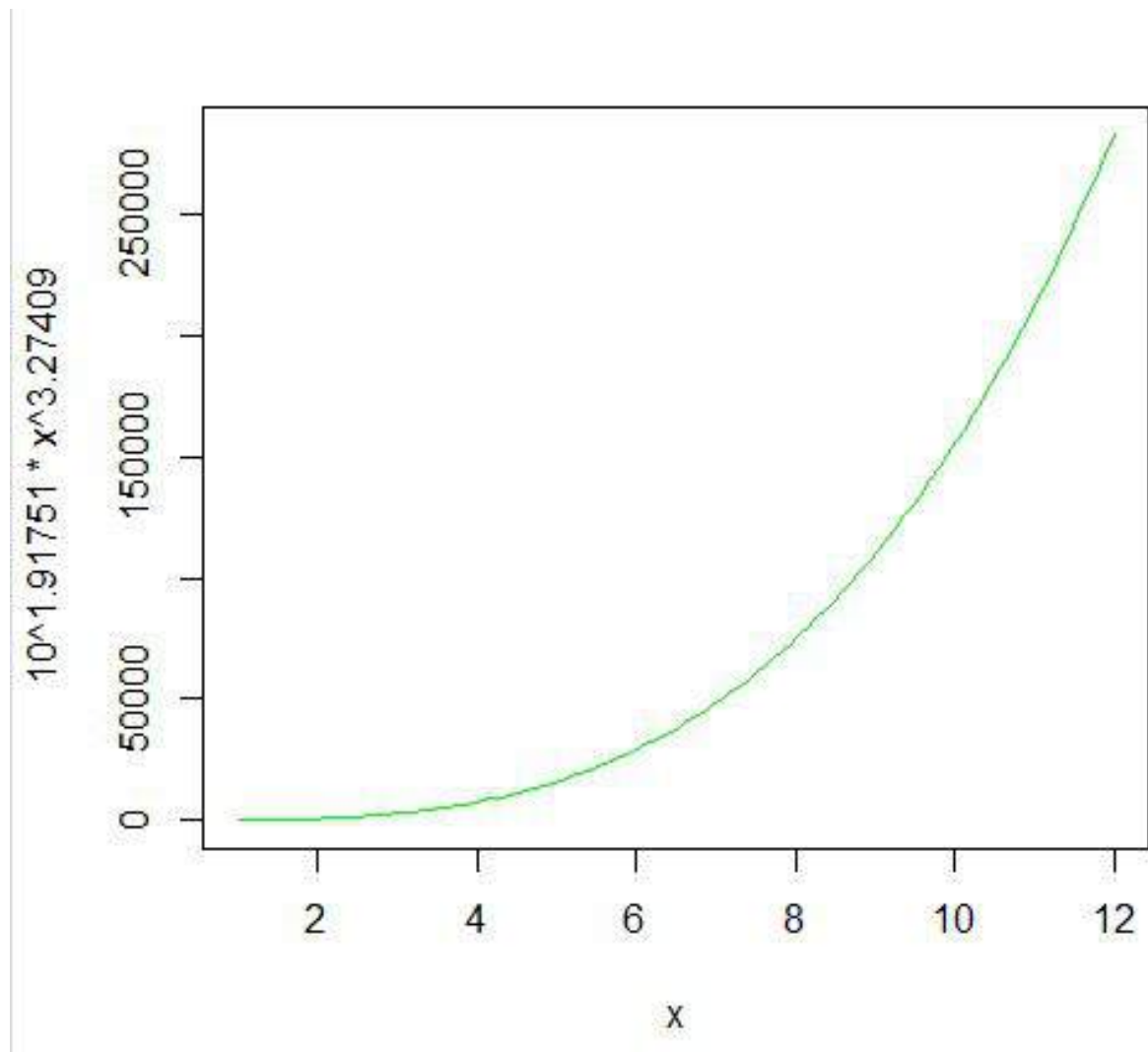


Figura 3.13: Curva tras reconversión parámetros



# Capítulo 4

## Vectores, entradas vacías, factores y listas generalizadas

### 4.1. Vectores

#### 4.1.1. Construcción de vectores

Para R un **vector** es una lista ordenada de datos **de un mismo tipo**. Dispone de una serie de funciones para definirlos:

- `c`: Sirve para concatenar datos.
- `scan`: Permite leer o importar datos.
- `rep`, `seq`: Vectores específicos.

#### Función `c`

Se aplica a los datos que tiene que formar el vector, ordenados y separados por comas. A continuación vamos a mostrar un ejemplo de vector:

Listing 4.1: Código función `c`

```
1 > x=c(1,2,3,4)
2 > x
3 [1] 1 2 3 4
4 > ABC=c("a","b","c")
5 > ABC
6 [1] "a" "b" "c"
```

Como podemos ver, para las palabras hay que entrecomillarlas, sino la función no es capaz de encontrarlas.

Esta función también sirve para concatenar vectores o añadir valores finales a un vector. Por ejemplo, podemos concatenar los dos vectores anteriores y añadir un 200 al final:

Listing 4.2: Código de funciones concatenadas con `c`

```
1 > c(x,ABC,200)
```

```

2 [1] "1" "2" "3" "4" "a" "b" "c"
3 [8] "200"

```

Se observa que los números pueden ser palabras, pero las palabras no pueden ser números por lo que les considera a todos palabras.

### Función rep

Esta función sirve para construir vectores repitiendo datos o vectores. Para definir un vector constante basta con utilizar la función `rep(x,n)` donde  $n$  es la cantidad de veces que queremos repetirlo. Pero si queremos aplicarla a vectores:

- `rep(vector, times=n)`: Repite  $n$  veces el vector en bloque.
- `rep(vector, each=n)`: Repite  $n$  veces cada entrada del vector.

A continuación vamos a ver varios ejemplos con cada uno de ellos:

Listing 4.3: Código vectores función sep

```

1 > rep("a",100)
2 [1] "a" "a" "a" "a" "a" "a" "a" "a" "a" "a" "a"
3 [11] "a" "a" "a" "a" "a" "a" "a" "a" "a" "a" "a"
4 [21] "a" "a" "a" "a" "a" "a" "a" "a" "a" "a" "a"
5 [31] "a" "a" "a" "a" "a" "a" "a" "a" "a" "a" "a"
6 [41] "a" "a" "a" "a" "a" "a" "a" "a" "a" "a" "a"
7 [51] "a" "a" "a" "a" "a" "a" "a" "a" "a" "a" "a"
8 [61] "a" "a" "a" "a" "a" "a" "a" "a" "a" "a" "a"
9 [71] "a" "a" "a" "a" "a" "a" "a" "a" "a" "a" "a"
10 [81] "a" "a" "a" "a" "a" "a" "a" "a" "a" "a" "a"
11 [91] "a" "a" "a" "a" "a" "a" "a" "a" "a" "a" "a"
12 > rep(x, times=100)
13 [1] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
14 [21] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
15 [41] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
16 [61] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
17 [81] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
18 [101] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
19 [121] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
20 [141] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
21 [161] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
22 [181] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
23 [201] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
24 [221] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
25 [241] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
26 [261] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
27 [281] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
28 [301] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
29 [321] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4

```

```

30 [341] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
31 [361] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
32 [381] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
33 > rep(x, each=100)
34 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
35 [21] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
36 [41] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
37 [61] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
38 [81] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
39 [101] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
40 [121] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
41 [141] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
42 [161] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
43 [181] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
44 [201] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
45 [221] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
46 [241] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
47 [261] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
48 [281] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
49 [301] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
50 [321] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
51 [341] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
52 [361] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
53 [381] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
54 > rep(c(1,2,3,4), times=c(5,10,15,20))
55 [1] 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3
56 [22] 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4
57 [43] 4 4 4 4 4 4 4 4 4

```

Como se puede ver en la última función podemos combinarla con la función `c` para concatenar la repetición de los distintos datos y además es importante no olvidarse de entrecomillar el texto.

### Función `seq`

Permite construir progresiones aritméticas. Sus usos básicos son los siguientes:

- `seq(a,b, by=r)`: de `a` a `b` con paso `r`.
- `seq(a, by=r, length.out=n)`: a partir de `a`, de longitud `n` con paso `r`.

Se pueden encontrar otros usos en la ayuda de la función. Ahora vamos a mostrar un ejemplo de cada uno de los usos:

Listing 4.4: Código función `seq`

```

1 > seq(0,120, by=5)
2 [1] 0 5 10 15 20 25 30 35 40 45
3 [11] 50 55 60 65 70 75 80 85 90 95

```

```

4 [21] 100 105 110 115 120
5 > seq(0,123, by=5)
6 [1] 0 5 10 15 20 25 30 35 40 45
7 [11] 50 55 60 65 70 75 80 85 90 95
8 [21] 100 105 110 115 120
9 > seq(100,17, by=-5)
10 [1] 100 95 90 85 80 75 70 65 60 55
11 [11] 50 45 40 35 30 25 20
12 > 0:20
13 [1] 0 1 2 3 4 5 6 7 8 9 10 11 12 13
14 [15] 14 15 16 17 18 19 20
15 > 20:0
16 [1] 20 19 18 17 16 15 14 13 12 11 10 9 8 7
17 [15] 6 5 4 3 2 1 0
18 > seq(0,by=5,length.out=20)
19 [1] 0 5 10 15 20 25 30 35 40 45 50 55 60 65
20 [15] 70 75 80 85 90 95

```

### Función scan

Esta función nos permite:

- `scan()`: Permite entrar en consola la lista de datos.
- `scan(fichero)`: Define un vector a partir del fichero.

Vamos a ver un ejemplo de cómo realizar un vector con la función. Es muy importante copiar los datos cuando nos aparezca un uno darle al enter y volver a darle al enter cuando haya terminado de leerlos.

Listing 4.5: Código función scan

```

1 > X=scan()
2 1: 4 10 4 9 8
3 6: 6 4 1 6 6
4 11: 9 10 4 3 7
5 16: 2 8 6 10 4
6 21: 6 10 4 3 4
7 26: 10 8 3 2 8
8 31: 9 9 6 3 4
9 36: 3 10 5 9 6
10 41:
11 Read 40 items
12 > X
13 [1] 4 10 4 9 8 6 4 1 6 6 9 10 4 3
14 [15] 7 2 8 6 10 4 6 10 4 3 4 10 8 3
15 [29] 2 8 9 9 6 3 4 3 10 5 9 6
16 > Y

```

```

17 [1] 4 10 4 9 8 6 4 1 6 6 9 10 4 3
18 [15] 7 2 8 6 10 4 6 10 4 3 4 10 8 3
19 [29] 2 8 9 9 6 3 4 3 10 5 9 6 5 9
20 [43] 7 5 7 2 8 6 8 4 9 7 4 9 3 3
21 [57] 8 8 7 6
22 > Z=scan("http://aprender.uib.es/Rdir/datostest9.txt")
23 Read 1000 items

```

Como podemos observar, se pueden extraer los vectores de un documento en el directorio o de una página web.

Además también admite algunos parámetros que son los siguientes:

- **sep**: Especifica el signo de separación de entradas.
- **dec**: Especifica el separador decimal.
- **what**: Especifica el tipo de datos.
- **encoding**: Especifica la codificación de alfabeto.

Listing 4.6: Código separación scan

```

1 > scan()
2 1: 1,2,3,4,5
3 1:
4 Error in scan() : scan() expected 'a real', got '1,2,3,4,5'
5 > scan(sep=",")
6 1: 1,2,3,4,5
7 6:
8 Read 5 items
9 [1] 1 2 3 4 5
10 > scan()
11 1: Sara Lucia David Rocio
12 Error in scan() : scan() expected 'a real', got 'Sara'
13 > scan(what="character")
14 1: Sara Lucia David Rocio
15 5:
16 Read 4 items
17 [1] "Sara" "Lucia" "David" "Rocio"

```

### 4.1.2. Operaciones con vectores

R permite aplicar una función a todas las entradas de un vector de golpe:

- Aplicar la función al vector: `función(vector)`.
- Si no funciona: `sapply(vector, FUN=función)`.

A continuación vamos a mostrar algunos ejemplos de progresión aritmética:

Listing 4.7: Código progresión aritmética

```

1 > x=1:10
2 > x
3 [1] 1 2 3 4 5 6 7 8 9 10
4 > x^2+5
5 [1] 6 9 14 21 30 41 54 69 86 105
6 > sqrt(x)
7 [1] 1.000000 1.414214 1.732051 2.000000
8 [5] 2.236068 2.449490 2.645751 2.828427
9 [9] 3.000000 3.162278
10 > 3+5*(0:19)
11 [1] 3 8 13 18 23 28 33 38 43 48 53 58 63 68
12 [15] 73 78 83 88 93 98
13 > sum(x)
14 [1] 55

```

Pero no todas pueden utilizar la primera función, ya que la función suma se aplica a cada una de sus entradas.

Listing 4.8: Código definición función

```

1 > f=function(n){sum(1:n)}
2 > f(1)
3 [1] 1
4 > f(2)
5 [1] 3

```

Esta función no se puede aplicar a los vectores pero si utilizamos [sapply](#) se podrá operar sin problemas.

Listing 4.9: Código función sapply

```

1 > sapply(1:10, FUN=f)
2 [1] 1 3 6 10 15 21 28 36 45 55

```

Además de poder realizar las funciones, podemos multiplicar cada entrada como vamos a ver en el siguiente ejemplo:

Listing 4.10: Código multiplicación entradas

```

1 > y=log(x)
2 > x*y
3 [1] 0.000000 1.386294 3.295837 5.545177
4 [5] 8.047190 10.750557 13.621371 16.635532
5 [9] 19.775021 23.025851

```

También podemos encontrar funciones específicas como:

- [length](#): Longitud.



- `max` y `min`: Máximo y mínimo.
- `sum`: Suma de las entradas.
- `prod`: Producto de las entradas.
- `mean`: Media aritmética de las entradas.
- `cumsum`: Vector sumas acumuladas de las entradas.
- `sort`: Ordena las entradas en orden creciente; el parámetro `dec=TRUE` indica que el orden sea decreciente.
- `rev`: Invierte el orden de las entradas.

Vamos a poner un ejemplo de cada una de estas funciones:

Listing 4.11: Código funciones específicas

```

1 > n=1:20
2 > x=3^n-10*2^n
3 > x
4 [1] -17 -31 -53
5 [4] -79 -77 89
6 [7] 907 4001 14563
7 [10] 48809 156667 490481
8 [13] 1512403 4619129 14021227
9 [16] 42391361 127829443 384799049
10 [19] 1157018587 3476298641
11 > length(x)
12 [1] 20
13 > max(x)
14 [1] 3476298641
15 > min(x)
16 [1] -79
17 > sum(x)
18 [1] 5209205100
19 > prod(x)
20 [1] -2.46226e+99
21 > mean(x)
22 [1] 260460255
23 > sum(x)/length(x)
24 [1] 260460255
25 > sqrt(sum(x^2))
26 [1] 3686430854
27 > sort(x)
28 [1] -79 -77 -53
29 [4] -31 -17 89
30 [7] 907 4001 14563
31 [10] 48809 156667 490481
32 [13] 1512403 4619129 14021227

```

```

33 [16] 42391361 127829443 384799049
34 [19] 1157018587 3476298641
35 > rev(x)
36 [1] 3476298641 1157018587 384799049
37 [4] 127829443 42391361 14021227
38 [7] 4619129 1512403 490481
39 [10] 156667 48809 14563
40 [13] 4001 907 89
41 [16] -77 -79 -53
42 [19] -31 -17
43 > rev(sort(x))
44 [1] 3476298641 1157018587 384799049
45 [4] 127829443 42391361 14021227
46 [7] 4619129 1512403 490481
47 [10] 156667 48809 14563
48 [13] 4001 907 89
49 [16] -17 -31 -53
50 [19] -77 -79
51 > cumsum(x)
52 [1] -17 -48 -101
53 [4] -180 -257 -168
54 [7] 739 4740 19303
55 [10] 68112 224779 715260
56 [13] 2227663 6846792 20868019
57 [16] 63259380 191088823 575887872
58 [19] 1732906459 5209205100

```

La función `cumsum` es muy útil para calcular frecuencias acumuladas en estadística.

### 4.1.3. Entradas y trozos de vectores

Para las entradas de los vectores tendremos en cuenta de que  $i$  es un índice  $e$ ,  $V$  es un vector y que es un vector de índices:

- $V[i]$ : Entrada  $i$ -ésima del vector  $V$ .
- $V[y]$ : Entradas del vector  $V$  de índices en  $y$ .
- $V[-y]$ : El complementario de  $V[y]$ .

Ahora vamos a proceder a ver un ejemplo de entrada de un vector:

Listing 4.12: Código entrada vector

```

1 > x=2+3*(1:50)
2 > x
3 [1] 5 8 11 14 17 20 23 26 29 32
4 [11] 35 38 41 44 47 50 53 56 59 62
5 [21] 65 68 71 74 77 80 83 86 89 92

```

```

6 [31] 95 98 101 104 107 110 113 116 119 122
7 [41] 125 128 131 134 137 140 143 146 149 152
8 > x[3]
9 [1] 11
10 > x[length(x)]
11 [1] 152
12 > x[c(1,3,10)]
13 [1] 5 11 32
14 > x[10:20]
15 [1] 32 35 38 41 44 47 50 53 56 59 62
16 > x[-(10:20)]
17 [1] 5 8 11 14 17 20 23 26 29 65
18 [11] 68 71 74 77 80 83 86 89 92 95
19 [21] 98 101 104 107 110 113 116 119 122 125
20 [31] 128 131 134 137 140 143 146 149 152
21 > x[10:20]==0
22 > x
23 [1] 5 8 11 14 17 20 23 26 29 0
24 [11] 0 0 0 0 0 0 0 0 0 0
25 [21] 65 68 71 74 77 80 83 86 89 92
26 [31] 95 98 101 104 107 110 113 116 119 122
27 [41] 125 128 131 134 137 140 143 146 149 152

```

Hay otras posibilidades de indicar la entrada de un vector:

- `V[condición]`: Entradas del vector  $V$  que satisfacen la condición.
- `which(condición)`: Índices de las entradas del vector  $V$  que satisfacen la condición.

Además de estas posibilidades encontraremos los siguientes signos lógicos para construir estas condiciones:

Cuadro 4.1: Signos lógicos

Operador	=	≠	<	>	≤	≥	Negación	Conjunción	Disyunción
Signo	==	!=	<	>	<=	>=	!	&	

Finalmente, procederemos a realizar un ejemplo de cada una de estas nuevas condiciones:

Listing 4.13: Código condicionales

```

1 > x[x>=20]
2 [1] 20 23 26 29 65 68 71 74 77 80
3 [11] 83 86 89 92 95 98 101 104 107 110
4 [21] 113 116 119 122 125 128 131 134 137 140
5 [31] 143 146 149 152
6 > x[x%/%==0]
7 [1] 0 0 0 0 0 0 0 0 0 0
8 > which(x%/%==0)

```

```

9 [1] 10 11 12 13 14 15 16 17 18 19 20
10 > x[x %/% 5 == 0 & x >= 20]
11 numeric(0)

```

## 4.2. Entradas vacías de un vector

En este capítulo vamos a ver el manejo de vectores con datos desconocidos.

R indica las entradas no definidas o vacías con `NA`. Vamos a proceder a dar un ejemplo de un vector de entrada vacía:

Listing 4.14: Código vector con entradas vacías

```

1 > x=1:10
2 > x
3 [1] 1 2 3 4 5 6 7 8 9 10
4 > x[13]=13
5 > x
6 [1] 1 2 3 4 5 6 7 8 9 10 NA NA 13
7 > sum(x)
8 [1] NA

```

Hay dos soluciones a este problema:

- `is.na(V)`: Condición “entradas  $V$  no definidas”.
- Entradas definidas de  $V$ : `V[!is.na(V)]` o `na.omit(V)`.

Seguidamente, vamos a realizar un ejemplo que solucione el ejemplo anterior con estas dos nuevas funciones:

Listing 4.15: Código nuevo vector sin entradas vacías

```

1 > which(is.na(x))
2 [1] 11 12
3 > y=x[!is.na(x)]
4 > y
5 [1] 1 2 3 4 5 6 7 8 9 10 13
6 > sum(y)
7 [1] 68
8 > sum(na.omit(x))
9 [1] 68

```

Por otro lado, podremos borrar los datos a mano, para ello tenemos la función `na.rm=TRUE` que se puede aplicar a una función a un vector sin tener en cuenta los `NA`. Vamos a proceder a realizar una suma con el vector anterior:

Listing 4.16: Código borrar datos a mano

```

1 > sum(x, na.rm = TRUE)
2 [1] 68
3 > mean(x)
4 [1] NA
5 > mean(x, na.rm = TRUE)
6 [1] 6.181818

```

Finalmente, no todas las funciones no admiten este parámetro. A continuación vamos a mostrar algún ejemplo de ellas:

Listing 4.17: Código funciones no admiten parámetro

```

1 > cumsum(x, na.rm=TRUE)
2 Error in cumsum(x, na.rm = TRUE) :
3   2 arguments passed to 'cumsum' which requires 1
4 > cumsum(na.omit(x))
5 [1] 1 3 6 10 15 21 28 36 45 55 68

```

### 4.3. Factores

Un factor es un vector con una estructura adecuada para analizar datos. Podemos definir un factor de dos maneras a partir de un vector:

- `as.factor(vector)`: Convierte el vector en un factor.

Vamos a poner un ejemplo con esta función para observar las diferencias entre un factor y un vector:

Listing 4.18: Código diferencias factor de vector

```

1 > zonas=c(1,3,2,2,2,2,1,4,3,2,4,4,2,4,2,1,4)
2 > zonas
3 [1] 1 3 2 2 2 2 1 4 3 2 4 4 2 4 2 1 4
4 > zonas.f=as.factor(zonas)
5 > zonas.f
6 [1] 1 3 2 2 2 2 1 4 3 2 4 4 2 4 2 1 4
7 Levels: 1 2 3 4
8 > sum(zonas)
9 [1] 43
10 > sum(zonas.f)
11 Error in Summary.factor(c(1L, 3L, 2L, 2L, 2L, 2L, 1L, 4L, 3L,
12   2L, 4L,  :
sum not meaningful for factors

```

Como podemos observar, el factor tiene un parámetro denominado Levels que permite considerar los elementos del factor como copia de estos niveles y otra diferencia es que no podemos aplicar las funciones del vector al factor.

- `factor(vector)`: Crea un factor a partir del vector. Esta función tiene los siguientes parámetros básicos:
  - `levels`: Especifica los niveles.
  - `labels`: Cambia los nombres de los niveles.
- `levels(factor)`: Vector de niveles del factor, permite modificarlos.

Vamos a realizar un ejemplo con estas nuevas funciones de creación de factores:

Listing 4.19: Código de definición de factores

```

1 > zonas.f2=factor(zonas, levels=c(1,2,3,4,5), labels = c("a",
  "e", "i", "o", "u"))
2 > zonas.f2
3 [1] a i e e e e a o i e o o e o e a o
4 Levels: a e i o u
5 > levels(zonas.f2)
6 [1] "a" "e" "i" "o" "u"
7 > levels(zonas.f2)=c("A", "B", "C", "D", "E")
8 > zonas.f2
9 [1] A C B B B B A D C B D D B D B A D
10 Levels: A B C D E

```

Finalmente, habrá ocasiones en las que nos interese que los niveles tengan un orden concreto y que nos permitirá contar los individuos de cada nivel, en esta caso utilizaremos la función `ordered(vector)` que crea un factor ordenado a partir del vector. Es importante que especifiquemos el orden de los niveles. Vamos a poner un ejemplo de un factor ordenado:

Listing 4.20: Código factor ordenado por niveles

```

1 > zonas.fo=ordered(zonas, levels=c(1,3,4,2), labels=c("S", "C-
  S", "C-N", "N"))
2 > zonas.fo
3 [1] S C-S N N N N S C-N C-S N
4 [11] C-N C-N N C-N N S C-N
5 Levels: S < C-S < C-N < N

```

## 4.4. Listas generalizadas

Como hemos dicho en capítulos anteriores, R necesita que todos los datos sean de un mismo tipo. Pero, a veces, necesitamos utilizar vectores de datos heterogéneos. En este capítulo vamos a ver como manejar estos vectores, que en R se llaman `list`.

Seguidamente, vamos a recordar los vectores homogéneos utilizados en capítulos anteriores:

Listing 4.21: Código vectores homogéneos

```

1 > edad=c(1,2,3,5,7,9,11,13)
2 > altura=c
      (76.11,86.45,95.27,109.18,122.03,133.73,143.73,156.41)
3 > summary(lm(altura~edad))
4
5 Call:
6 lm(formula = altura ~ edad)
7
8 Residuals:
9      Min       1Q   Median       3Q      Max
10 -4.351  -1.743   0.408   2.018   2.745
11
12 Coefficients:
13             Estimate Std. Error t value
14 (Intercept)  73.9681     1.7979  41.14
15 edad         6.4934     0.2374  27.36
16             Pr(>|t|)
17 (Intercept) 1.38e-08 ***
18 edad        1.58e-07 ***
19 ---
20 Signif. codes:
21 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
22
23 Residual standard error: 2.746 on 6 degrees of freedom
24 Multiple R-squared:  0.992, Adjusted R-squared:  0.9907
25 F-statistic: 748.4 on 1 and 6 DF, p-value: 1.577e-07
26 > str(summary(lm(altura~edad)))
27 List of 11
28 $ call      : language lm(formula = altura ~ edad)
29 $ terms     : Classes 'terms', 'formula' language altura ~
      edad

```

Hemos acertado el `str` para quedarnos con lo importante de la función. Esta función nos indica que hay una lista de 11 elementos.

Seguidamente, vamos a ver con qué funciones podemos definir nuestras `lists`:

- `list(objeto1, objeto2,...)`: Crea list con los objetos.
- `list(nombre1=objeto1, nombre2=objeto2,...)`: Crea list con los objetos, y les asigna los nombres internos.
- `str(list)`: Estructura interna de la list.

Vamos a realizar un ejemplo con estas nuevas funciones:

Listing 4.22: Código creación nuevas listas

```
1 > datos=list(vec=edad, nombre.vec="edad", long=length(edad),
2           media=mean(edad))
3 > datos
4 $vec
5 [1] 1 2 3 5 7 9 11 13
6
7 $nombre.vec
8 [1] "edad"
9
10 $long
11 [1] 8
12
13 $media
14 [1] 6.375
15 > str(datos)
16 List of 4
17 $ vec      : num [1:8] 1 2 3 5 7 9 11 13
18 $ nombre.vec: chr "edad"
19 $ long     : int 8
20 $ media    : num 6.38
```

Finalmente, vamos a ver cómo sacar un objeto de una list. Tenemos dos opciones:

- `list$nombre`: Objeto de la list correspondiente al nombre.
- `list[[i]]`: Objeto i-ésimo de la list.
- `list[i]`: List formada por el objeto i-ésimo de la list.

Las dos últimas funciones se pueden aplicar a vectores de índices. Vamos a ver cómo extraer los datos de nuestro ejemplo:

Listing 4.23: Código extracción objetos de la list

```
1 > datos$vec
2 [1] 1 2 3 5 7 9 11 13
3 > datos[[1]]
4 [1] 1 2 3 5 7 9 11 13
5 > datos$nombre.vec
6 [1] "edad"
7 > datos[[2]]
8 [1] "edad"
9 > nombre.vec
10 Error: object 'nombre.vec' not found
11 > datos[[1]]
12 [1] 1 2 3 5 7 9 11 13
13 > datos[1]
14 $vec
```



```
15 [1] 1 2 3 5 7 9 11 13
```



# Capítulo 5

## Matrices y vectores propios

### 5.1. Matrices

#### 5.1.1. Construcción de matrices

Las matrices son tablas rectangulares de un mismo tipo, normalmente números. En este capítulo vamos a ver las dos maneras distintas que tiene R para construir matrices. Es importante que todas sus entradas sean del mismo tipo de datos. Podemos construir matrices con las siguientes funciones:

- `matrix(vector,...)`: Construye una matriz con las entradas del vector. Tiene los siguientes parámetros básicos:
  - `byrow`: Indica si se organizan por filas (**TRUE**), o por columnas (**FALSE**).
  - `nrow` o `ncol`: Número de filas o de columnas.

Vamos a proceder a construir una matriz sencilla con esta función:

Listing 5.1: Código construcción matrices

```
1 > x=1:12
2 > matrix(x, nrow=3)
3      [,1] [,2] [,3] [,4]
4 [1,]    1    4    7   10
5 [2,]    2    5    8   11
6 [3,]    3    6    9   12
7 > matrix(x, nrow=3, byrow = TRUE)
8      [,1] [,2] [,3] [,4]
9 [1,]    1    2    3    4
10 [2,]    5    6    7    8
11 [3,]    9   10   11   12
12 > matrix(x, ncol=4, byrow = TRUE)
13      [,1] [,2] [,3] [,4]
14 [1,]    1    2    3    4
15 [2,]    5    6    7    8
16 [3,]    9   10   11   12
17 > matrix(x, ncol=5, byrow = TRUE)
```

```

18      [,1] [,2] [,3] [,4] [,5]
19 [1,]    1    2    3    4    5
20 [2,]    6    7    8    9   10
21 [3,]   11   12    1    2    3
22 Warning message:
23 In matrix(x, ncol = 5, byrow = TRUE) :
24   data length [12] is not a sub-multiple or multiple of the
      number of columns [5]
25 > matrix(x, ncol=5,nrow = 5, byrow = TRUE)
26      [,1] [,2] [,3] [,4] [,5]
27 [1,]    1    2    3    4    5
28 [2,]    6    7    8    9   10
29 [3,]   11   12    1    2    3
30 [4,]    4    5    6    7    8
31 [5,]    9   10   11   12    1
32 Warning message:
33 In matrix(x, ncol = 5, nrow = 5, byrow = TRUE) :
34   data length [12] is not a sub-multiple or multiple of the
      number of rows [5]
35 > matrix(1, nrow = 3, ncol = 3)
36      [,1] [,2] [,3]
37 [1,]    1    1    1
38 [2,]    1    1    1
39 [3,]    1    1    1

```

Esta función es útil para representar matrices constantes.

Otra manera de construir matrices es mediante la concatenación de vectores, y se realiza mediante las siguientes funciones:

- `cbind`: Concatena matrices y vectores por columnas.
- `rbind`: Concatena matrices y vectores por filas.

En ambos casos las dimensiones han de ser adecuadas. Vamos a proceder a ver un ejemplo con estas funciones:

Listing 5.2: Código concatenación vectores y matrices

```

1 > a=1:4
2 > b=c(1,3,5,7)
3 > c=rep(0,4)
4 > rbind(a,b,c)
5      [,1] [,2] [,3] [,4]
6 a      1    2    3    4
7 b      1    3    5    7
8 c      0    0    0    0
9 > cbind(a,b,c)
10      a b c
11 [1,]  1 1 0

```

```

12 [2,] 2 3 0
13 [3,] 3 5 0
14 [4,] 4 7 0
15 > A=diag(4)
16 > A
17      [,1] [,2] [,3] [,4]
18 [1,]    1    0    0    0
19 [2,]    0    1    0    0
20 [3,]    0    0    1    0
21 [4,]    0    0    0    1
22 > rbind(A,a,b,c)
23      [,1] [,2] [,3] [,4]
24      1    0    0    0
25      0    1    0    0
26      0    0    1    0
27      0    0    0    1
28 a      1    2    3    4
29 b      1    3    5    7
30 c      0    0    0    0
31 > cbind(a,A,b,c)
32      a      b c
33 [1,] 1 1 0 0 0 1 0
34 [2,] 2 0 1 0 0 3 0
35 [3,] 3 0 0 1 0 5 0
36 [4,] 4 0 0 0 1 7 0

```

Como podemos observar, la función `diag` crea una matriz diagonal de 4x4.

### 5.1.2. Entradas y trozos de matrices

Es similar al capítulo de los vectores, pero tendremos que tener en cuenta las dos dimensiones de la matriz.

En general las entradas de la matriz se indican como en los vectores pero con una función bidimensional que es `M[i,j]`, donde primero se presenta la fila y luego la columna. Vamos a ver un ejemplo sencillo de esta función:

Listing 5.3: Código entrada matriz

```

1 > A=matrix(1:30,nrow = 5)
2 > A
3      [,1] [,2] [,3] [,4] [,5] [,6]
4 [1,]    1    6   11   16   21   26
5 [2,]    2    7   12   17   22   27
6 [3,]    3    8   13   18   23   28
7 [4,]    4    9   14   19   24   29

```

```

8 [ 5 , ]      5   10   15   20   25   30
9 > A[ 2 , 3 ]
10 [ 1 ] 12

```

Podremos distinguir entre filas y columnas a través de las siguientes funciones:

- `M[i,]`: Fila.
- `M[,j]`: Columna.

Usando vectores de índices en vez de índices, se obtienen submatrices determinadas por las filas y columnas indicadas. Vamos a ver un ejemplo con estas distinciones a partir del ejemplo anterior:

Listing 5.4: Código distinción filas y columnas

```

1 > A[ 2 , 3 ]
2 [ 1 ] 12
3 > A[ 3 , ]
4 [ 1 ] 3  8 13 18 23 28
5 > A[ , 3 ]
6 [ 1 ] 11 12 13 14 15
7 > A[ , 1:2 ]
8      [ , 1 ] [ , 2 ]
9 [ 1 , ]      1      6
10 [ 2 , ]      2      7
11 [ 3 , ]      3      8
12 [ 4 , ]      4      9
13 [ 5 , ]      5     10
14 > A[ c( 1 , 3 , 5 ) , c( 2 , 4 , 6 ) ]
15      [ , 1 ] [ , 2 ] [ , 3 ]
16 [ 1 , ]      6     16     26
17 [ 2 , ]      8     18     28
18 [ 3 , ]     10     20     30

```

Finalmente, tenemos la función `diag` que nos da la matriz diagonal. Vamos a ver otro ejemplo de matriz diagonal con la matriz ejemplo anterior:

Listing 5.5: Código matriz diagonal

```

1 > diag(A)
2 [ 1 ] 1  7 13 19 25

```

### 5.1.3. Funciones para matrices

En este capítulo vamos a ver las funciones básicas con matrices. Primero vamos a ver las funciones relacionadas con las dimensiones de las matrices:

- `nrow`: Número de filas.

- `ncol`: Número de columnas.
- `dim`: Vector con ambas dimensiones.

Vamos a realizar un ejemplo con estas funciones básicas:

Listing 5.6: Código funciones básicas dimensiones

```

1 > A=matrix(1:30, nrow = 5)
2 > A
3      [,1] [,2] [,3] [,4] [,5] [,6]
4 [1,]    1    6   11   16   21   26
5 [2,]    2    7   12   17   22   27
6 [3,]    3    8   13   18   23   28
7 [4,]    4    9   14   19   24   29
8 [5,]    5   10   15   20   25   30
9 > nrow(A)
10 [1] 5
11 > ncol(A)
12 [1] 6
13 > dim(A)
14 [1] 5 6

```

Podremos utilizar algunas funciones que se utilizan para vectores como `sum`, `prod`, `mean`,..., también podemos aplicar las funciones a las filas o columnas de la matriz. Como por ejemplo, podremos aplicar las siguientes funciones:

- `colSums`: Vector de sumas de columnas.
- `rowSums`: Vector de sumas de filas.
- `colMeans`: Vector de medias de columnas.
- `rowMeans`: Vector de medias de filas.

Vamos a aplicar estas funciones al ejemplo anterior:

Listing 5.7: Código funciones básicas comunes vectores

```

1 > mean(A)
2 [1] 15.5
3 > rowSums(A)
4 [1] 81 87 93 99 105
5 > colMeans(A)
6 [1] 3 8 13 18 23 28

```

Finalmente, también podremos aplicar la función `apply(matriz, FUN=función, MARGIN=...)`, que aplica la función a la matriz:

- `MARGIN=1`: Fila a fila.
- `MARGIN=2`: Columna a columna.

- **MARGIN=c(1,2)**: Entrada a entrada.

Esta función también puede ser utilizada como **función(matriz)** para que lo aplique a todas las entradas de la matriz. Vamos a continuar con ejemplos de esta función aplicados a la matriz ejemplo A:

Listing 5.8: Código función elemento a elemento

```

1 > apply(A, FUN=prod, MARGIN=1)
2 [1] 576576 1696464 3616704 6664896
3 [5] 11250000
4 > f=function(x){sqrt(sum(x^2))}
5 > apply(A, FUN=f, MARGIN = 2)
6 [1] 7.416198 18.165902 29.240383 40.373258
7 [5] 51.526692 62.689712
8 > log(A)
9          [,1]      [,2]      [,3]      [,4]
10 [1,] 0.0000000 1.791759 2.397895 2.772589
11 [2,] 0.6931472 1.945910 2.484907 2.833213
12 [3,] 1.0986123 2.079442 2.564949 2.890372
13 [4,] 1.3862944 2.197225 2.639057 2.944439
14 [5,] 1.6094379 2.302585 2.708050 2.995732
15          [,5]      [,6]
16 [1,] 3.044522 3.258097
17 [2,] 3.091042 3.295837
18 [3,] 3.135494 3.332205
19 [4,] 3.178054 3.367296
20 [5,] 3.218876 3.401197
21 > apply(A, FUN=log, MARGIN=c(1,2) )
22          [,1]      [,2]      [,3]      [,4]
23 [1,] 0.0000000 1.791759 2.397895 2.772589
24 [2,] 0.6931472 1.945910 2.484907 2.833213
25 [3,] 1.0986123 2.079442 2.564949 2.890372
26 [4,] 1.3862944 2.197225 2.639057 2.944439
27 [5,] 1.6094379 2.302585 2.708050 2.995732
28          [,5]      [,6]
29 [1,] 3.044522 3.258097
30 [2,] 3.091042 3.295837
31 [3,] 3.135494 3.332205
32 [4,] 3.178054 3.367296
33 [5,] 3.218876 3.401197

```

#### 5.1.4. Álgebra matricial

En este capítulo vamos a ver las operaciones algebraicas con matrices, que son las siguientes:

- Suma: +.



- Producto por escalar: `*`.
- Producto: `%*%`.
- Transpuesta: `t`.
- Determinante: `det`.
- Inversa: `solve`.
- Potencia: Instalar antes el paquete `expm` y se realiza con la función `%^%`.
- Rango: `qr(...)$rank`.

Vamos a ver algunos ejemplos de estas operaciones algebraicas con matrices:

Listing 5.9: Código operaciones algebraicas con matrices

```

1 > A=matrix(1:4, nrow=2)
2 > B=matrix(1, nrow = 2, ncol = 2)
3 > C=matrix(1:6, nrow = 2)
4 > A
5      [,1] [,2]
6 [1,]    1    3
7 [2,]    2    4
8 > B
9      [,1] [,2]
10 [1,]    1    1
11 [2,]    1    1
12 > C
13      [,1] [,2] [,3]
14 [1,]    1    3    5
15 [2,]    2    4    6
16 > A+B
17      [,1] [,2]
18 [1,]    2    4
19 [2,]    3    5
20 > A+C
21 Error in A + C : non-conformable arrays
22 > 5*A
23      [,1] [,2]
24 [1,]    5   15
25 [2,]   10   20
26 > A%*%B
27      [,1] [,2]
28 [1,]    4    4
29 [2,]    6    6
30 > A*B
31      [,1] [,2]
32 [1,]    1    3
33 [2,]    2    4

```

```

34 > t(C)
35      [,1] [,2]
36 [1,]    1    2
37 [2,]    3    4
38 [3,]    5    6
39 > det(A)
40 [1] -2
41 > det(C)
42 Error in determinant.matrix(x, logarithm, ...) :
43   'x' must be a square matrix
44 > solve(A)
45      [,1] [,2]
46 [1,]   -2  1.5
47 [2,]    1 -0.5
48 > solve(B)
49 Error in solve.default(B) :
50   Lapack routine dgesv: system is exactly singular: U[2,2] = 0
51 > qr(B)$rank
52 [1] 1
53 > B%^%20
54      [,1] [,2]
55 [1,] 524288 524288
56 [2,] 524288 524288
57 > B^20
58      [,1] [,2]
59 [1,]    1    1
60 [2,]    1    1

```

Podremos resolver sistemas de ecuaciones si tenemos en cuenta que  $A$  es una matriz cuadrada de  $n \times n$  y  $b$  es un vector de longitud  $n$ , para ello vamos a utilizar la siguiente función:

- `solve(A,b)`:  $A^{-1} \cdot b$ .

Este sistema se utiliza para resolver sistemas compatibles determinados. Vamos a poner un sistema compatible determinado de ejemplo con su resolución y su solución en R:

$$\left. \begin{array}{l} x + 3y = 1 \\ 2x + 4y = 3 \end{array} \right\} \rightarrow \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}^{-1} \cdot \begin{pmatrix} 1 \\ 3 \end{pmatrix}$$

Listing 5.10: Código solución sistema compatible determinado

```

1 > solve(A, c(1,3))
2 [1] 2.5 -0.5

```

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2,5 \\ -0,5 \end{pmatrix}$$

## 5.2. Cálculo de valores y vectores propios

En este capítulo vamos a ver cómo calcular los valores y vectores propios de una matriz y cómo descomponer los datos de una matriz diagonalizable.

Primero vamos a definir cuando un vector es propio. Un vector  $v = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \neq 0$  es un vector propio de la matriz

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \vdots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}$$

cuando existe  $\lambda \in \mathbb{C}$  tal que  $A \cdot v = \lambda \cdot v$ .

Entonces,  $\lambda$  es un valor propio de  $A$ , y  $v$  es un vector propio de  $A$  de valor propio  $\lambda$ . Vamos a proceder a realizar un ejemplo de vector propio:

$$A = \begin{pmatrix} 4 & -6 & 6 \\ 3 & -5 & 8 \\ 2 & -4 & 7 \end{pmatrix}$$

$v = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$  es vector propio de  $A$  de valor propio 3, porque:

$$\begin{pmatrix} 4 & -6 & 6 \\ 3 & -5 & 8 \\ 2 & -4 & 7 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 3 \\ 3 \end{pmatrix} = 3 \cdot \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

En cambio  $w = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$  no es vector propio de  $A$ , porque:

$$\begin{pmatrix} 4 & -6 & 6 \\ 3 & -5 & 8 \\ 2 & -4 & 7 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \\ 5 \end{pmatrix} \neq \begin{pmatrix} \lambda \\ \lambda \\ \lambda \end{pmatrix} = \lambda \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Los valores y vectores propios de una matriz  $A$  se calculan con la función `eigen(A)`, cuyo resultado da una list con dos objetos:

- **values**: Vector de valores propios.
- **vectors**: Matriz de columnas vectores propios, en el mismo orden.

Vamos a ver un ejemplo con esta función:

Listing 5.11: Código función valores y vectores propios

```

1 > A=matrix(c(4, -6, 6, 3, -5, 8, 2, -4, 7), nrow=3, byrow=TRUE)
2 > B=matrix(c(4, 5, 3, -4, 13, 3, -4, 5, 11), nrow=3, byrow=TRUE)
3 > C=matrix(c(0, 3, 4, 4, -4, -8, -4, 8, 12), nrow = 3, byrow = TRUE)
4 > A

```

```

5      [,1] [,2] [,3]
6 [1,]    4   -6    6
7 [2,]    3   -5    8
8 [3,]    2   -4    7
9 > eigen(A)
10 eigen() decomposition
11 $values
12 [1] 3 2 1
13
14 $vectors
15      [,1] [,2] [,3]
16 [1,] 9.186681e-16 -0.8017837 -8.944272e-01
17 [2,] 7.071068e-01  0.2672612 -4.472136e-01
18 [3,] 7.071068e-01  0.5345225  1.297573e-15
19
20 > eigen(A)$values
21 [1] 3 2 1
22 > eigen(A)$vectors
23      [,1] [,2] [,3]
24 [1,] 9.186681e-16 -0.8017837 -8.944272e-01
25 [2,] 7.071068e-01  0.2672612 -4.472136e-01
26 [3,] 7.071068e-01  0.5345225  1.297573e-15
27 > B
28      [,1] [,2] [,3]
29 [1,]    4    5    3
30 [2,]   -4   13    3
31 [3,]   -4    5   11
32 > eigen(B)
33 eigen() decomposition
34 $values
35 [1] 12 8 8
36
37 $vectors
38      [,1] [,2] [,3]
39 [1,] 0.5773503 -0.8164966 -0.3064116
40 [2,] 0.5773503 -0.4082483 -0.6579007
41 [3,] 0.5773503 -0.4082483  0.6879524

```

Si  $A$  es diagonalizable, del resultado `eigen(A)` se obtiene una descomposición canónica:

$$A = P \cdot D \cdot P^{-1}$$

**D:** Matriz diagonal con diagonal principal `eigen(A)$values` cuya función es `diag(eigen(A)$values)`.

**P:** La matriz `eigen(A)$vectors`.

Vamos a ver cómo aplicar esta descomposición canónica con un ejemplo:

Listing 5.12: Código descomposición canónica

```

1 > A
2     [,1] [,2] [,3]
3 [1,]    4  -6    6
4 [2,]    3  -5    8
5 [3,]    2  -4    7
6 > DA=diag(eigen(A)$values)
7 > PA=eigen(A)$vectors
8 > PA%*%DA%*%solve(PA)
9     [,1] [,2] [,3]
10 [1,]    4  -6    6
11 [2,]    3  -5    8
12 [3,]    2  -4    7
13 > DB=diag(eigen(B)$values)
14 > PB=eigen(B)$vectors
15 > PB%*%DB%*%solve(PB)
16     [,1] [,2] [,3]
17 [1,]    4    5    3
18 [2,]   -4   13    3
19 [3,]   -4    5   11
20 > B
21     [,1] [,2] [,3]
22 [1,]    4    5    3
23 [2,]   -4   13    3
24 [3,]   -4    5   11
25 > C
26     [,1] [,2] [,3]
27 [1,]    0    3    4
28 [2,]    4   -4   -8
29 [3,]   -4    8   12
30 > eigen(C)
31 eigen() decomposition
32 $values
33 [1] 4+0i 2+0i 2-0i
34
35 $vectors
36     [,1] [,2]
37 [1,] 0.2981424+0i -0.3333333+0i
38 [2,] -0.5962848+0i 0.6666667+0i
39 [3,] 0.7453560+0i -0.6666667+0i
40     [,3]
41 [1,] -0.3333333-0i
42 [2,] 0.6666667+0i
43 [3,] -0.6666667-0i

```



# Capítulo 6

## Data frame

### 6.1. Introducción

En este capítulo vamos a ver el uso, manejo y características de los data frame.

Un data frame es una tabla de doble entrada formada por variables en las columnas y observaciones de estas en las filas. Tienen el mismo aspecto que una matriz, pero pueden contener diferentes datos. Pero los datos de una misma columna deben ser de un mismo tipo.

Cuadro 6.1: Estructura general data frame

	Variable 1	Variable 2	...	Variable m
Obs. 1				
Obs. 2				
...				
Obs. n				

Las columnas son factores o vectores, ya que son de un mismo tipo, mientras que las filas son lists.

Vamos a considerar los siguientes tipos de datos:

- **Atributo o cualitativos:** Son los que expresan una cualidad del individuo. En R los guardaremos en vectores en factores.
- **Ordinales:** Similares a los anteriores pero se pueden ordenar de forma natural. En R los guardaremos en factores ordenados.
- **Cuantitativos:** Son datos numerales que se refieren a medidas. En R los guardaremos en vectores de números.

### 6.2. Estructura

En este capítulo vamos a ver las distintas funciones utilizadas para obtener información sobre los data frame.

Una tabla de las más conocidas de R es la data frame iris. Contiene la longitud y la anchura de los pétalos y sépalos y la especie de 150 flores iris. Para más información sobre estos datos podemos leer la entrada de la wikipedia <https://es.wikipedia.org/wiki/>

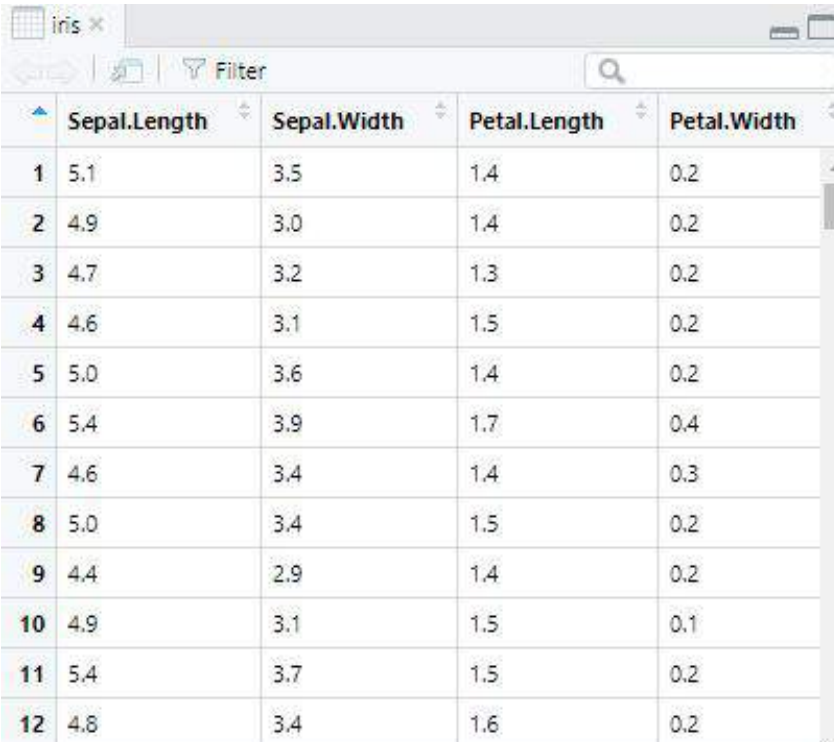
Iris\_flor\_conjunto\_de\_datos, y si se quiere más información específica podéis acceder a la ayuda.

Vamos a proceder con un ejemplo cómo empezar a ver cómo se forma un data frame:

Listing 6.1: Código visualización data frame

```
1 > d.f=iris
2 > data("iris")
3 > View(iris)
```

Con la función `data()` volvemos a la data frame original, con la función `View()` podremos ver nuestras variables en el editor de la siguiente forma:



	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
1	5.1	3.5	1.4	0.2
2	4.9	3.0	1.4	0.2
3	4.7	3.2	1.3	0.2
4	4.6	3.1	1.5	0.2
5	5.0	3.6	1.4	0.2
6	5.4	3.9	1.7	0.4
7	4.6	3.4	1.4	0.3
8	5.0	3.4	1.5	0.2
9	4.4	2.9	1.4	0.2
10	4.9	3.1	1.5	0.1
11	5.4	3.7	1.5	0.2
12	4.8	3.4	1.6	0.2

Figura 6.1: Data frame iris

Además de estas funciones podremos utilizar las siguientes para referirnos a su estructura interna:

- `head`: Muestra las primeras  $n$  filas del data frame.
- `tail`: Muestra las  $n$  últimas filas del data frame.  
Tanto `head` como `tail` tienen por defecto  $n = 6$ .
- `str`: Da la estructura global de un objeto de datos.

Vamos a seguir con el ejemplo anterior mostrando las primeras y las últimas filas:

Listing 6.2: Código mostrar primeras y últimas filas

```
1 > head(d.f)
```



```

2   Sepal.Length Sepal.Width Petal.Length
3   1           5.1         3.5         1.4
4   2           4.9         3.0         1.4
5   3           4.7         3.2         1.3
6   4           4.6         3.1         1.5
7   5           5.0         3.6         1.4
8   6           5.4         3.9         1.7
9   Petal.Width Species
10  1           0.2   setosa
11  2           0.2   setosa
12  3           0.2   setosa
13  4           0.2   setosa
14  5           0.2   setosa
15  6           0.4   setosa
16 > tail(d.f)
17   Sepal.Length Sepal.Width Petal.Length
18  145           6.7         3.3         5.7
19  146           6.7         3.0         5.2
20  147           6.3         2.5         5.0
21  148           6.5         3.0         5.2
22  149           6.2         3.4         5.4
23  150           5.9         3.0         5.1
24   Petal.Width Species
25  145           2.5 virginica
26  146           2.3 virginica
27  147           1.9 virginica
28  148           2.0 virginica
29  149           2.3 virginica
30  150           1.8 virginica
31 > head(d.f,5)
32   Sepal.Length Sepal.Width Petal.Length
33  1           5.1         3.5         1.4
34  2           4.9         3.0         1.4
35  3           4.7         3.2         1.3
36  4           4.6         3.1         1.5
37  5           5.0         3.6         1.4
38   Petal.Width Species
39  1           0.2   setosa
40  2           0.2   setosa
41  3           0.2   setosa
42  4           0.2   setosa
43  5           0.2   setosa
44 > str(d.f)
45 'data.frame':   150 obs. of  5 variables:
46 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
47 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1
...

```

```

48 $ Petal.Length: num  1.4  1.4  1.3  1.5  1.4  1.7  1.4  1.5  1.4
    1.5...
49 $ Petal.Width : num  0.2  0.2  0.2  0.2  0.2  0.4  0.3  0.2  0.2  0.1
    ...
50 $ Species      : Factor w/ 3 levels "setosa","versicolor",...:
    1 1 1 1 1 1 1 1 1 1 ...

```

Seguidamente, vamos a ver cómo obtener sus dimensiones, los nombres de sus variables e identificadores de sus filas.

Vamos a ver cómo obtener los nombres, los identificadores y las dimensiones:

- **names**: Para obtener un vector con los nombres de la columna de un data frame.
- **rownames**: Para obtener un vector con los identificadores de las filas de un data frame.
- **dimnames**: Para obtener una list formada por el vector de los identificadores de las filas y el vector de los nombres de las columnas de un data frame.

Permiten modificar estos vectores:

- **dim**: Para obtener un vector con el número de filas y el número de columnas.

Ahora vamos a aplicar estas nuevas funciones al ejemplo anterior:

Listing 6.3: Código dimensiones data frame

```

1 > names(d.f)
2 [1] "Sepal.Length" "Sepal.Width"
3 [3] "Petal.Length" "Petal.Width"
4 [5] "Species"
5 > rownames(d.f)
6 [1] "1" "2" "3" "4" "5" "6"
7 [7] "7" "8" "9" "10" "11" "12"
8 [13] "13" "14" "15" "16" "17" "18"
9 [19] "19" "20" "21" "22" "23" "24"
10 [25] "25" "26" "27" "28" "29" "30"
11 [31] "31" "32" "33" "34" "35" "36"
12 [37] "37" "38" "39" "40" "41" "42"
13 [43] "43" "44" "45" "46" "47" "48"
14 [49] "49" "50" "51" "52" "53" "54"
15 [55] "55" "56" "57" "58" "59" "60"
16 [61] "61" "62" "63" "64" "65" "66"
17 [67] "67" "68" "69" "70" "71" "72"
18 [73] "73" "74" "75" "76" "77" "78"
19 [79] "79" "80" "81" "82" "83" "84"
20 [85] "85" "86" "87" "88" "89" "90"
21 [91] "91" "92" "93" "94" "95" "96"
22 [97] "97" "98" "99" "100" "101" "102"
23 [103] "103" "104" "105" "106" "107" "108"

```

```

34 [109] "109" "110" "111" "112" "113" "114"
35 [115] "115" "116" "117" "118" "119" "120"
36 [121] "121" "122" "123" "124" "125" "126"
37 [127] "127" "128" "129" "130" "131" "132"
38 [133] "133" "134" "135" "136" "137" "138"
39 [139] "139" "140" "141" "142" "143" "144"
40 [145] "145" "146" "147" "148" "149" "150"
41 > dimnames(d.f)
42 [[1]]
43 [1] "1" "2" "3" "4" "5" "6"
44 [7] "7" "8" "9" "10" "11" "12"
45 [13] "13" "14" "15" "16" "17" "18"
46 [19] "19" "20" "21" "22" "23" "24"
47 [25] "25" "26" "27" "28" "29" "30"
48 [31] "31" "32" "33" "34" "35" "36"
49 [37] "37" "38" "39" "40" "41" "42"
50 [43] "43" "44" "45" "46" "47" "48"
51 [49] "49" "50" "51" "52" "53" "54"
52 [55] "55" "56" "57" "58" "59" "60"
53 [61] "61" "62" "63" "64" "65" "66"
54 [67] "67" "68" "69" "70" "71" "72"
55 [73] "73" "74" "75" "76" "77" "78"
56 [79] "79" "80" "81" "82" "83" "84"
57 [85] "85" "86" "87" "88" "89" "90"
58 [91] "91" "92" "93" "94" "95" "96"
59 [97] "97" "98" "99" "100" "101" "102"
60 [103] "103" "104" "105" "106" "107" "108"
61 [109] "109" "110" "111" "112" "113" "114"
62 [115] "115" "116" "117" "118" "119" "120"
63 [121] "121" "122" "123" "124" "125" "126"
64 [127] "127" "128" "129" "130" "131" "132"
65 [133] "133" "134" "135" "136" "137" "138"
66 [139] "139" "140" "141" "142" "143" "144"
67 [145] "145" "146" "147" "148" "149" "150"
68
69 [[2]]
70 [1] "Sepal.Length" "Sepal.Width"
71 [3] "Petal.Length" "Petal.Width"
72 [5] "Species"
73
74 > dim(d.f)
75 [1] 150 5
76 > d.f$Sepal.Length[1:30]
77 [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8
78 [13] 4.8 4.3 5.8 5.7 5.4 5.1 5.7 5.1 5.4 5.1 4.6 5.1
79 [25] 4.8 5.0 5.0 5.2 5.2 4.7
80 > d.f$Species[1:30]

```

```

81 [1] setosa setosa setosa setosa setosa setosa setosa setosa
82 [8] setosa setosa setosa setosa setosa setosa setosa setosa
83 [15] setosa setosa setosa setosa setosa setosa setosa setosa
84 [22] setosa setosa setosa setosa setosa setosa setosa setosa
85 [29] setosa setosa
86 Levels: setosa versicolor virginica
87 > Sepal.Length
88 Error: object 'Sepal.Length' not found

```

No se puede declarar las variables internas por sí solas, aunque si que se verán más adelante cómo hacerlo.

Los data frame comparten con las matrices la extracción de datos entre corchetes, vamos a ver un ejemplo de esto:

Listing 6.4: Código extracción datos data frame

```

1 > d.f[1:5,]
2   Sepal.Length Sepal.Width Petal.Length Petal.Width
3 1           5.1           3.5           1.4           0.2
4 2           4.9           3.0           1.4           0.2
5 3           4.7           3.2           1.3           0.2
6 4           4.6           3.1           1.5           0.2
7 5           5.0           3.6           1.4           0.2
8   Species
9 1   setosa
10 2   setosa
11 3   setosa
12 4   setosa
13 5   setosa
14 > d.f[1:5,1:3]
15   Sepal.Length Sepal.Width Petal.Length
16 1           5.1           3.5           1.4
17 2           4.9           3.0           1.4
18 3           4.7           3.2           1.3
19 4           4.6           3.1           1.5
20 5           5.0           3.6           1.4
21 > d.f[d.f$Species=="virginica"& d.f$Sepal.Length>7,]
22   Sepal.Length Sepal.Width Petal.Length Petal.Width
23 103           7.1           3.0           5.9           2.1
24 106           7.6           3.0           6.6           2.1
25 108           7.3           2.9           6.3           1.8
26 110           7.2           3.6           6.1           2.5
27 118           7.7           3.8           6.7           2.2
28 119           7.7           2.6           6.9           2.3
29 123           7.7           2.8           6.7           2.0
30 126           7.2           3.2           6.0           1.8
31 130           7.2           3.0           5.8           1.6

```

```

32 131          7.4          2.8          6.1          1.9
33 132          7.9          3.8          6.4          2.0
34 136          7.7          3.0          6.1          2.3
35      Species
36 103 virginica
37 106 virginica
38 108 virginica
39 110 virginica
40 118 virginica
41 119 virginica
42 123 virginica
43 126 virginica
44 130 virginica
45 131 virginica
46 132 virginica
47 136 virginica

```

### 6.3. Importar y exportar data frame

En este capítulo vamos a ver cómo importar un data frame a través de un fichero externo y cómo exportarlos de R.

Primero vamos a ver cómo importar un data frame de un fichero externo, para ello vamos a utilizar la función `read.table` que permite definir un data frame a partir de una tabla de datos contenida en un fichero de texto simple externo. La función completa sería `read.table("ubicación del fichero de texto")`. En esta función podremos utilizar parámetros como:

- **sep**: Para especificar las separaciones entre columnas del fichero, por defecto son espacios en blanco.
- **header**: Si la tabla que importamos tiene una primera fila con los nombres de las columnas hay que especificar `header=TRUE`.

Vamos a realizar un ejemplo con dos data frame, en la primera parte se hará con los ficheros en nuestro directorio y en la segunda parte desde una dirección web:

Listing 6.5: Código importar data frame desde fichero y web

```

1 > NH1=read.table("NotasHermanos.txt", header=TRUE)
2 > head(NH1)
3   Grado Hermanos  Nota
4 1     BQ         1    93
5 2     BL         1    35
6 3     BL         1    55
7 4     BL         1    77
8 5     BL         2    81
9 6     BQ         0    86

```

```

10 > str(NH1)
11 'data.frame': 87 obs. of 3 variables:
12 $ Grado : Factor w/ 2 levels "BL","BQ": 2 1 1 1 1 2 2 1 1 1
    ...
13 $ Hermanos: int 1 1 1 1 2 0 1 1 1 1 ...
14 $ Nota : int 93 35 55 77 81 86 97 40 16 31 ...
15 > NH2=read.table("NotasHermanos.txt", header=TRUE, sep=",")
16 > str(NH2)
17 'data.frame': 87 obs. of 1 variable:
18 $ Grado.Hermanos.Nota: Factor w/ 71 levels "BL 0 20","BL 0 40
    ",...: 62 12 18 24 38 47 63 15 9 11 ...
19 > head(NH2)
20 Grado.Hermanos.Nota
21 1 BQ 1 93
22 2 BL 1 35
23 3 BL 1 55
24 4 BL 1 77
25 5 BL 2 81
26 6 BQ 0 86
27 > NH3=read.table("http://aprender.uib.es/Rdir/NotaHermanos.txt
    ", header=TRUE)
28 > NH4=read.table("http://aprender.uib.es/Rdir/NotaHermanosc.
    txt", header=TRUE, sep=",")
29 > str(NH3)
30 'data.frame': 87 obs. of 3 variables:
31 $ Grado : Factor w/ 2 levels "BL","BQ": 2 1 1 1 1 2 2 1 1 1
    ...
32 $ Hermanos: int 1 1 1 1 2 0 1 1 1 1 ...
33 $ Nota : int 93 35 55 77 81 86 97 40 16 31 ...
34 > head(NH3)
35 Grado Hermanos Nota
36 1 BQ 1 93
37 2 BL 1 35
38 3 BL 1 55
39 4 BL 1 77
40 5 BL 2 81
41 6 BQ 0 86
42 > str(NH4)
43 'data.frame': 87 obs. of 3 variables:
44 $ Grado : Factor w/ 2 levels "BL","BQ": 2 1 1 1 1 2 2 1 1 1
    ...
45 $ Hermanos: int 1 1 1 1 2 0 1 1 1 1 ...
46 $ Nota : int 93 35 55 77 81 86 97 40 16 31 ...
47 > head(NH4)
48 Grado Hermanos Nota

```

51	1	BQ	1	93
52	2	BL	1	35
53	3	BL	1	55
54	4	BL	1	77
55	5	BL	2	81
56	6	BQ	0	86

Además, como hemos dicho en capítulos anteriores también podremos importar nuestros ficheros desde la zona de entornos a través de [Import Dataset](#).

Finalmente, podremos exportar un data frame a un fichero externo a través de la función `write.table("data frametitle=nombre del fichero")`. Esta función tiene algunos parámetros como:

- **sep**: Para especificar las separaciones entre columnas del fichero, por defecto son espacios en blanco.
- **dec**: Para indicar el separador decimal.

## 6.4. Crear data frame

En este capítulo veremos como crear data frame sin necesidad de importar de un fichero externo.

Para ello vamos a utilizar la función `data.frame` aplicada a los vectores que serán las columnas. Esta función tiene los siguientes parámetros:

- **row.names**: Para especificar los identificadores de las filas.
- **stringsAsFactors**: Con `stringsAsFactors=FALSE`, imponemos que los vectores de palabra se mantengan como tales en el data frame, por defecto se crean factores.

Vamos a proceder a crear nuestro data frame con las siguientes variables:

- Sexo del estudiante.
- Edad (en años) del estudiante.
- Número de hermanos.

Listing 6.6: Código de creación data frame

```

1 > Sexo=c("Hombre", "Hombre", "Mujer", "Hombre", "Hombre", "
      Hombre", "Mujer")
2 > Edad=c(17,18,20,18,18,18,19)
3 > Hermanos=c(2,0,0,1,1,1,0)
4 > df.1=data.frame(Sexo, Edad, Hermanos)
5 > dfbk.1=df.1
6 > str(df.1)
7 'data.frame':  7 obs. of  3 variables:

```

```
8 $ Sexo      : Factor w/ 2 levels "Hombre","Mujer": 1 1 2 1 1 1
  2
9 $ Edad      : num  17 18 20 18 18 18 19
10 $ Hermanos: num  2 0 0 1 1 1 0
11 > df.2=data.frame(Sexo,Edad,Hermanos,stringsAsFactors = FALSE,
  row.names = c("E1","E2","E3","E4","E5","E6","E7"))
12 > df.2
13      Sexo Edad Hermanos
14 E1 Hombre  17         2
15 E2 Hombre  18         0
16 E3 Mujer   20         0
17 E4 Hombre  18         1
18 E5 Hombre  18         1
19 E6 Hombre  18         1
20 E7 Mujer   19         0
21 > str(df.2)
22 'data.frame':  7 obs. of  3 variables:
23 $ Sexo      : chr  "Hombre" "Hombre" "Mujer" "Hombre" ...
24 $ Edad      : num  17 18 20 18 18 18 19
25 $ Hermanos: num  2 0 0 1 1 1 0
26 > df.2[c("E1","E2"),]
27      Sexo Edad Hermanos
28 E1 Hombre  17         2
29 E2 Hombre  18         0
```

Finalmente, podremos editar estos datos desde el editor de datos a través de la función `fix()`, que la veremos en el siguiente ejemplo:

Listing 6.7: Código función `fix`

```
1 > fix(df.2)
```

Una vez la hayamos puesto veremos lo siguiente:



	row.names	Sexo	Edad	Hermanos	var5	var6	var7
1	E1	Hombre	17	2			
2	E2	Hombre	18	0			
3	E3	Mujer	20	0			
4	E4	Hombre	18	1			
5	E5	Hombre	18	1			
6	E6	Hombre	18	1			
7	E7	Mujer	19	0			
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							

Figura 6.2: Editor de datos

En esta ventana podremos añadir nuevas variables o modificar las que tenemos a nuestro dataframe.

## 6.5. Cómo modificar un data frame

En este capítulo veremos cómo modificar un data frame ya creado o importado. Podremos modificar el data frame a través de las siguientes funciones:

- Cambiar todos los nombres de las variables: `names(data frame)`=vector con los nombres de las variables.
- Modificar los identificadores de las filas: `rownames(data frame)`= vector con los nombres de las filas.
- Modificar los nombres de las filas y de las columnas simultáneamente: `dimnames(data frame)`= list (vector con los nombres de las filas, vector con los nombres de las columnas).

Vamos a aplicar estas modificaciones al ejemplo anterior:

Listing 6.8: Código modificadores data frame

```
1 > df.1
```

```

2      Sexo Edad Hermanos
3 1 Hombre 17 2
4 2 Hombre 18 0
5 3 Mujer 20 0
6 4 Hombre 18 1
7 5 Hombre 18 1
8 6 Hombre 18 1
9 7 Mujer 19 0
10 > str(df.1)
11 'data.frame': 7 obs. of 3 variables:
12 $ Sexo : Factor w/ 2 levels "Hombre","Mujer": 1 1 2 1 1 1 1
13 $ Edad : num 17 18 20 18 18 18 19
14 $ Hermanos: num 2 0 0 1 1 1 0
15 > names(df.1)=c("S","E","H")
16 > str(df.1)
17 'data.frame': 7 obs. of 3 variables:
18 $ S: Factor w/ 2 levels "Hombre","Mujer": 1 1 2 1 1 1 2
19 $ E: num 17 18 20 18 18 18 19
20 $ H: num 2 0 0 1 1 1 0
21 > names(df.1)[2]="Edad"
22 > names(df.1)[names(df.1)=="E"]="Edad"
23 > str(df.1)
24 'data.frame': 7 obs. of 3 variables:
25 $ S : Factor w/ 2 levels "Hombre","Mujer": 1 1 2 1 1 1 2
26 $ Edad: num 17 18 20 18 18 18 19
27 $ H : num 2 0 0 1 1 1 0
28 > rownames(df.1)=paste("Alumno", 1:7, sep="_")
29 > df.1
30      S Edad H
31 Alumno_1 Hombre 17 2
32 Alumno_2 Hombre 18 0
33 Alumno_3 Mujer 20 0
34 Alumno_4 Hombre 18 1
35 Alumno_5 Hombre 18 1
36 Alumno_6 Hombre 18 1
37 Alumno_7 Mujer 19 0
38 > dimnames(df.1)=list(c("I","II","III","IV","V","VI","VII"),
39 c("V1","V2","V3"))
40 > df.1
41      V1 V2 V3
42 I Hombre 17 2
43 II Hombre 18 0
44 III Mujer 20 0
45 IV Hombre 18 1
46 V Hombre 18 1
47 VI Hombre 18 1

```

```

47 VII Mujer 19 0
48 > df.1[1,2]="Joven"
49 > str(df.1)
50 'data.frame': 7 obs. of 3 variables:
51 $ V1: Factor w/ 2 levels "Hombre","Mujer": 1 1 2 1 1 1 2
52 $ V2: chr "Joven" "18" "20" "18" ...
53 $ V3: num 2 0 0 1 1 1 0
54 > df.1[1,2]=17
55 > str(df.1)
56 'data.frame': 7 obs. of 3 variables:
57 $ V1: Factor w/ 2 levels "Hombre","Mujer": 1 1 2 1 1 1 2
58 $ V2: chr "17" "18" "20" "18" ...
59 $ V3: num 2 0 0 1 1 1 0

```

Como podemos ver en la última parte, una vez que cambiamos el tipo de variable, toda la columna se queda con el mismo tipo de variable. Para evitarlo, podemos modificar la columna entera con la función `data.frame$variable=nuevo valor` ó cambiar el tipo de objeto con funciones tipo `as.tipo_de_objeto`, que son las siguientes:

- `as.factor`.
- `as.character`.
- `as.integer`.
- `as.numeric`.

Finalmente, vamos a transformar la variable de la columna 2 de palabra a valor numérico con el siguiente ejemplo:

Listing 6.9: Código modificar variable

```

1 > df.1$V2=as.numeric(df.1$V2)
2 > str(df.1)
3 'data.frame': 7 obs. of 3 variables:
4 $ V1: Factor w/ 2 levels "Hombre","Mujer": 1 1 2 1 1 1 2
5 $ V2: num 17 18 20 18 18 18 19
6 $ V3: num 2 0 0 1 1 1 0

```

## 6.6. Cómo añadir filas y columnas a un data frame

En este capítulo añadiremos nuevas observaciones y nuevas variables a nuestro data frame. Primero vamos a ver cómo añadir filas a nuestro data frame:

- a). Crear un nuevo data frame con los mismos nombres de las variables.
- b). Concatenar las filas usando `rbind`.

A continuación vamos a añadirle filas al ejercicio anterior:

Listing 6.10: Código añadir filas

```

1 > Sexo=c("Hombre","Hombre","Mujer","Hombre","Hombre","
      Hombre","Mujer")
2 > Edad=c(17,18,20,18,18,18,19)
3 > Hermanos=c(2,0,0,1,1,1,0)
4 > df.1=data.frame(Sexo,Edad,Hermanos)
5 > dfbk.1=df.1
6 > df.1
7      Sexo Edad Hermanos
8 1 Hombre  17         2
9 2 Hombre  18         0
10 3  Mujer  20         0
11 4 Hombre  18         1
12 5 Hombre  18         1
13 6 Hombre  18         1
14 7  Mujer  19         0
15 > str(df.1)
16 'data.frame':  7 obs. of  3 variables:
17 $ Sexo      : Factor w/ 2 levels "Hombre","Mujer": 1 1 2 1 1 1
18              2
19 $ Edad      : num  17 18 20 18 18 18 19
20 $ Hermanos : num  2 0 0 1 1 1 0
21 > nuevas.filas=data.frame(Sexo=c("Hombre","Hombre"), Edad=c
22      (18,18),Hermanos=c(1,2))
23 > df.1=rbind(df.1,nuevas.filas)
24 > df.1
25      Sexo Edad Hermanos
26 1 Hombre  17         2
27 2 Hombre  18         0
28 3  Mujer  20         0
29 4 Hombre  18         1
30 5 Hombre  18         1
31 6 Hombre  18         1
32 7  Mujer  19         0
33 8 Hombre  18         1
34 9 Hombre  18         2

```

Finalmente, vamos a ver cómo añadir columnas a nuestro data frame, para ello tendremos dos opciones:

- Especificar el valor de la columna: `data.frame$Nueva=` nuevos valores.
- Usando la función `cbind` a una variable con la misma longitud de variables que el data frame.

Vamos a volver a aplicar esta modificación al ejemplo anterior:

Listing 6.11: Código modificador columnas

```

1 > df.1$Nueva=df.1$Edad*df.1$Hermanos
2 > df.1
3      Sexo Edad Hermanos Nueva
4 1 Hombre  17         2    34
5 2 Hombre  18         0     0
6 3  Mujer  20         0     0
7 4 Hombre  18         1    18
8 5 Hombre  18         1    18
9 6 Hombre  18         1    18
10 7  Mujer  19         0     0
11 8 Hombre  18         1    18
12 9 Hombre  18         2    36
13 > df.1=cbind(df.1,Grado=rep("Biologia",9))
14 > df.1
15      Sexo Edad Hermanos Nueva  Grado
16 1 Hombre  17         2    34 Biologia
17 2 Hombre  18         0     0 Biologia
18 3  Mujer  20         0     0 Biologia
19 4 Hombre  18         1    18 Biologia
20 5 Hombre  18         1    18 Biologia
21 6 Hombre  18         1    18 Biologia
22 7  Mujer  19         0     0 Biologia
23 8 Hombre  18         1    18 Biologia
24 9 Hombre  18         2    36 Biologia

```

## 6.7. Cómo seleccionar trozos de un data frame

En este capítulo vamos a ver cómo seleccionar datos de un data frame para crear nuevas subtablas. Como en el caso de las matrices, vamos a ver la selección de filas como de columnas.

Para seleccionar datos de un data frame haremos lo siguiente:

- Índices de las filas y columnas.
- Nombre de las filas y columnas.

Siempre dentro de corchetes.

Listing 6.12: Código selección de trozos

```

1 > dfbk.1
2      Sexo Edad Hermanos
3 1 Hombre  17         2
4 2 Hombre  18         0
5 3  Mujer  20         0
6 4 Hombre  18         1
7 5 Hombre  18         1
8 6 Hombre  18         1

```

```
8 7 Mujer 19 0
9 > str(dfbk.1)
10 'data.frame': 7 obs. of 3 variables:
11 $ Sexo : Factor w/ 2 levels "Hombre","Mujer": 1 1 2 1 1 1
12 $ Edad : num 17 18 20 18 18 18 19
13 $ Hermanos: num 2 0 0 1 1 1 0
14 > dfbk.1[,c(1,2)]
15 Sexo Edad
16 1 Hombre 17
17 2 Hombre 18
18 3 Mujer 20
19 4 Hombre 18
20 5 Hombre 18
21 6 Hombre 18
22 7 Mujer 19
23 > dfbk.1[, -3]
24 Sexo Edad
25 1 Hombre 17
26 2 Hombre 18
27 3 Mujer 20
28 4 Hombre 18
29 5 Hombre 18
30 6 Hombre 18
31 7 Mujer 19
32 > dfbk.1[,c("Sexo","Edad")]
33 Sexo Edad
34 1 Hombre 17
35 2 Hombre 18
36 3 Mujer 20
37 4 Hombre 18
38 5 Hombre 18
39 6 Hombre 18
40 7 Mujer 19
41 > df.2=dfbk.1[,c("Edad","Hermanos","Sexo")]
42 > df.2
43 Edad Hermanos Sexo
44 1 17 2 Hombre
45 2 18 0 Hombre
46 3 20 0 Mujer
47 4 18 1 Hombre
48 5 18 1 Hombre
49 6 18 1 Hombre
50 7 19 0 Mujer
51 > df.2[1:2,]
52 Edad Hermanos Sexo
53 1 17 2 Hombre
```

```

55 2    18        0 Hombre
56 > df.2 [df.2$Edad<19,]
57   Edad Hermanos  Sexo
58 1    17         2 Hombre
59 2    18         0 Hombre
60 4    18         1 Hombre
61 5    18         1 Hombre
62 6    18         1 Hombre
63 > df.2 [df.2$Hermanos<19 & df.2$Hermanos==1,]
64   Edad Hermanos  Sexo
65 4    18         1 Hombre
66 5    18         1 Hombre
67 6    18         1 Hombre
68 > df.18=df.2 [df.2$Hermanos<19 & df.2$Hermanos==1,]
69 > str(df.18)
70 'data.frame':   3 obs. of  3 variables:
71 $ Edad      : num  18 18 18
72 $ Hermanos : num  1 1 1
73 $ Sexo      : Factor w/ 2 levels "Hombre","Mujer": 1 1 1
74 > df.18=droplevels(df.18)
75 > str(df.18)
76 'data.frame':   3 obs. of  3 variables:
77 $ Edad      : num  18 18 18
78 $ Hermanos : num  1 1 1
79 $ Sexo      : Factor w/ 1 level "Hombre": 1 1 1

```

Es importante decir, que en este ejemplo hemos elegido el data frame de backup, para no volver a tener que escribir el mismo ejemplo otra vez.

Por otro lado, si queremos ampliar las posibilidades para especificar las variables que queremos extraer deberemos instalar primero el paquete [dplyr](#) y de ahí utilizar la función [select](#), que realiza lo siguiente:

- `select(data frame, starst_with("x"))`: Extrae del data frame las variables cuyo nombre empieza con la palabra x.
- `select(data frame, ends_with("x"))`: Extrae del data frame las variables cuyo nombre termina con la palabra x.
- `select(data frame, contains("x"))`: Extrae del data frame las variables cuyo nombre contiene en algún sitio la palabra x.

Vamos a ver un ejemplo de uso de esta función:

Listing 6.13: Código de la función `select`

```

1 > detach("package:dplyr", unload = TRUE)
2 > library(dplyr)
3
4 Attaching package: "dplyr"

```

```

5
6 The following objects are masked from 'package:'stats:
7
8     filter , lag
9
10 The following objects are masked from 'package:'base:
11
12     intersect , setdiff , setequal , union
13
14 > iris_Petal=select(iris , starts_with("Petal"))
15 > head(iris_Petal)
16   Petal.Length Petal.Width
17 1           1.4           0.2
18 2           1.4           0.2
19 3           1.3           0.2
20 4           1.5           0.2
21 5           1.4           0.2
22 6           1.7           0.4
23 > iris_Lenght=select(iris , ends_with("Length"))
24 > head(iris_Lenght)
25   Sepal.Length Petal.Length
26 1           5.1           1.4
27 2           4.9           1.4
28 3           4.7           1.3
29 4           4.6           1.5
30 5           5.0           1.4
31 6           5.4           1.7

```

Finalmente, además de estas funciones tendremos la función `subset(data frame, condición, select=columnas)` que extrae del data frame las filas que cumplen la condición y las columnas especificadas en el parámetro select. Para seleccionarlas:

- Todas las filas: No se especifica ninguna condición.
- Todas las columnas: No hace falta especificar el parámetro select.

A continuación vamos a realizar un ejemplo con esta función:

Listing 6.14: Código función subset

```

1 > iris_vir=subset(iris , Species=="virginica")
2 > head(iris_vir)
3   Sepal.Length Sepal.Width Petal.Length Petal.Width
4 101           6.3           3.3           6.0           2.5
5 102           5.8           2.7           5.1           1.9
6 103           7.1           3.0           5.9           2.1
7 104           6.3           2.9           5.6           1.8
8 105           6.5           3.0           5.8           2.2
9 106           7.6           3.0           6.6           2.1

```



```

10      Species
11 101 virginica
12 102 virginica
13 103 virginica
14 104 virginica
15 105 virginica
16 106 virginica

```

## 6.8. Cómo aplicar una función a un data frame

En este capítulo vamos a ver cómo aplicar funciones a las variables de un data frame, a sus columnas y veremos algunos ejemplos de aplicación.

La mejor forma de aplicar una función a todas las columnas de un data frame es la función `sapply(data frame, FUN=función, na.rm=TRUE)`. Por defecto, `na.rm=FALSE` pero es conveniente añadirlo para evitar obtener NA si en alguna posición del data frame hay NA.

Vamos a aplicar esta función en el data frame del iris:

Listing 6.15: Código función sapply

```

1 > str(iris)
2 'data.frame': 150 obs. of 5 variables:
3 $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
4 $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1
5 ...
6 $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5
7 ...
8 $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1
9 ...
10 $ Species : Factor w/ 3 levels "setosa","versicolor",...:
11 1 1 1 1 1 1 1 1 1 1 ...
12 > sapply(iris[,1:4],FUN=mean)
13 Sepal.Length Sepal.Width Petal.Length Petal.Width
14 5.843333 3.057333 3.758000 1.199333
15 > sapply(iris[,1:4],FUN=sum)
16 Sepal.Length Sepal.Width Petal.Length Petal.Width
17 876.5 458.6 563.7 179.9
18 > f=function(x){sqrt(sum(x^2))}
19 > sapply(iris[,1:4],FUN=f)
20 Sepal.Length Sepal.Width Petal.Length Petal.Width
21 72.27621 37.82063 50.82037 17.38764
22 > D=data.frame(V1=c(1,2,NA,3),V2=c(2,5,2,NA))
23 > D
24 V1 V2
25 1 1 2
26 2 2 5

```

```

23 3 NA 2
24 4 3 NA
25 > sapply(D,FUN=mean)
26 V1 V2
27 NA NA
28 > sapply(D,FUN=mean, na.rm=TRUE)
29 V1 V2

```

Por otro lado, la función `aggregate` permite aplicar una función a variables de un data frame clasificadas por los niveles de un, o más de un factor. Para ello utilizaremos la función `aggregate(variable~factor, data=data frame, FUN=función)`, que realiza lo siguiente:

- Para aplicar la función a más de una variable: `cbind(variable1,variable2,...)~factor`.
- Para separar las variables mediante más de un factor: `variable~factor1+factor2+...`

El resultado será un data frame.

Vamos a realizar un ejemplo realizando la media de cada una de las medias de las longitudes de los pétalos:

Listing 6.16: Código función `aggregate`

```

1 > aggregate(Petal.Length~Species, data=iris, FUN=mean, na.rm=
  TRUE)
2   Species Petal.Length
3 1   setosa      1.462
4 2 versicolor  4.260
5 3  virginica  5.552
6 > aggregate(cbind(Petal.Length, Petal.Width)~Species, data=
  iris, FUN=mean, na.rm=TRUE)
7   Species Petal.Length Petal.Width
8 1   setosa      1.462      0.246
9 2 versicolor  4.260      1.326
10 3  virginica  5.552      2.026

```

Seguidamente, vamos a realizar otro ejemplo con el paquete `alr4` que contiene la tabla de datos **Rateprof**, con los resultados globales de la evaluación de un grupo de profesores universitarios por parte de sus estudiantes.

Algunas variables:

- **gender**: Sexo del profesor.
- **pepper**: Indica si las encuestas se le ha considerado mayoritariamente atractivo o no.
- **clarity**: Valora la claridad de exposición.
- **easiness**: Valora la accesibilidad del profesor.

Vamos a calcular la media de estas dos últimas variables poniéndolas al lado de atractivo.

Listing 6.17: Código ejemplo paquete alr4

```

1 > library(alr4)
2 > aggregate(cbind(clarity , easiness)~gender+pepper , data=
  Rateprof , FUN=mean)
3   gender pepper  clarity easiness
4 1 female     no  3.341391 3.147606
5 2  male     no  3.451456 2.999056
6 3 female     yes 4.345082 3.599128
7 4  male     yes 4.371824 3.689741

```

## 6.9. Cómo añadir las variables de un data frame al entorno global

En este capítulo veremos como definir las variables de un data frame en un entorno global de R para poder acceder a ellas escribiendo únicamente su nombre.

Para referirnos a una variable de un data frame solemos escribir `data frame$variable`, pero tenemos dos funciones para acceder a ellas.

Por un lado la función `attach` que nos permite definir sus variables globales y referirnos a ellas únicamente con su nombre, para utilizarla usaríamos `attach(data frame)`.

Por otro lado, tenemos la función `detach` que devuelve a la situación original, eliminando del entorno global las variables del data frame, para utilizarla usaríamos `detach (data frame)`.

Como ejemplo, vamos añadir las variables de iris al entorno global de R:

Listing 6.18: Código añadir variables de un data frame al entorno global

```

1 > Petal.Lenght
2 Error: object 'Petal.Lenght' not found
3 > attach(iris)
4 > Petal.Length
5   [1] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 1.5 1.6
6   [13] 1.4 1.1 1.2 1.5 1.3 1.4 1.7 1.5 1.7 1.5 1.0 1.7
7   [25] 1.9 1.6 1.6 1.5 1.4 1.6 1.6 1.5 1.5 1.4 1.5 1.2
8   [37] 1.3 1.4 1.3 1.5 1.3 1.3 1.3 1.6 1.9 1.4 1.6 1.4
9   [49] 1.5 1.4 4.7 4.5 4.9 4.0 4.6 4.5 4.7 3.3 4.6 3.9
10  [61] 3.5 4.2 4.0 4.7 3.6 4.4 4.5 4.1 4.5 3.9 4.8 4.0
11  [73] 4.9 4.7 4.3 4.4 4.8 5.0 4.5 3.5 3.8 3.7 3.9 5.1
12  [85] 4.5 4.5 4.7 4.4 4.1 4.0 4.4 4.6 4.0 3.3 4.2 4.2
13  [97] 4.2 4.3 3.0 4.1 6.0 5.1 5.9 5.6 5.8 6.6 4.5 6.3
14 [109] 5.8 6.1 5.1 5.3 5.5 5.0 5.1 5.3 5.5 6.7 6.9 5.0
15 [121] 5.7 4.9 6.7 4.9 5.7 6.0 4.8 4.9 5.6 5.8 6.1 6.4
16 [133] 5.6 5.1 5.6 6.1 5.6 5.5 4.8 5.4 5.6 5.1 5.1 5.9
17 [145] 5.7 5.2 5.0 5.2 5.4 5.1
18 > Sepal.Width=rep(0,150)
19 > Sepal.Width

```

```
20 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
21 [25] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
22 [49] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
23 [73] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
24 [97] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
25 [121] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
26 [145] 0 0 0 0 0 0
27 > head(iris)
28 Sepal.Length Sepal.Width Petal.Length Petal.Width
29 1           5.1           3.5           1.4           0.2
30 2           4.9           3.0           1.4           0.2
31 3           4.7           3.2           1.3           0.2
32 4           4.6           3.1           1.5           0.2
33 5           5.0           3.6           1.4           0.2
34 6           5.4           3.9           1.7           0.4
35 Species
36 1 setosa
37 2 setosa
38 3 setosa
39 4 setosa
40 5 setosa
41 6 setosa
42 > detach(iris)
43 > Petal.Length
44 Error: object 'Petal.Length' not found
```

# Capítulo 7

## Función plot

En este capítulo veremos la función `plot` que se encarga de representar en un gráfico los distintos datos.

La función `plot` es una función genérica de R para dibujar gráficos sencillos en los ejes  $XY$ . Tiene los siguientes usos:

- `plot(x,y)`: Dibuja la familia de puntos  $(x_1, y_1), \dots, (x_n, y_n)$  donde  $x = (x_1, \dots, x_n)$  e  $y = (y_1, \dots, y_n)$ .
- `plot(x)`: Dibuja la familia de puntos  $(1, x_1), \dots, (1, x_n)$  donde  $x = (x_1, \dots, x_n)$ .
- `plot(f)`: Dibuja el gráfico de la función  $f$  definida mediante `function`.

Vamos a ver un ejemplo de cada uso de la función `plot`:

Listing 7.1: Código usos función plot

```
1 > #(1,1) ,(2,3) ,(3,5) ,(4,1) ,(7,8)
2 > x=c(1,2,3,4,7)
3 > y=c(1,3,5,1,8)
4 > plot(x,y)
```

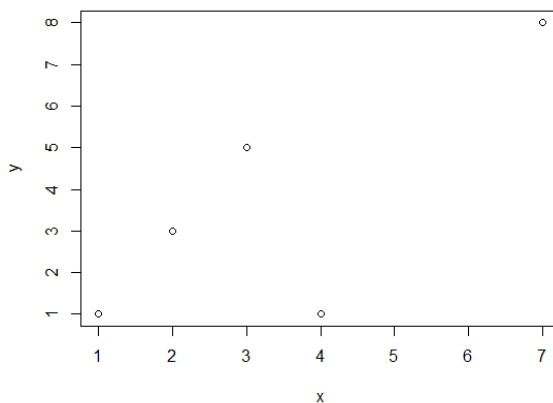


Figura 7.1: Función `plot(x,y)`

Podremos verlo más de cerca con la opción [Zoom](#) o podremos guardarlo como imagen con la opción [Export](#).

Listing 7.2: Código usos función plot

```
1 > #(1,1) ,(2,3) ,(3,5) ,(4,1) ,(7,8)
2 > x=c(1,2,3,4,7)
3 > y=c(1,3,5,1,8)
4 > plot(x)
5 > 1:5
6 [1] 1 2 3 4 5
7 > x
8 [1] 1 2 3 4 7
```

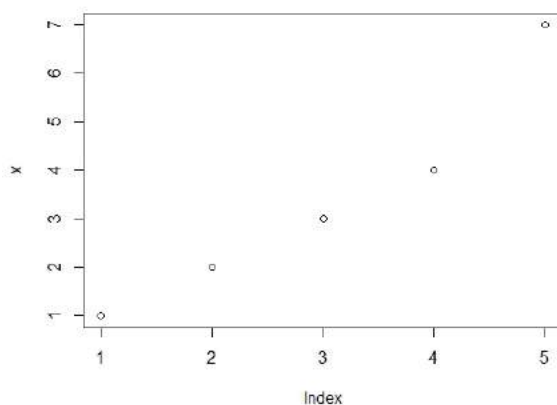


Figura 7.2: Función plot(x)

Listing 7.3: Código usos función plot

```
1 > f=function(x){sqrt(x)}
2 > plot(f)
```

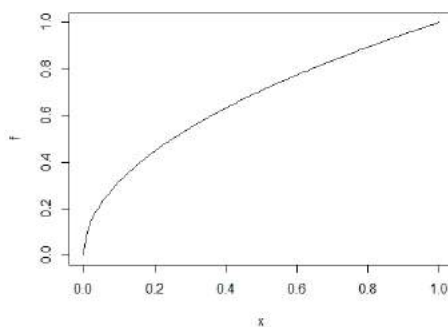


Figura 7.3: Función plot(f)

## 7.1. Parámetros

En este capítulo veremos los parámetros para modificar el gráfico y ajustarlo a nuestras necesidades.

Disponemos de unos tipos de parámetros:

- **Aspecto exterior:** Especifican el título y los nombres de los ejes de coordenadas.
- **Aspecto de los puntos:** Especifican el estilo, el tamaño y el color de los puntos.
- **Tipo de gráfico:** Puntos aislados, unidos por líneas, etc.
- **Ejes de coordenadas:** Especifican los rangos de los ejes y las posiciones de las marcas.

Podemos ver más parámetros a través de la ayuda con la función `plot`.

### 7.1.1. Parámetros de aspecto exterior

- `main`: Especifica el título.
- `xlab`, `ylab`: Especifican las etiquetas de los ejes de coordenadas. En caso de usar una fórmula matemática, hay que usar `expression`.

En ocasiones, es posible que el título o nombres de los ejes queden fuera de los márgenes del gráfico. Para modificarlos, se tiene que ejecutar (antes del `plot`).

- `par`: Modifica el aspecto general del gráfico.
- `mar`: Modifica los márgenes del gráfico.

Vamos a ejecutar una gráfica sencilla modificando su aspecto exterior:

Listing 7.4: Código modificación aspecto exterior

```
1 > f=function(x){x^3}
2 > plot(f)
3 > plot(f, main="Gráfico de la función cubo",ylab = "f(x)=x^3")
4 > viejo.par=par()
5 > par(mar=c(5,4,4,2)+0.5)
6 > plot(f,main="Gráfico de la función cubo",ylab=expression(f(x)
  )=x^3))
```

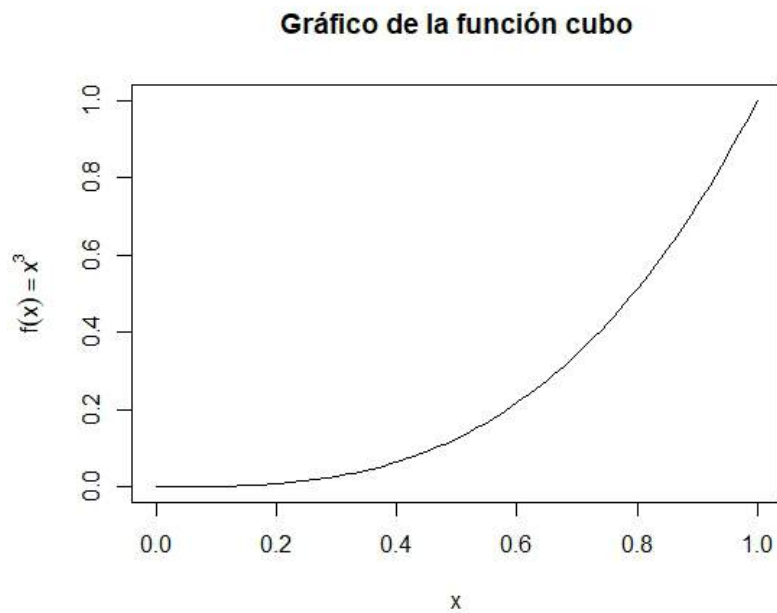


Figura 7.4: Figura modificada márgenes y aspecto exterior

### 7.1.2. Parámetros de aspecto de los puntos

Los parámetros de los puntos se pueden modificar a través de las siguientes funciones:

- `pch`: Especifica el estilo de los puntos.



Figura 7.5: Número con estilo de punto

- `cex`: Especifica el tamaño de los puntos en relación al tamaño por defecto.
- `col`: Especifica el color del borde de los puntos. En el caso de usar puntos con estilo `pch` de 21 a 25, se puede especificar un color de relleno con `bg`.

Vamos a realizar un nuevo ejemplo modificando sus puntos:

Listing 7.5: Código modificación aspecto de los puntos

```

1 > x=c(1,3,6,7)
2 > y=c(2,3,4,1)
3 > plot(x,y)
4 > plot(x,y, pch=2, col="red", cex=2)

```



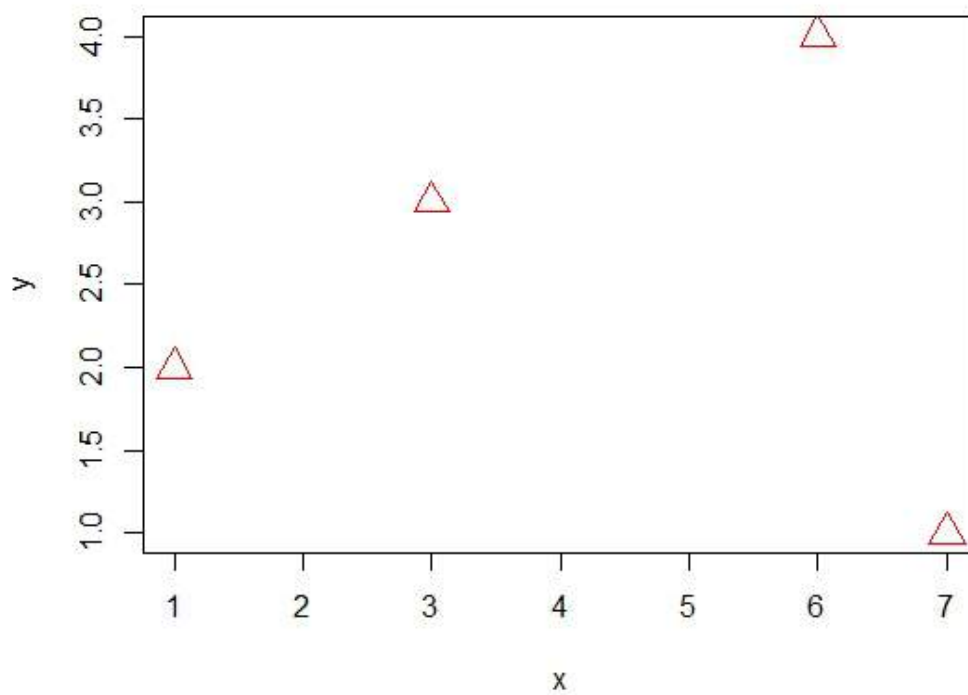


Figura 7.6: Primera modificación puntos

Seguimos con el mismo ejemplo, pero esta vez les pondremos borde negro y los rellenaremos de algún color:

Listing 7.6: Código modificación aspecto de los puntos

```
1 > plot(x,y, pch=24,col="black",bg="green",cex=2)
```

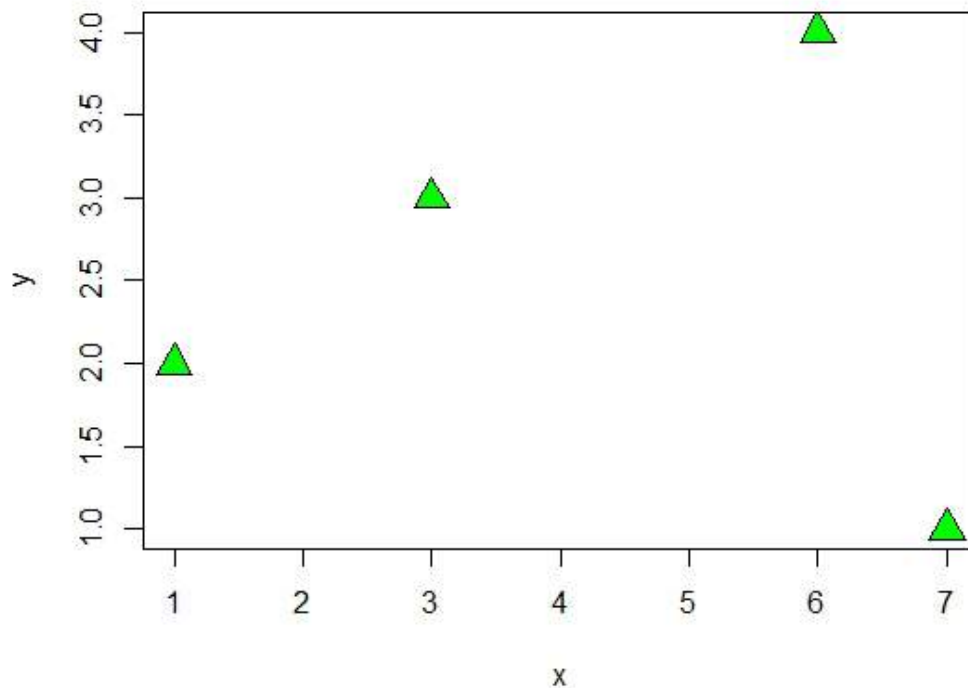


Figura 7.7: Segunda modificación puntos

### 7.1.3. Parámetros de tipo de gráfico

- **type**: Indica el tipo de gráfico.
  - **"p"**: Puntos como simples puntos. Valor por defecto.
  - **"l"**: Puntos unidos por líneas rectas. No se ven los puntos.
  - **"b"**: Puntos unidos por líneas rectas. Se ven los puntos, las líneas no entran en los signos de los puntos.
  - **"o"**: Como "b" pero ahora las líneas sí entran.
  - **"h"**: Dibuja un histograma de líneas.
  - **"s"**: Dibuja un diagrama de escalones.
  - **"n"**: Sólo dibuja el exterior del gráfico.

Vamos a proceder a realizar un ejemplo modificando el tipo de gráfico:

Listing 7.7: Código modificando el tipo de gráfico

```
1 > x=c(1,3,6,7)
2 > y=c(2,3,4,1)
3 > plot(x,y, type = "p")
```

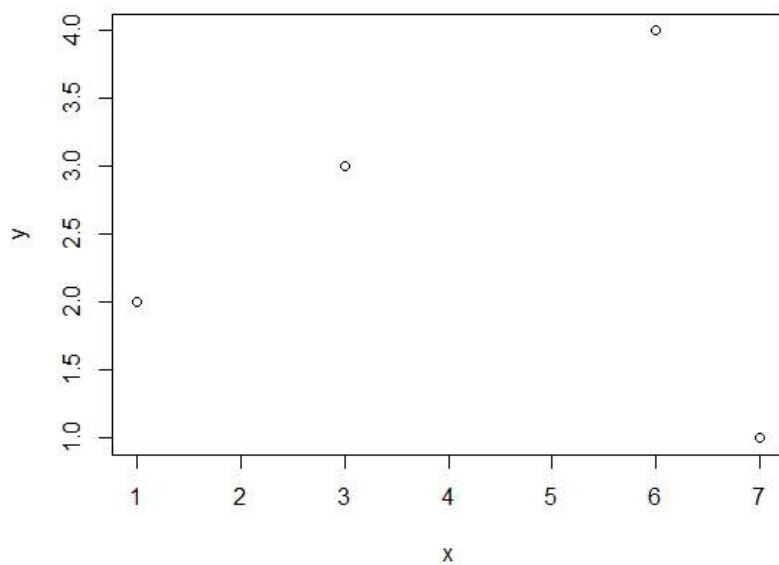


Figura 7.8: Primera modificación del gráfico

Listing 7.8: Código modificando el tipo de gráfico

```
1 > x=c(1,3,6,7)
2 > y=c(2,3,4,1)
3 > plot(x,y, type = "l")
```

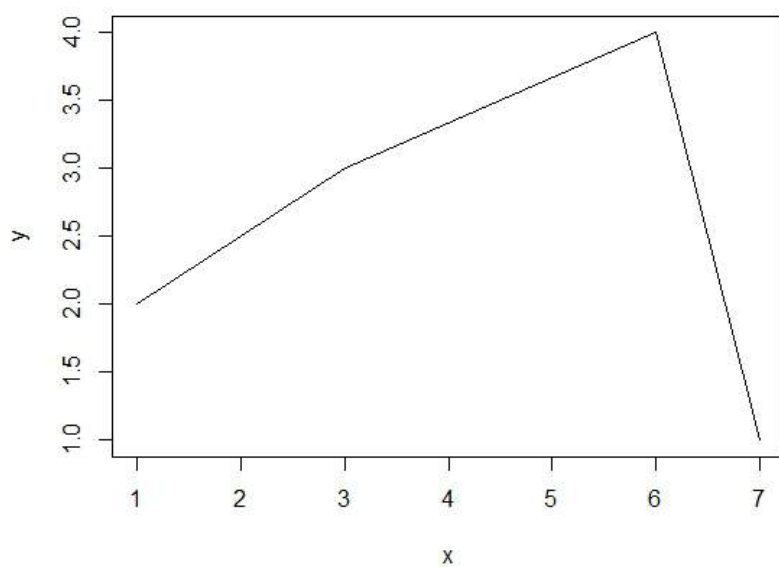


Figura 7.9: Segunda modificación del gráfico

Listing 7.9: Código modificando el tipo de gráfico

```
1 > x=c(1,3,6,7)
2 > y=c(2,3,4,1)
3 > plot(x,y, type = "b")
```

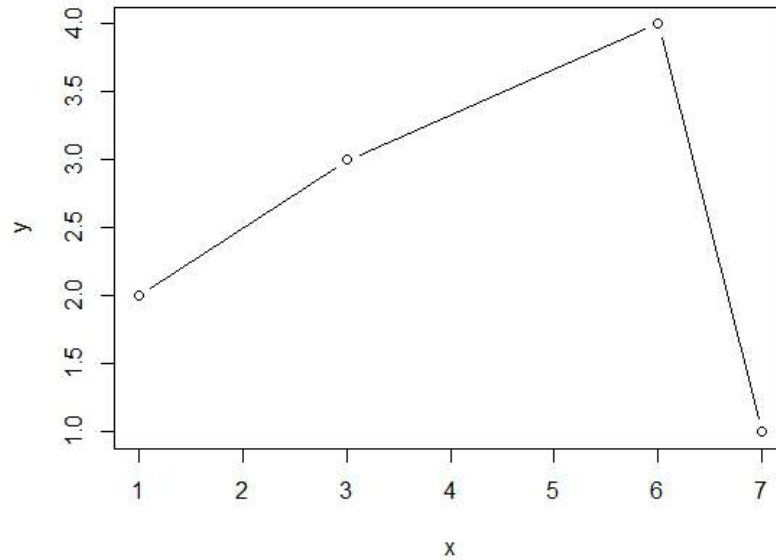


Figura 7.10: Tercera modificación del gráfico

Listing 7.10: Código modificando el tipo de gráfico

```
1 > x=c(1,3,6,7)
2 > y=c(2,3,4,1)
3 > plot(x,y, type = "o")
```

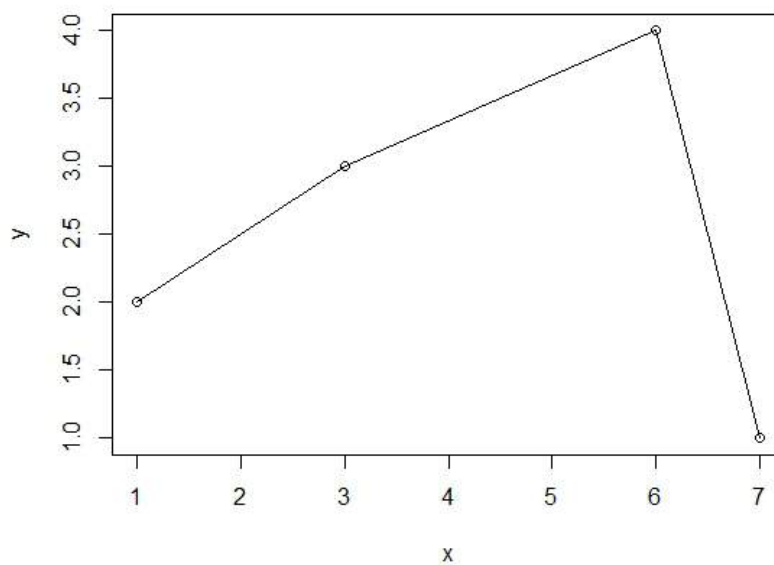


Figura 7.11: Cuarta modificación del gráfico

Listing 7.11: Código modificando el tipo de gráfico

```
1 > x=c(1,3,6,7)
2 > y=c(2,3,4,1)
3 plot(x,y, type = "h")
```

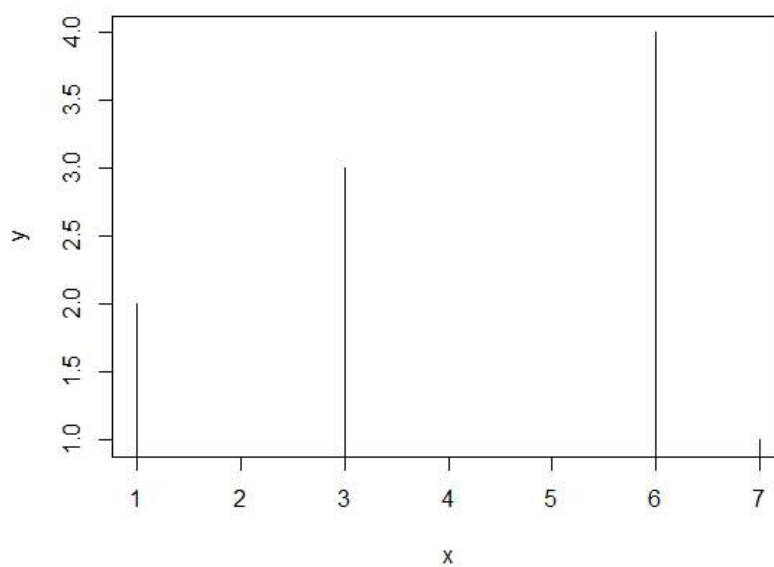


Figura 7.12: Quinta modificación del gráfico

Listing 7.12: Código modificando el tipo de gráfico

```
1 > x=c(1,3,6,7)
2 > y=c(2,3,4,1)
3 plot(x,y, type = "s")
```

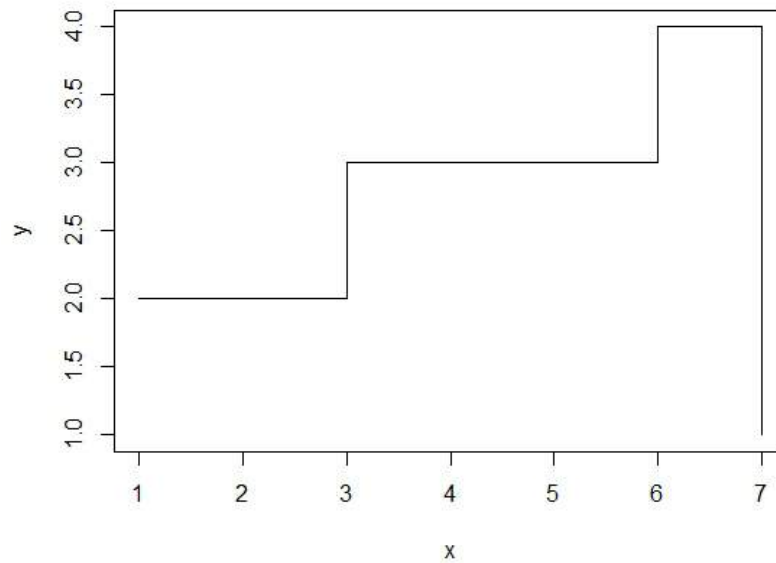


Figura 7.13: Sexta modificación del gráfico

Listing 7.13: Código modificando el tipo de gráfico

```
1 > x=c(1,3,6,7)
2 > y=c(2,3,4,1)
3 plot(x,y, type = "n")
```

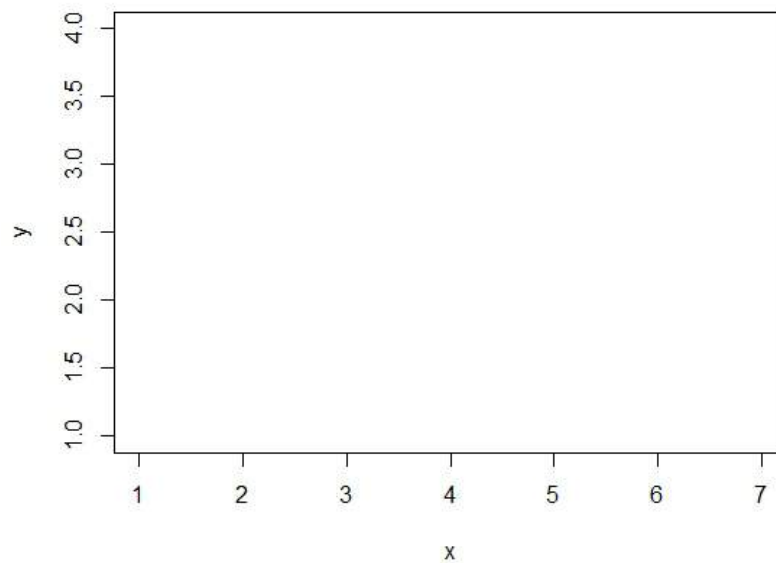


Figura 7.14: Séptima modificación del gráfico

En el caso de un gráfico de estilo “l”, se puede modificar el estilo de las líneas.

- **col**: Especifica el color.
- **lty**: Especifica el tipo de línea.
  - **1**: Continua. Valor por defecto.
  - **2**: Discontinua.
  - **3**: Línea de puntos.
  - **4**: Línea de puntos y rayas.
- **lwd**: Especifica el grosor de línea en relación al valor por defecto.

Vamos a modificar nuestro gráfico de estilo “l” con la mayor parte de modificaciones que podamos:

Listing 7.14: Código modificación gráfico estilo l

```
1 > x=c(1,3,6,7)
2 > y=c(2,3,4,1)
3 > plot(x,y, type = "l", col="blue", lty=2,lwd=3)
```

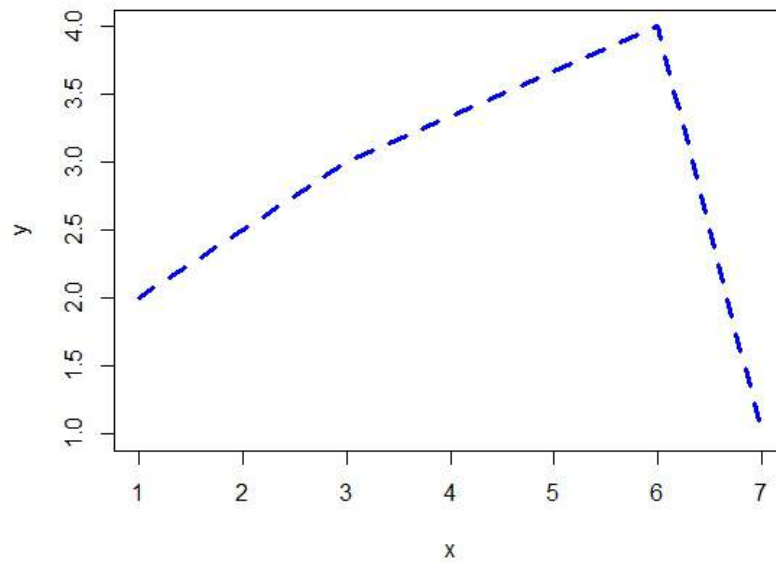


Figura 7.15: Gráfico modificado tipo 1

#### 7.1.4. Parámetros de ejes de coordenadas

- `xlim`, `ylim`: Especifican los rangos de los ejes mediante un vector con los extremos de los rangos.
- `xaxp`, `yaxp`: Especifican las marcas de los ejes como un vector  $c(n1,n2,m)$ , dibujando  $m+1$  marcas entre los valores  $n1$  y  $n2$ .

Vamos a hacer un ejemplo de una función modificando los parámetros de los ejes de coordenadas:

Listing 7.15: Código modificando ejes de coordenadas

```
1 > f=function(x){exp(x)}  
2 > plot(f)  
3 > plot(f,xlim=(0,10))  
4 > plot(f,xlim=c(0,10),xaxp=c(0,10,10),)
```



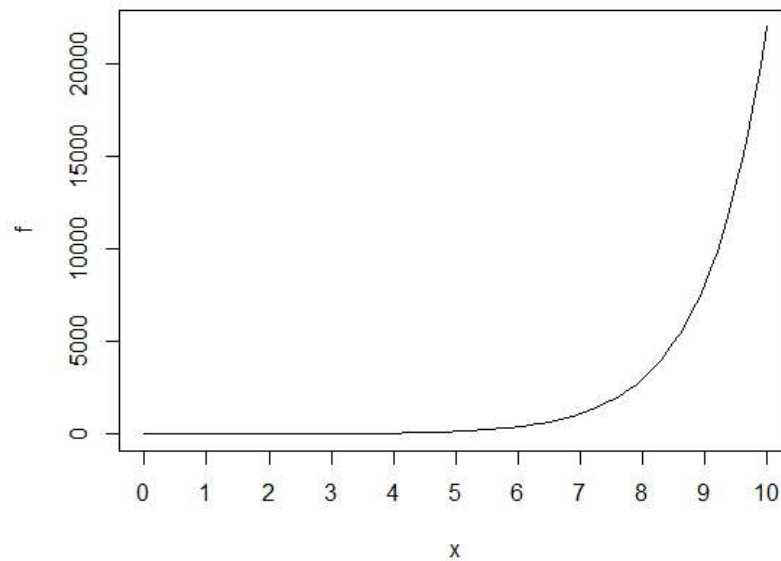


Figura 7.16: Gráfica modificada ejes

## 7.2. Cómo añadir elementos a un gráfico

En este capítulo aprenderemos añadir elementos a un gráfico a través de unas funciones adecuadas.

### 7.2.1. Añadir puntos

Podremos añadir puntos mediante la función `points(x,y)`, que permite añadir puntos de coordenadas  $x$  e  $y$  al gráfico. Se puede indicar su color, estilo, etc. mediante los parámetros específicos.

Vamos a realizar un ejemplo añadiendo puntos a una función:

Listing 7.16: Código añadiendo puntos

```
1 > f=function(x){x^2}
2 > plot(f,xlim=c(-1,1),ylim=c(-1,1))
3 > points(0,0,pch=16)
```

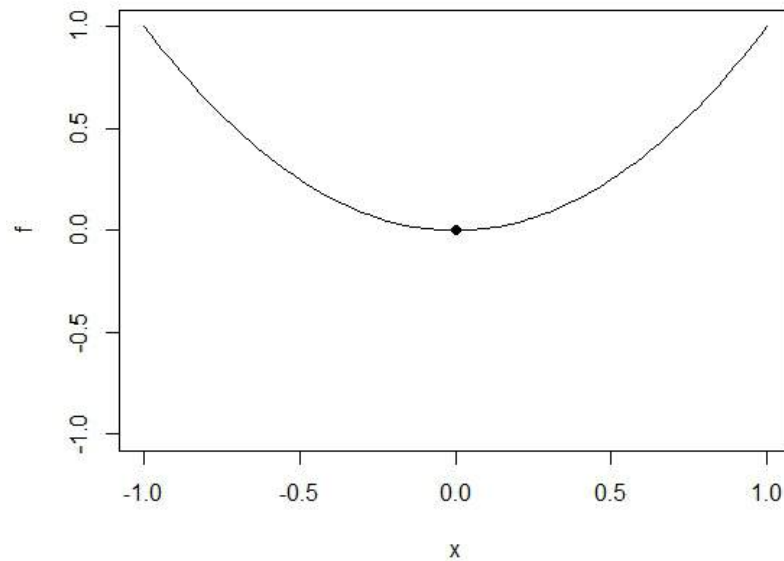


Figura 7.17: Gráfica con punto añadido

### 7.2.2. Añadir rectas

Podremos añadir rectas mediante la función `abline`, según el tipo de recta podremos añadir:

- `abline(a,b)` añade la recta  $y = a + bx$ .
- `abline(v=xo)` añade la recta vertical  $x = xo$ .
- `abline(h=yo)` añade la recta horizontal  $y = yo$ .

Se puede indicar el color, el grosor, etc. de la recta mediante los parámetros específicos. Vamos a dibujar una recta tangente al punto anterior:

Listing 7.17: Código añadir recta

```
1 > f=function(x){x^2}
2 > plot(f,xlim=c(-1,1),ylim=c(-1,1))
3 > points(0,0,pch=16)
4 > abline(h=0,col="green")
```

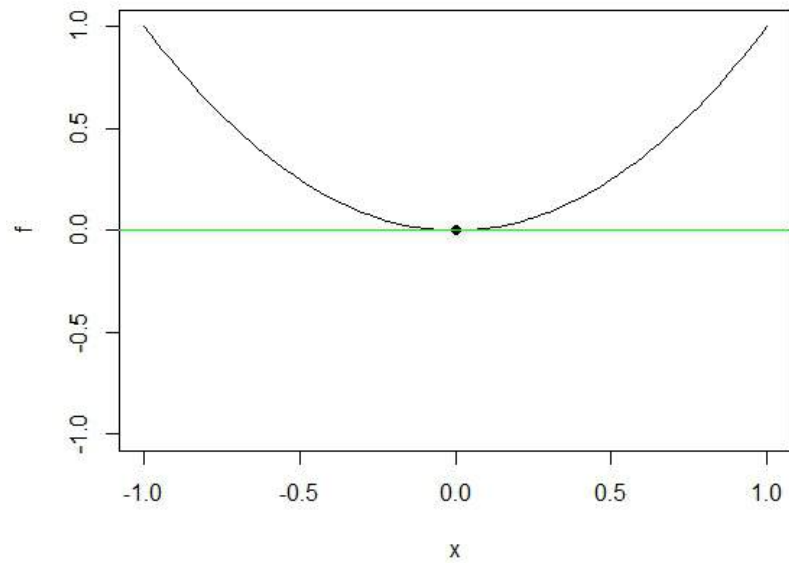


Figura 7.18: Gráfica con punto añadido y recta añadida

### 7.2.3. Añadir curvas

La función `curve` permite con el parámetro `add=TRUE` añadir una curva al gráfico. Si no añadimos el `add=TRUE`, `curve` tiene el mismo comportamiento que `plot` con la ventaja que sabe interpretar expresiones algebraicas con la variable  $x$  sin necesidad de utilizar `function` previamente.

Vamos a añadir al ejemplo de arriba una curva a la tangente previamente dibujada:

Listing 7.18: Código añadir curva

```
1 > f=function(x){x^2}
2 > plot(f, xlim=c(-1,1), ylim=c(-1,1))
3 > points(0,0, pch=16)
4 > abline(h=0, col="green")
5 > curve(-x^2, col="red", add=TRUE)
```

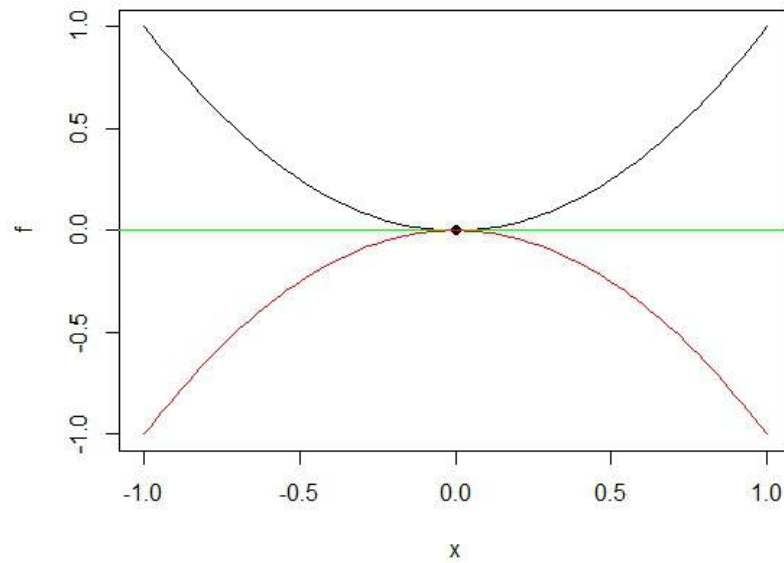


Figura 7.19: Gráfica con punto añadido, recta añadida y curva añadida

#### 7.2.4. Añadir texto

Para añadir texto utilizaremos la función `text(x,y,labels=...,pos=...)`, que permite añadir el texto especificado en `labels`, mediante comillas o `expression`, en  $(x,y)$ . La posición exacta depende del valor de `pos`:

- `pos=1`: Abajo del punto.
- `pos=2`: Izquierda del punto.
- `pos=3`: Arriba del punto.
- `pos=4`: Derecha del punto.
- `pos=NULL`: Encima del punto. Valor por defecto.

Vamos a indicar que el punto marcado es el punto de intersección entre las dos curvas y la recta:

Listing 7.19: Código añadir texto

```
1 > f=function(x){x^2}
2 > plot(f,xlim=c(-1,1),ylim=c(-1,1))
3 > points(0,0,pch=16)
4 > abline(h=0,col="green")
5 > curve(-x^2,col="red",add=TRUE)
6 > text(0,0,labels="pt.intersección",pos=1)
```

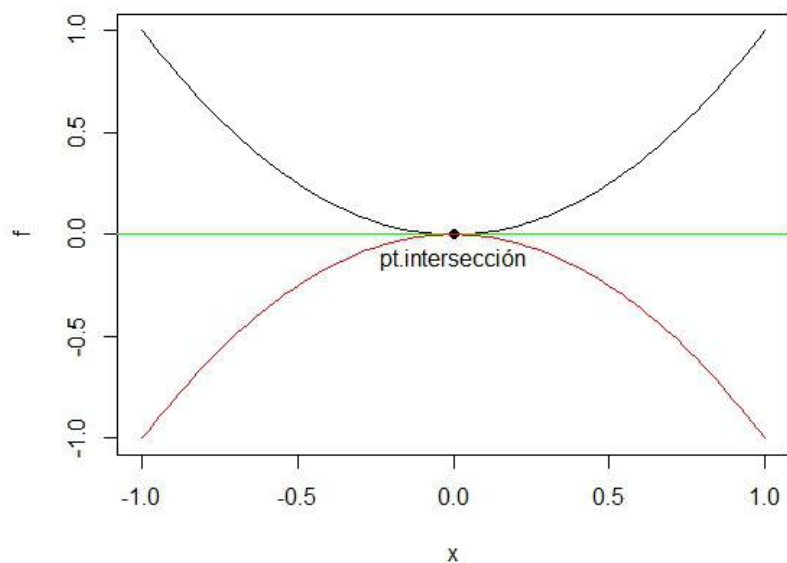


Figura 7.20: Gráfica con punto añadido, recta añadida, curva añadida y texto añadido

### 7.2.5. Añadir una leyenda

En gráficos con muchas curvas o rectas mediante la función `legend(posición, legend=..., parámetro=...,...,parámetro=...)` nos permite añadir una leyenda al gráfico activo. La posición indica donde queremos situar la leyenda, el vector `legend` contiene los nombres de las distintas curvas y cada `parámetro` contiene un vector de los valores del parámetro sobre las distintas curvas.

Vamos a añadir una leyenda al gráfico que hemos utilizado para todos los ejemplos:

Listing 7.20: Código añadir leyenda

```

1 > f=function(x){x^2}
2 > plot(f,xlim=c(-1,1),ylim=c(-1,1))
3 > points(0,0,pch=16)
4 > abline(h=0,col="green")
5 > curve(-x^2,col="red",add=TRUE)
6 > text(0,0,labels="pt.intersección",pos=1)
7 > legend("topleft",legend=c(expression(x^2),expression(-x^2)
  ,expression(y=0)),col=c("black","red","green"),lwd=3)

```

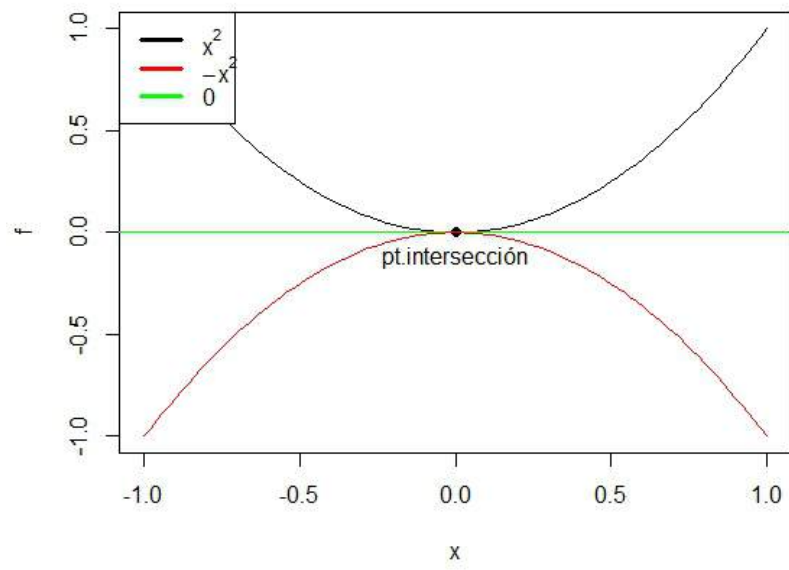


Figura 7.21: Gráfica con leyenda añadida

# Capítulo 8

## Tablas de frecuencias, diagramas de barras, otros gráficos básicos para datos cualitativos y un ejemplo completo

### 8.1. Tablas de frecuencias

#### 8.1.1. Tablas unidimensionales de frecuencias

Vamos a poner un ejemplo de una variable cualitativa, es decir, que sólo se puede comparar en número.

Mujer, Mujer, Hombre, Mujer, Mujer, Mujer, Mujer, Mujer, Mujer, Hombre, Mujer, Hombre, Hombre, Mujer, Mujer, Hombre, Mujer, Mujer, Mujer, Hombre.

A partir de estos datos vamos a sacar sus frecuencias absolutas, relativas y porcentajes.

**Frecuencias absolutas:**

Hombre: 6, Mujer: 14.

**Frecuencias relativas:**

Hombre:  $\frac{6}{20} = 0,3$ , Mujer:  $\frac{14}{20} = 0,7$ .

**Porcentajes:**

Hombre: 30 %, Mujer: 70 %.

La moda sería los objetos con la frecuencia relativa o absoluta más grande.

Las frecuencias absoluta y relativa se resumen en la tabla de frecuencias, de este ejemplo sería la siguiente:

Cuadro 8.1: Tabla de frecuencias del ejemplo

Sexo	n <sub>j</sub>	f <sub>j</sub>	%
Hombre	6	0.3	30 %
Mujer	14	0.7	70 %
Total	20	1	100 %

Ahora vamos a realizar un ejemplo de tabla unidimensionales de frecuencia en R:

Listing 8.1: Código tabla unidimensional de frecuencia

```

1 > x=c(3,2,5,1,3,1,5,6,2,2,2,1,3,5,2)
2 #Frecuencias absolutas
3 > table(x)
4 x
5 1 2 3 5 6
6 3 5 3 3 1
7 #Nombres tabla de frecuencia
8 > names(table(x))
9 [1] "1" "2" "3" "5" "6"
10 > table(x)[4]
11 5
12 3
13 > table(x)[3]
14 3
15 3
16 #Sumar valores tabla de frecuencia
17 > sum(table(x))
18 [1] 15
19 #Raíces cuadradas
20 > sqrt(table(x))
21 x
22      1      2      3      5      6
23 1.732051 2.236068 1.732051 1.732051 1.000000
24 #Tabla de frecuencias relativas
25 > prop.table(table(x))
26 x
27      1      2      3      5
28 0.20000000 0.33333333 0.20000000 0.20000000
29      6
30 0.06666667
31 #Moda
32 names(which(table(x)==3))
33 > names(which(table(x)==3))
34 [1] "1" "3" "5"
35 names(which(table(x)==max(table(x))))
36 > names(which(table(x)==max(table(x))))
37 [1] "2"

```

### 8.1.2. Tablas bidimensionales de frecuencias

Son tablas de frecuencias de dos variables. Vamos a ver un ejemplo de una de ellas:



Cuadro 8.2: Tabla de respuestas Si o no de hombre y mujer

Variable	Valores							
Respuesta	No	No	Si	No	Si	No	No	Si
Sexo	M	M	M	H	H	H	H	H

Cuadro 8.3: Tabla bidimensional de frecuencias

	Sexo	
Respuestas	H	M
No	3	2
Si	2	1

Vamos a proceder a hallar esta tabla en R:

Listing 8.2: Código tabla bidimensional

```

1 Respuestas=c("No", "No", "Si", "No", "Si", "No", "No", "Si")
2 Sexo=c("M", "M", "M", "H", "H", "H", "H", "H") #H es hombre, M es
  mujer
3
4 table(Respuestas, Sexo)
5      Sexo
6 Respuestas H M
7      No 3 2
8      Si 2 1
9 #Referise entrada
10 table(Respuestas, Sexo)[1,2]
11 [1] 2
12 table(Respuestas, Sexo)[ "No", "M" ]
13 [1] 2

```

### Tipos de frecuencias relativas

Como en el caso anterior, la función `proptable` halla las frecuencias relativas de los pares de variables. En este caso, tenemos dos tipos:

- **Frecuencias relativas globales:** Fracción de individuos que pertenecen a ambos niveles respecto del total de la muestra.
- **Frecuencias relativas marginales:** Fracción de individuos que pertenecen al segundo nivel respecto del total de la subpoblación definida por el primer nivel.

Vamos a seguir con el ejemplo anterior para hallar sus frecuencias relativas:

Listing 8.3: Código frecuencias relativas globales y marginales

```

1 #Frecuencias relativas globales
2 prop.table(table(Respuestas, Sexo))

```

```

3           Sexo
4 Respuestas      H      M
5           No 0.375 0.250
6           Si 0.250 0.125
7 #Frecuencias relativas marginales
8 #Respuestas
9 prop.table(table(Respuestas, Sexo), margin = 1)
10          Sexo
11 Respuestas      H      M
12          No 0.6000000 0.4000000
13          Si 0.6666667 0.3333333
14 #Sexo
15 prop.table(table(Respuestas, Sexo), margin = 2)
16          Sexo
17 Respuestas      H      M
18          No 0.6000000 0.6666667
19          Si 0.4000000 0.3333333

```

### 8.1.3. Tablas multidimensionales de frecuencias

Son tablas de frecuencias con cualquier número de variables.

A continuación vamos a ver un ejemplo de muchas variables cualitativas:

Cuadro 8.4: Tabla con tres variables

Variable	Valores							
Respuesta	No	No	Si	No	Si	No	No	Si
Sexo	M	M	M	H	H	H	H	H
País	Fra	Ale	Ita	Ita	Ita	Ita	Ale	Fra

Cuadro 8.5: Tabla tridimensional

		Respuesta	
País	Sexo	No	Si
Ale	H	1	0
	M	1	0
Ita	H	2	1
	M	0	1
Fra	H	0	1
	M	1	0

Vamos a proceder a hallar su tabla en R:

Listing 8.4: Código tabla tridimensional

```

1 Respuestas=c("No", "No", "Si", "No", "Si", "No", "No", "Si")

```

```

3 Sexo=c("M","M","M","H","H","H","H","H")#H es hombre, M es
  mujer
4 País=c("Fra","Ale","Ita","Ita","Ita","Ita","Ale","Fra")
5
6 table(Sexo,Respuestas,País)
7 , , País = Ale
8
9     Respuestas
10  Sexo No Si
11   H   1  0
12   M   1  0
13
14 , , País = Fra
15
16     Respuestas
17  Sexo No Si
18   H   0  1
19   M   1  0
20
21 , , País = Ita
22
23     Respuestas
24  Sexo No Si
25   H   2  1
26   M   0  1
27 ftable(Sexo,Respuestas,País)
28           País Ale Fra Ita
29 Sexo Respuestas
30 H     No           1  0  2
31     Si           0  1  1
32 M     No           1  1  0
33     Si           0  0  1
34 ftable(Sexo,Respuestas,País,col.vars=c("Sexo","Respuestas"))
35     Sexo      H      M
36     Respuestas No Si No Si
37 País
38 Ale           1  0  1  0
39 Fra           0  1  1  0
40 Ita           2  1  0  1
41 #Referirse entrada
42 table(Sexo,Respuestas,País)[ "H", "Si", "Ita" ]
43 [1] 1
44 table(Sexo,Respuestas,País)[ , "Si", "Ita" ]
45 H M
46 1 1
47 table(Sexo,Respuestas,País)[ , , "Ita" ]
48     Respuestas

```

```

49 Sexo No Si
50   H  2  1
51   M  0  1
52 #Frecuencias relativas globales
53 prop.table(table(Sexo, Respuestas, País))
54 , , País = Ale
55
56   Respuestas
57 Sexo    No    Si
58   H 0.125 0.000
59   M 0.125 0.000
60
61 , , País = Fra
62
63   Respuestas
64 Sexo    No    Si
65   H 0.000 0.125
66   M 0.125 0.000
67
68 , , País = Ita
69
70   Respuestas
71 Sexo    No    Si
72   H 0.250 0.125
73   M 0.000 0.125
74 prop.table(ftable(Sexo, Respuestas, País))
75           País  Ale  Fra  Ita
76 Sexo Respuestas
77 H    No           0.125 0.000 0.250
78     Si           0.000 0.125 0.125
79 M    No           0.125 0.125 0.000
80     Si           0.000 0.000 0.125
81 #Frecuencias marginales por país
82 prop.table(table(Sexo, Respuestas, País), margin=3)
83 , , País = Ale
84
85   Respuestas
86 Sexo    No    Si
87   H 0.50 0.00
88   M 0.50 0.00
89
90 , , País = Fra
91
92   Respuestas
93 Sexo    No    Si
94   H 0.00 0.50
95   M 0.50 0.00

```

```

96
97 , , País = Ita
98
99     Respuestas
100 Sexo   No   Si
101   H 0.50 0.25
102   M 0.00 0.25
103 #Sexo y país
104 prop.table(table(Sexo, Respuestas, País), margin=c(1,3))
105 , , País = Ale
106
107     Respuestas
108 Sexo           No           Si
109   H 1.0000000 0.0000000
110   M 1.0000000 0.0000000
111
112 , , País = Fra
113
114     Respuestas
115 Sexo           No           Si
116   H 0.0000000 1.0000000
117   M 1.0000000 0.0000000
118
119 , , País = Ita
120
121     Respuestas
122 Sexo           No           Si
123   H 0.6666667 0.3333333
124   M 0.0000000 1.0000000

```

Como podemos ver, R muestra la tabla como una lista de tablas bidimensionales, pero con `ftable` las muestra en un formato plano. Además de realizar la tabla hemos calculado las frecuencias relativas globales y marginales.

## 8.2. Tablas de frecuencias a partir de data frames

En este capítulo explicaremos cómo calcular frecuencias absolutas a partir de un data frame.

Para explicar cómo realizarlo vamos a utilizar los datos de la siguiente dirección: <http://aprender.uib.es/Rdir/bebenerg.txt>. Donde encontramos las siguientes variables:

- **sexo**: Sexo de los estudiantes.
- **estudio**: Grado en el que están matriculados.
- **bebe**: Si consumen bebidas energéticas.

Listing 8.5: Código de tabla de frecuencias a partir de data frames

```

1  Beb_Energ=read.table("http://aprender.uib.es/Rdir/bebenerg.txt
   ", header=TRUE, encoding="UTF-8")
2  head(Beb_Energ)
3      estudio bebe  sexo
4  1  Informática No  Mujer
5  2  Matemáticas No  Hombre
6  3  Ing.Industrial Sí  Mujer
7  4  Informática Sí  Hombre
8  5  Ing.Industrial No  Mujer
9  6  Matemáticas No  Mujer
10 #Frecuencias absolutas de cada variable
11 summary(Beb_Energ)
12      estudio  bebe      sexo
13  Informática :53   No:97   Hombre:83
14  Ing.Industrial:37  Sí:25   Mujer :39
15  Matemáticas :16
16  Telemática  :16
17 #Formato tabla de frecuencias
18 sapply(Beb_Energ, FUN=table)
19 $estudio
20
21  Informática Ing.Industrial  Matemáticas
22      53          37          16
23  Telemática
24      16
25
26 $bebe
27
28 No Sí
29 97 25
30
31 $sexo
32
33 Hombre  Mujer
34     83     39
35 #Tabla de frecuencias absolutas ordenadas
36 table(Beb_Energ)
37 , sexo = Hombre
38
39      bebe
40 estudio  No Sí
41  Informática  30 7
42  Ing.Industrial 19 6
43  Matemáticas   8 1
44  Telemática   10 2

```

```

46
47 , , sexo = Mujer
48
49         bebe
50 estudio      No Sí
51  Informática    11 5
52  Ing.Industrial 10 2
53  Matemáticas    6 1
54  Telemática     3 1
55 ftable(Beb_Energ)
56
57         sexo Hombre Mujer
58 estudio      bebe
59  Informática    No      30    11
60                Sí      7     5
61  Ing.Industrial No      19    10
62                Sí      6     2
63  Matemáticas    No      8     6
64                Sí      1     1
65  Telemática     No      10    3
66                Sí      2     1

```

### 8.3. Diagramas de barras

El gráfico para representar una variable cualitativa es el diagrama de barras. La altura de la barra es la frecuencia de la variable cualitativa.

Vamos a poner un ejemplo de variable cualitativa:

Mujer, Mujer, Hombre, Mujer, Mujer, Mujer, Mujer, Mujer, Hombre, Mujer, Hombre, Hombre, Mujer, Mujer, Hombre, Mujer, Mujer, Mujer, Hombre.

Vamos a ver cómo sería su código para generar el diagrama de barras de este ejemplo:

Listing 8.6: Código del diagrama de barras

```

1 Sexo_Ger=c("Mujer", "Mujer", "Hombre", "Mujer", "Mujer", "
2   Mujer", "Mujer", "Mujer", "Hombre", "Mujer", "Hombre", "
3   Hombre", "Mujer", "Mujer", "Hombre", "Mujer", "Mujer", "
4   Mujer", "Mujer", "Hombre")
5 #Diagrama de barras frecuencias absolutas
6 barplot(table(Sexo_Ger), col=c("lightblue", "pink"), main="
7   Diagrama de barras de las frecuencias absolutas de la
8   variable \"Sexo_Ger")
9 #Diagrama de barras frecuencias relativas
10 barplot(prop.table(table(Sexo_Ger)), col=c("lightblue", "pink"),
11   main="Diagrama de barras de las frecuencias relativas de la
12   variable \"Sexo_Ger")

```

Diagrama de barras de las frecuencias relativas de la variable "Sexo"

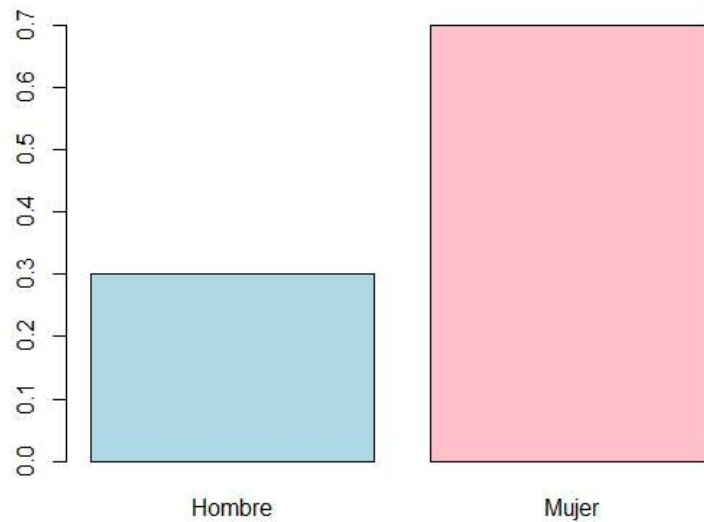


Figura 8.1: Diagrama de barras de las frecuencias relativas

Vamos a aplicar la función `barplot` a una tabla bidimensional.

Listing 8.7: Código función `barplot` tabla bidimensional

```

1 Respuestas=c("No", "No", "Si", "No", "Si", "No", "No", "Si")
2 Sexo=c("M", "M", "M", "H", "H", "H", "H", "H") #H es hombre, M es
  mujer
3 barplot(table(Sexo, Respuestas))
4 #Una al lado de la otra
5 barplot(table(Sexo, Respuestas), beside=TRUE)
6 #Con leyenda
7 barplot(table(Sexo, Respuestas), beside=TRUE, legend.text = TRUE)

```



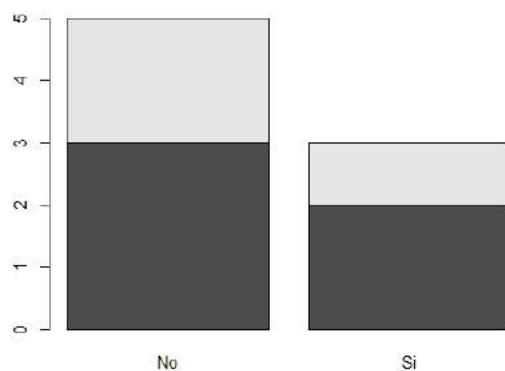


Figura 8.2: Diagrama de barras bidimensional

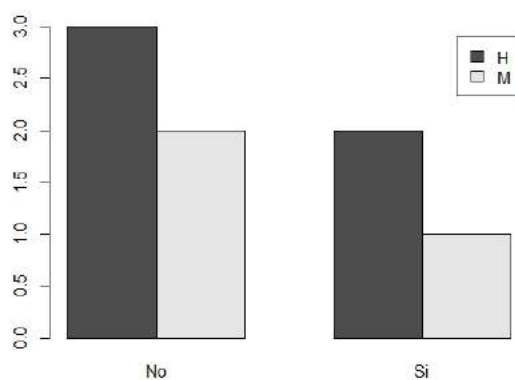


Figura 8.3: Diagrama de barras bidimensional datos enfrentados y leyenda

## 8.4. Otros gráficos básicos para datos cualitativos

### 8.4.1. Diagrama circular

Uno de los gráficos más populares es el diagrama circular o pie chart, donde se muestran las variables cualitativas como sectores circulares. El área del sector circular serán las frecuencias. Vamos a realizar un ejemplo de un diagrama circular:

Listing 8.8: Código diagrama circular

```

1 x=c(3,2,5,1,3,1,5,6,2,2,2,1,3,5,2)
2 pie(table(x), main="Diagrama circular")

```

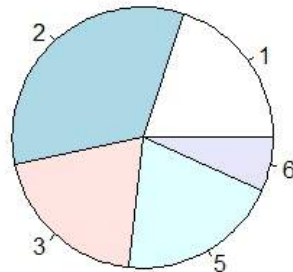
**Diagrama circular**

Figura 8.4: Diagrama circular

Esta función admite cambios de color y otros parámetros explicados anteriormente.

**8.4.2. Gráficos de mosaico**

Otro tipo de gráficos son los gráficos mosaico. Se obtienen sustituyendo cada entrada de frecuencias por una región rectangular de área proporcional a su valor. Vamos a realizar un diagrama mosaico de un ejemplo anterior:

Listing 8.9: Código gráfico mosaico

```
1 Respuestas=c("No", "No", "Si", "No", "Si", "No", "No", "Si")
2 Sexo=c("M", "M", "M", "H", "H", "H", "H", "H")
3 plot(table(Sexo, Respuestas), main="Gráfico mosaico")
```

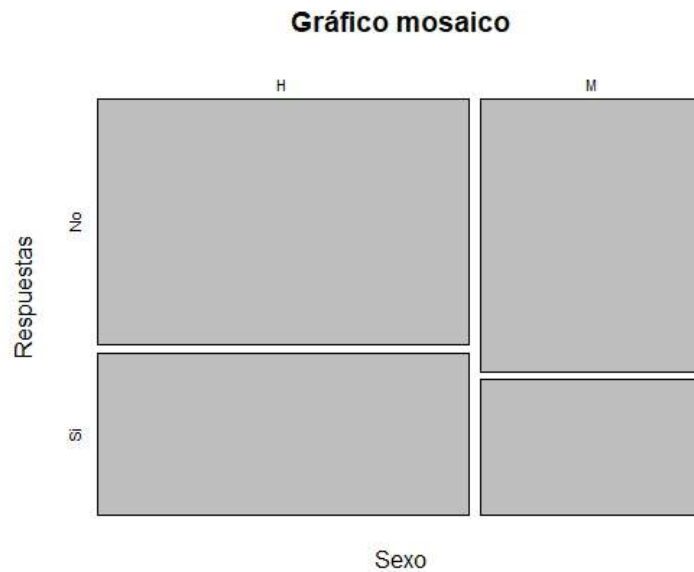


Figura 8.5: Gráfico mosaico

Admite parámetros como:

- **col**: Para asignar colores a los niveles de la última variable.
- **dir**: Igualado a un vector de direcciones v (vertical) y h (horizontal), sirve para especificar la dirección de las barras de cada variable.

### Paquete vcd

Primero instalaremos el paquete `vcd`, que incorpora una función `mosaic`. Y de ejemplo, utilizaremos la tabla de datos `HairEyeColor` que contiene el color del pelo, color de ojos y sexo de 592 estudiantes de estadística. Y las variables son:

- **Hair**: Black, Brown, Red, Blond.
- **Eye**: Brown, Blue, Hazel, Green.
- **Sex**: Male, Female.

Listing 8.10: Código gráfico mosaico `HairEyeColor`

```

1 > library(vcd)
2 mosaic(HairEyeColor, dir=c("v","h","v"), highlighting="Sex",
  highlighting_fill=c("pink","lightblue"), main="Gráfico de
  mosaico de la tabla HairEyeColor" )

```

## Gráfico de mosaico de la tabla HairEyeColor

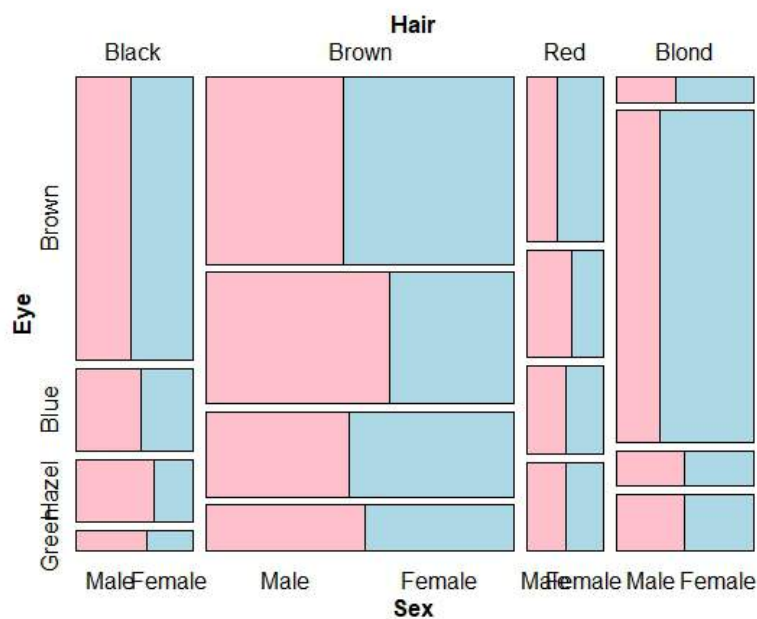


Figura 8.6: Gráfico mosaico de la tabla HairEyeColor

En esta función podremos encontrar los parámetros:

- **dir**: Es el equivalente al parámetro `dir` explicado anteriormente.
- **highlighting**: Sirve para destacar una variable.
- **highlighting\_fill**: Sirve para asignar colores a los niveles de la variable destacada.

## 8.5. Un ejemplo completo

En este capítulo vamos a repasar todos los contenidos de los capítulos anteriores. Para ello, utilizaremos los datos del paquete `vcd`. Los datos son los de `HairEyeColor`. Y las variables son:

- **Hair**: Black, Brown, Red, Blond.
- **Eye**: Brown, Blue, Hazel, Green.
- **Sex**: Male, Female.

Listing 8.11: Código ejemplo completo

```

1 #Tabla bidimensional de frecuencias absolutas
2 HEC=as.table(apply(HairEyeColor,MARGIN=c(1,2),FUN=sum))
3 > HEC
4      Eye
5 Hair  Brown Blue Hazel Green
6  Black   68   20   15    5

```

```

7   Brown    119    84    54    29
8   Red      26     17    14    14
9   Blond    7      94    10    16
10  #Nombres al castellano
11  dimnames(HEC)
12  $Hair
13  [1] "Black" "Brown" "Red"   "Blond"
14
15  $Eye
16  [1] "Brown" "Blue"  "Hazel" "Green"
17  dimnames(HEC)=list(Cabello=c("Negro", "Castaño", "Rojo", "Rubio")
18                      ,Ojos=c("Marrones", "Azules", "Pardos", "Verdes"))
19  > HEC
20
21      Ojos
22  Cabello  Marrones  Azules  Pardos  Verdes
23  Negro    68        20      15      5
24  Castaño  119        84      54      29
25  Rojo     26        17      14      14
26  Rubio    7          94      10      16
27  #Diagrama de mosaico
28  plot(HEC, col=c("lightblue"), main="Diagrama de mosaico de la
29      tabla bidimensional de frecuencias de colores de cabello y
30      ojos")
31  #Tablas de frecuencias absoluta y relativa de cada variable
32  sum(HEC)
33  [1] 592
34  #Frecuencia absoluta color de ojos
35  colSums(HEC)
36  Marrones  Azules  Pardos  Verdes
37  220       215     93      64
38  #Frecuencia absoluta color de cabello
39  rowSums(HEC)
40  Negro  Castaño  Rojo  Rubio
41  108    286    71   127
42  #Frecuencias relativas
43  #Color de los ojos
44  round(prop.table(colSums(HEC)),3)
45  Marrones  Azules  Pardos  Verdes
46  0.372     0.363  0.157  0.108
47  #Color del cabello
48  round(prop.table(rowSums(HEC)),3)
49  Negro  Castaño  Rojo  Rubio
50  0.182  0.483  0.120  0.215
51  #Diagrama de barras de la frecuencia relativa color ojos
52  barplot(prop.table(colSums(HEC)), ylim=c(0,0.4), col=c("
53      burlywood", "lightblue", "orange3", "lightgreen"), main="
54      Frecuencias relativas de colores de ojos")

```

```

49 #Color del cabello
50 barplot(prop.table(rowSums(HEC)),ylim=c(0,0.5),col=c("black","
    brown","red","gold"),main="Frecuencias relativas de colores
    de cabello")
51 #Frecuencias relativas globales
52 round(prop.table(HEC),3)
53     Ojos
54 Cabello    Marrones  Azules  Pardos  Verdes
55 Negro      0.115   0.034   0.025   0.008
56 Castaño    0.201   0.142   0.091   0.049
57 Rojo       0.044   0.029   0.024   0.024
58 Rubio      0.012   0.159   0.017   0.02
59 #Color del cabello
60 round(prop.table(HEC,margin=1),3)
61     Ojos
62 Cabello    Marrones  Azules  Pardos  Verdes
63 Negro      0.630   0.185   0.139   0.046
64 Castaño    0.416   0.294   0.189   0.101
65 Rojo       0.366   0.239   0.197   0.197
66 Rubio      0.055   0.740   0.079   0.126
67 #Color de los ojos
68 round(prop.table(HEC,margin=2),3)
69     Ojos
70 Cabello    Marrones  Azules  Pardos  Verdes
71 Negro      0.309   0.093   0.161   0.078
72 Castaño    0.541   0.391   0.581   0.453
73 Rojo       0.118   0.079   0.151   0.219
74 Rubio      0.032   0.437   0.108   0.250
75 #Diagrama de barras de frecuencias relativas
76 #Colores de cabello en cada ojos
77 barplot(prop.table(HEC,margin=1),beside=TRUE,legend.text=TRUE,
    col=c("black","brown","red","gold"),ylim=c(0,0.8),main="
    Frecuencias relativas de colores de cabello \n
    en cada color de ojos")
78 #Colores en ojos en cada color cabello
79 barplot(prop.table(HEC,margin=2),beside=TRUE,legend.text=TRUE,
    col=c("burlywood4","lightblue","orange3","lightgreen"),ylim=
    c(0,0.6),main="Frecuencias relativas de colores de ojos \n
    en cada color de cabello")
80 > library(vcd)
81 #Mosaico
82 mosaic(HairEyeColor,dir=c("v","h","v"),highlighting="Sex",
    highlighting_fill=c("pink","lightblue"),main="Gráfico
    mosaico de la tabla HairEyeColor")

```

Figura 8.7: Diagrama mosaico de la tabla bidimensional de frecuencias de colores de ojos y pelo

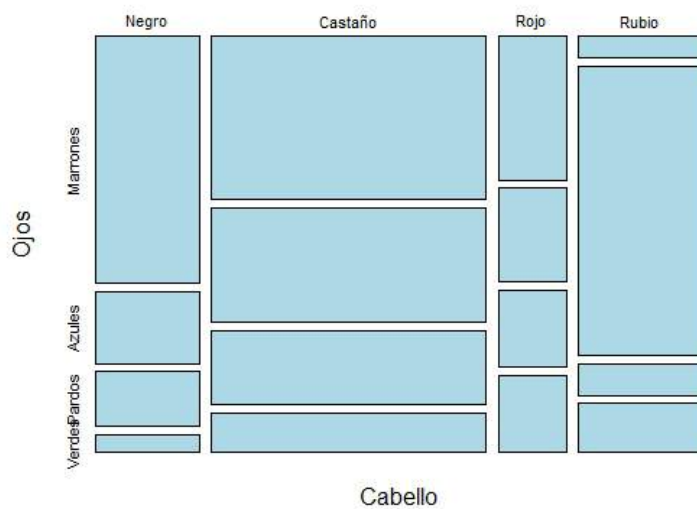


Figura 8.7: Diagrama mosaico ejemplo

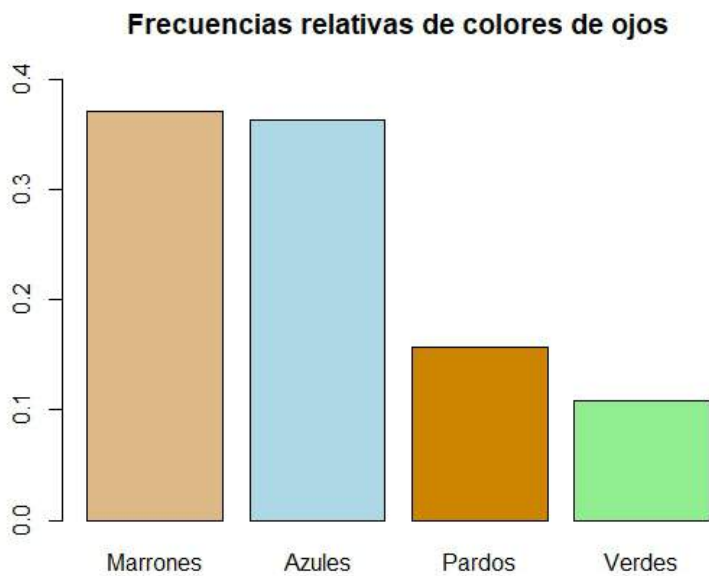


Figura 8.8: Diagrama de barras frecuencias relativas color ojos

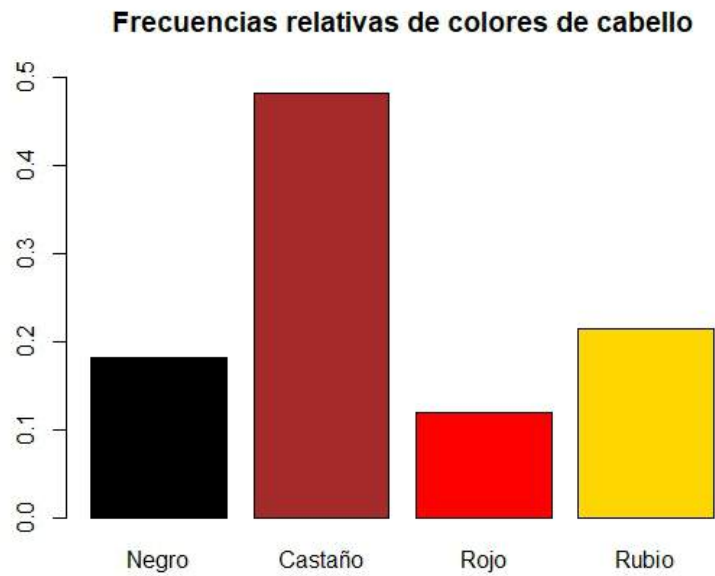


Figura 8.9: Diagrama de barras frecuencias relativas color cabello

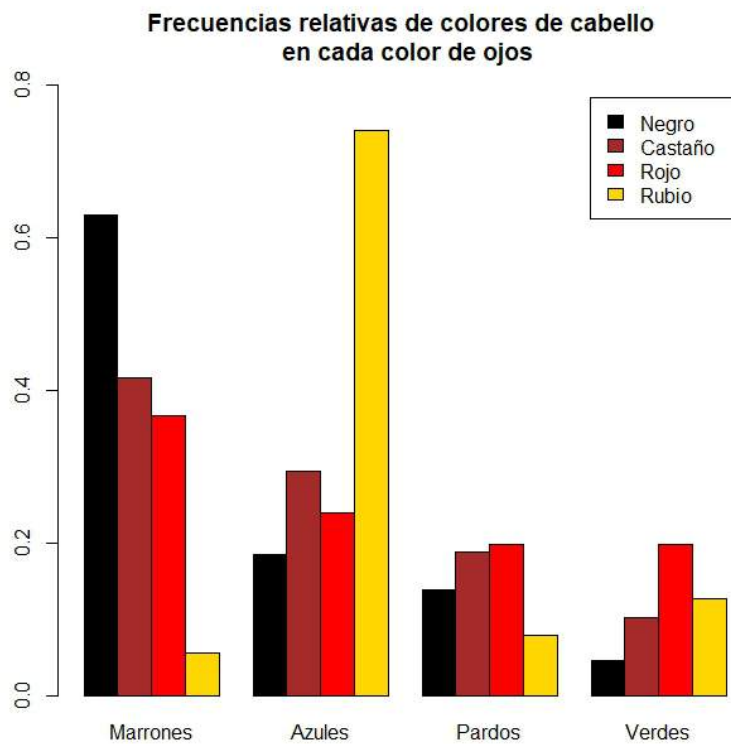


Figura 8.10: Diagrama de barras frecuencias relativas color cabello en cada ojos



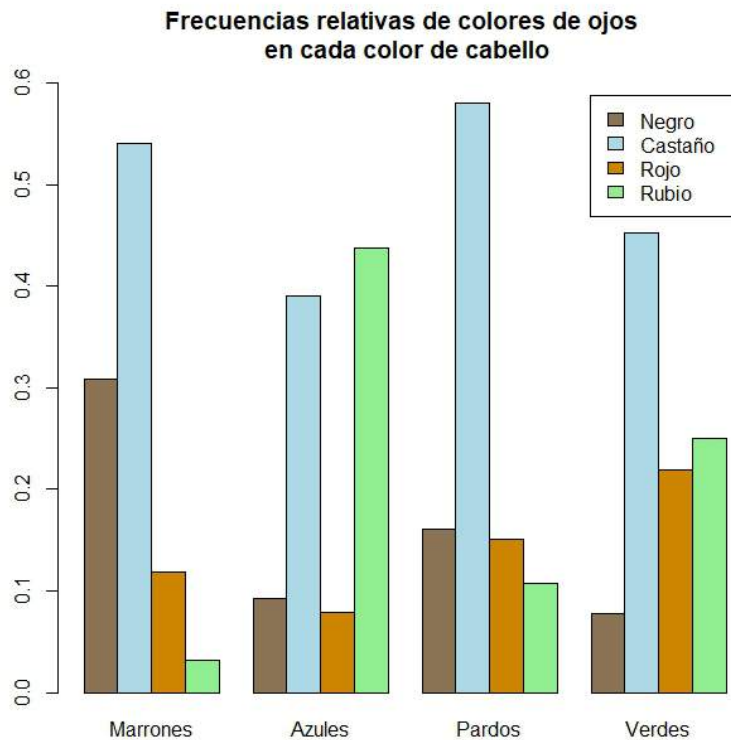


Figura 8.11: Diagrama de barras frecuencias relativas color ojos en cada color cabello

### Gráfico de mosaico de la tabla HairEyeColor

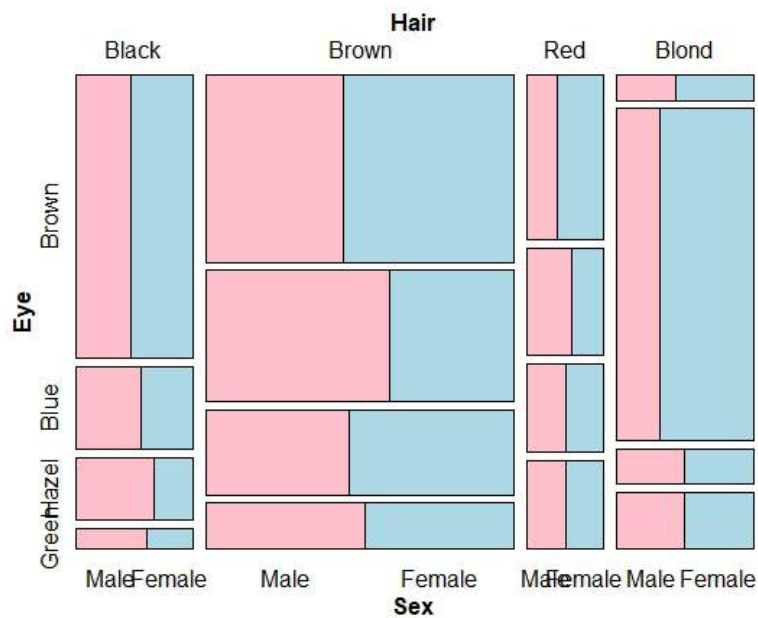


Figura 8.12: Gráfico mosaico de la tabla HairEyeColor



# Capítulo 9

## Datos ordinales

A diferencia de los valores cualitativos, los datos ordinales si que tienen un orden natural. En este capítulo vamos a ver cómo tratar los datos ordinales.

Se van a tratar igual que los cualitativos con la función `table()` obtendremos las frecuencias absolutas, con la función `barplot(table())` dibujaremos nuestro diagrama de barras correspondiente, para obtener las frecuencias relativas acumuladas aplicamos la función `cumsum(prop.table(table()))` y con la función `barplot(cumsum(prop.table(table())))` tendremos el diagrama de barras de las frecuencias acumuladas.

Por otro lado, si la table es multidimensional no podremos aplicar la función `cumsum`, por lo que tendremos que aplicar la función `apply(tabla,MARGIN=,FUN=cumsum)` y según queramos sumar filas o columnas utilizaremos:

- `MARGIN=1`: Por filas.
- `MARGIN=2`: Por columnas.



# Capítulo 10

## Frecuencias, medidas de tendencia central y de posición, medidas de dispersión y diagrama de cajas

### 10.1. Frecuencias

Los datos cuantitativos pueden ser ordenados usando el orden natural de los números reales.

**Datos:** 18,22,16,19,23,35,18,18,16.

**Orden valores observados:** 16<18<19<22<23<35.

Se pueden calcular sus frecuencias y frecuencias acumuladas. Sólo se tienen en cuenta los valores observados.

A continuación vamos a sacar las frecuencias y las frecuencias acumuladas de la cantidad de pases de un jugador de rugby:

Listing 10.1: Código medidas frecuencias pases

```
1 > pases=scan()
2 1: 31 34 35 28 29 29 29 32 35 31 29 31 28 26 30 30 33 35 30 29
   27 27 28 30 29
3 26:
4 Read 25 items
5 > table(pases)
6 pases
7 26 27 28 29 30 31 32 33 34 35
8  1  2  3  6  4  3  1  1  1  3
9 > cumsum(table(pases))
10 26 27 28 29 30 31 32 33 34 35
11  1  3  6 12 16 19 20 21 22 25
12 > prop.table(table(pases))
13 pases
14  26  27  28  29  30  31  32  33  34  35
15 0.04 0.08 0.12 0.24 0.16 0.12 0.04 0.04 0.04 0.12
16 > cumsum(prop.table(table(pases)))
```

17	26	27	28	29	30	31	32	33	34	35
18	0.04	0.12	0.24	0.48	0.64	0.76	0.80	0.84	0.88	1.00

## 10.2. Medidas de tendencia central y de posición

En este capítulo veremos cómo calcular con R la media, la mediana y los cuantiles de un vector numérico de datos.

### 10.2.1. Medidas de tendencia central

Dan un valor representativo de todas las observaciones.

- **Moda:** Es el valor o valores de máxima frecuencia.
- **Media aritmética:** Es la división de la suma de los datos entre el número total de datos.
- **Mediana:** Representa el valor central de la lista ordenada de observaciones. Si el número de datos es par, se calcula como la media aritmética de los dos valores centrales.

#### Moda

Para calcular la moda con datos cualitativos utilizamos la función `names(which(table(x))==max(table(x)))`. Pero en este capítulo estamos tratando datos cuantitativos por lo que transformaremos el resultado de esta función con la función `as.numeric`.

#### Media aritmética y mediana

Se calculan con las funciones `mean` y `median`.

Vamos a realizar un ejemplo con estas funciones:

Listing 10.2: Código medidas de tendencia central

```

1 > paradas=scan()
2 1: 12 9 10 13 13 4 11 10 8 10 9 12 8 10 8 6 14 7 11 7 9 12 5 6
   6
3 26:
4 Read 25 items
5 #Moda
6 > as.numeric(names(which(table(paradas)==max(table(paradas)))))
7 [1] 10
8 > table(paradas)
9 paradas
10  4  5  6  7  8  9 10 11 12 13 14
11  1  1  3  2  3  3  4  2  3  2  1
12 #Media

```

```

13 > mean(paradas)
14 [1] 9.2
15 #Mediana
16 > median(paradas)
17 [1] 9

```

### 10.2.2. Medidas de posición

Estiman qué valores dividen la población en unas determinadas proporciones. Los valores que determinan estas posiciones reciben el nombre de cuantiles.

Dada una proporción  $0 < p < 1$ , el cuantil de orden  $p$  de una variable cuantitativa, que denotaremos por  $Q_p$ , es el valor más pequeño tal que su frecuencia relativa acumulada es mayor o igual que  $p$ .

Algunos cuantiles importantes:

- **Mediana:** Es el cuantil  $Q_{0,5}$ .
- **Cuartiles:** Son los cuantiles  $Q_{0,25}$ ,  $Q_{0,5}$  y  $Q_{0,75}$ .
- **Deciles:** Son los cuantiles  $Q_p$  con  $p$  un múltiplo entero de 0.1.
- **Percentiles:** Son los cuantiles  $Q_p$  con  $p$  un múltiplo entero de 0.01.

### 10.2.3. Cálculo del cuantil

Con R los cuantiles de orden  $p$  se calculan con la función `quantile(x,p)` donde  $x$  es un vector de datos. R dispone de 9 métodos distintos para calcular los cuantiles a través del parámetro `type`. Usaremos `type=7` que corresponde al valor por defecto.

Vamos a calcular los cuantiles 0.25, 0.5 y 0.75 del ejemplo anterior:

Listing 10.3: Código cálculo cuantiles

```

1 > quantile(paradas,0.25)
2 25%
3 7
4 > quantile(paradas,0.5)
5 50%
6 9
7 > quantile(paradas,0.75)
8 75%
9 11
10 > cumsum(prop.table(table(paradas)))
11 4 5 6 7 8 9 10 11 12 13 14
12 0.04 0.08 0.20 0.28 0.40 0.52 0.68 0.76 0.88 0.96 1.00

```

## 10.3. Medidas de dispersión

En este capítulo veremos la variabilidad de los datos a través de las medidas de dispersión. Las medidas de dispersión evalúan lo desperdigados que están los datos.

- **Rango:** Es la diferencia entre el máximo y el mínimo de las observaciones. Se calcula con la función `diff(range(x))` donde `range(x)` da el valor mínimo y máximo de los datos.
- **Rango intercuartílico:** Es la diferencia entre los cuartiles  $Q_{0,75}$  y  $Q_{0,25}$ . Se calcula con la función `IQR`.
- **Varianza:** Es la media aritmética de las diferencias al cuadrado entre los datos  $x_i$  y la media  $\bar{x}$ .

$$s^2 = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n}.$$

Se calcula mediante la función `(n-1)/n*var`.

- **Varianza muestral:** ES la corrección de la varianza dada por:

$$\bar{s}^2 = \frac{n}{n-1} s^2.$$

Se calcula mediante la función `var`.

La varianza muestral de una muestra aproxima significativamente mejor la varianza real de la población que la varianza de la muestra.

- **Desviación típica:** Es la raíz de la varianza. Se calcula mediante `sqrt((n-1)/n)*sd`.
  - **Desviación típica muestral:** Es la raíz de la varianza muestral. Se calcula mediante la función `sd`.
- Las desviaciones típicas están en la misma unidad que las observaciones.

Vamos a realizar un ejemplo calculando este tipo de medidas, para ello, necesitaremos el data frame `ChickWeight`.

Listing 10.4: Código medidas de dispersión

```

1 data("ChickWeight")
2 #Rango
3 diff(range(ChickWeight$weight))
4 [1] 338
5 #Rango intercuartilico
6 IQR(ChickWeight$weight)
7 [1] 100.75
8 #Varianza muestral
9 var_mu=var(ChickWeight$weight)
10 > var_mu
11 [1] 5051.223
12 #Varianza verdadera
13 var_v=var(ChickWeight$weight)*(length(ChickWeight$weight)-1)/
    length(ChickWeight$weight)

```



```

14 > var_v
15 [1] 5042.484
16 #Desviación típica muestra
17 dtip_mu=sqrt(var_mu)
18 > dtip_mu
19 [1] 71.07196
20 sd(ChickWeight$weight)
21 [1] 71.07196
22 #Desviación típica verdadera
23 dtip=sqrt(var_v)
24 > dtip
25 [1] 71.01045
26 sd(ChickWeight$weight)*sqrt((length(ChickWeight$weight)-1)/
    length(ChickWeight$weight))
27 [1] 71.01045

```

### 10.3.1. Resumen estadístico

Si queremos obtener un resumen estadístico de un vector numérico que contenga los valores mínimo y máximo, sus tres cuartiles y su media, hay que usar la función `summary`. Esta función también puede aplicarse a un data frame, obteniendo un resumen estadístico de todas sus variables.

Si queremos aplicar cualquier función (y en particular, obtener el resumen estadístico) a algunas columnas de un data frame segmentándolas según los niveles de factor se usa `by(columnas,factor,FUN=función)`.

Vamos a aplicar esta función al ejemplo anterior:

Listing 10.5: Código resumen estadístico

```

1 #Resumen estadístico
2 summary(ChickWeight)
3      weight      Time      Chick      Diet
4 Min.   : 35.0   Min.   : 0.00   13      : 12   1:220
5 1st Qu.: 63.0   1st Qu.: 4.00    9       : 12   2:120
6 Median :103.0   Median :10.00   20      : 12   3:120
7 Mean   :121.8   Mean    :10.72   10      : 12   4:118
8 3rd Qu.:163.8   3rd Qu.:16.00   17      : 12
9 Max.   :373.0   Max.     :21.00   19      : 12
10                                     (Other):506
11 by(ChickWeight$weight, ChickWeight$Diet, FUN=summary)
12      ChickWeight$Diet : 1
13   Min. 1st Qu.  Median    Mean 3rd Qu.  Max.
14   35.00  57.75   88.00 102.65 136.50 305.00
15 ChickWeight$Diet : 2
16   Min. 1st Qu.  Median    Mean 3rd Qu.  Max.
17   39.0  65.5   104.5  122.6  163.0  331.0

```

```

17 -----
18 ChickWeight$Diet: 3
19   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
20   39.0   67.5   125.5   142.9   198.8   373.0
21 -----
22 ChickWeight$Diet: 4
23   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
24   39.00   71.25  129.50   135.26  184.75   322.00

```

## 10.4. Diagramas de cajas

Los diagramas de cajas son unas herramientas muy importantes para ver a simple vista datos estadísticos. Es un gráfico que resume algunos datos estadísticos de una variable cuantitativa.

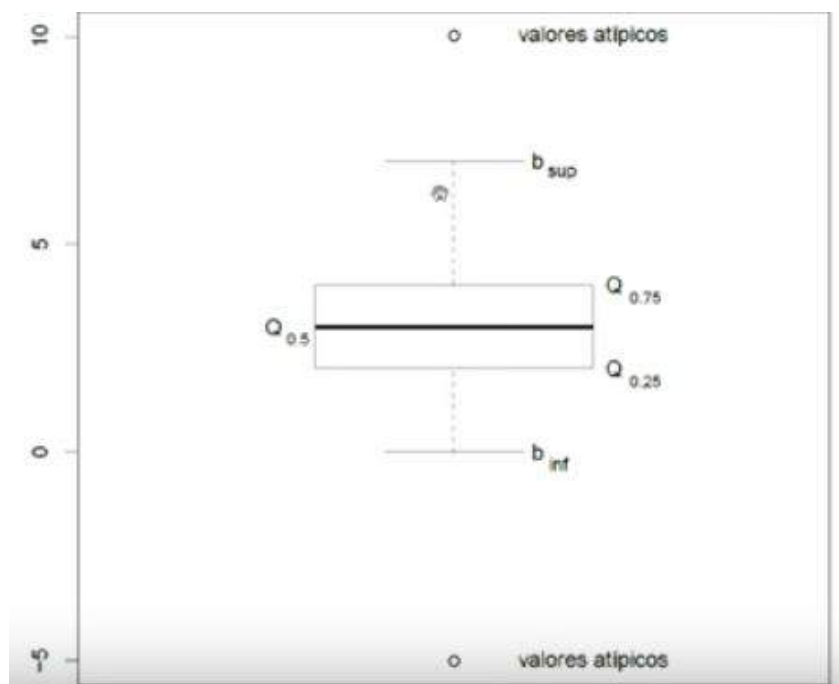


Figura 10.1: Ejemplo de diagrama de caja

En las dos puntas de las cajas tenemos los percentiles, en la esquina superior tenemos el máximo y en la esquina inferior el mínimo.

La función básica para dibujar un diagrama de caja con R es `boxplot`. Admite los parámetros usuales de `plot` para mejorar el resultado.

Hay distintos usos de esta función dependiendo de sus entradas:

- `boxplot(x)`: Calcula el diagrama de caja de un vector numérico.
- `boxplot(x,y,z)`: Si la aplicamos a varios vectores numéricos dibujará diagramas de caja en un mismo gráfico.

- `boxplot(df)`: Aplicado a un data frame representa todos los diagramas de caja del data frame en un solo paso.
- `boxplot(variable numérica variable factor, data=data frame)`: Dibuja en un único gráfico los diagramas de caja de una variable numérica de un data frame segmentada por un factor.

Vamos a utilizar los mismos datos que en el capítulo anterior para hacer distintos diagramas de cajas.

Listing 10.6: Código diagramas de cajas

```
1 #Diagramas de cajas
2 boxplot(ChickWeight$weight, main="Diagrama de caja de peso",
3         ylab="peso", col="red")
3 boxplot(ChickWeight$weight~ChickWeight$Diet, main="Diagrama de
4         caja de peso según la dieta", ylab="peso", xlab="dieta", col="
5         red")
```

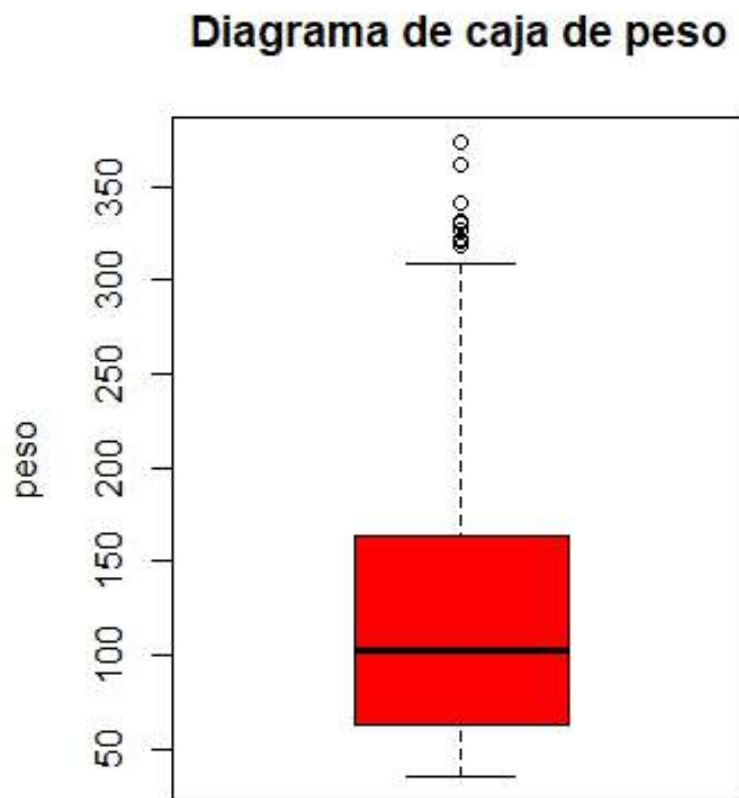


Figura 10.2: Diagrama de caja de peso

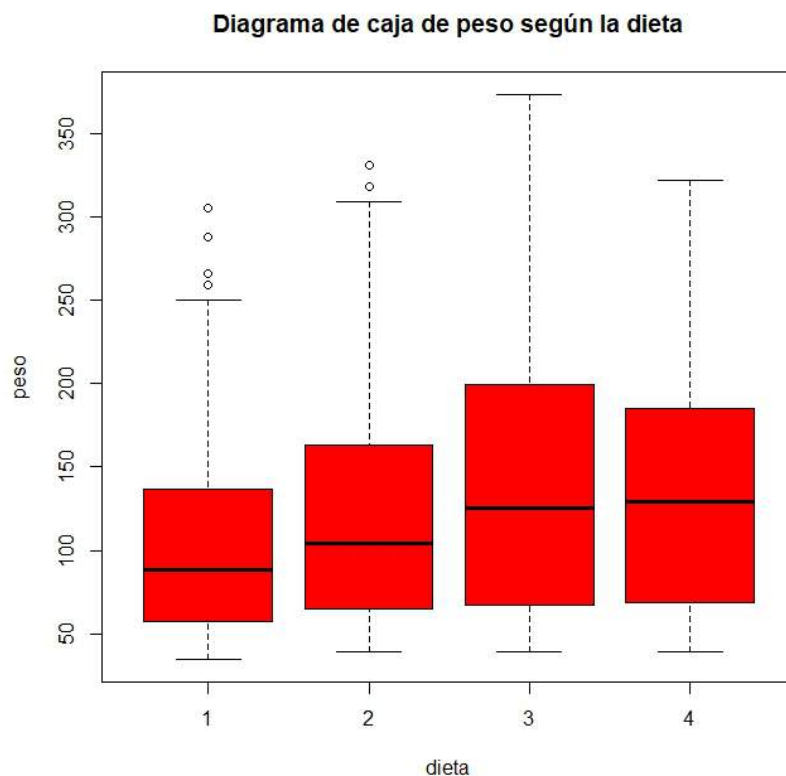


Figura 10.3: Diagrama de caja de peso según la dieta

Además, podremos obtener el resultado de una `boxplot` en una list. De la que podemos destacar:

- **stats**: Para cada diagrama de caja del gráfico, da las ordenadas de sus cinco líneas horizontales.
- **out**: Da los valores atípicos, indicando a qué diagrama pertenecen en la componente **group**.

También dispone de otros parámetros como:

- **horizontal**: Igualado a `TRUE`, dibuja las cajas horizontales.
- **names**: Especifica los nombres bajo los diagramas de cajas.
- **notch**: Dibuja las cinturas alrededor de las medianas que permiten contrastar si las medianas poblacionales son diferentes.
- **plot**: Igualado a `FALSE` calcula el diagrama de caja, pero no lo dibuja.

Vamos a comprobar con estos parámetros el ejemplo anterior:

Listing 10.7: Código diagrama de caja con parámetros modificados

```
1 boxplot(ChickWeight$weight~ChickWeight$Diet, main="Diagrama de
  caja de peso según la dieta", ylab="peso", xlab="dieta", col="
  red", notch=TRUE)
```

```

2  boxplot(ChickWeight$weight~ChickWeight$Diet , plot=FALSE)$stats
3      [,1] [,2] [,3] [,4]
4  [1,]  35.0  39.0  39.0  39.0
5  [2,]  57.5  65.0  67.0  69.0
6  [3,]  88.0 104.5 125.5 129.5
7  [4,] 137.0 163.0 199.5 185.0
8  [5,] 250.0 309.0 373.0 322.0
9  boxplot(ChickWeight$weight~ChickWeight$Diet , plot=FALSE)$out
10 [1] 288 305 259 266 318 331
11 boxplot(ChickWeight$weight~ChickWeight$Diet , plot=FALSE)$group
12 [1] 1 1 1 1 2 2

```

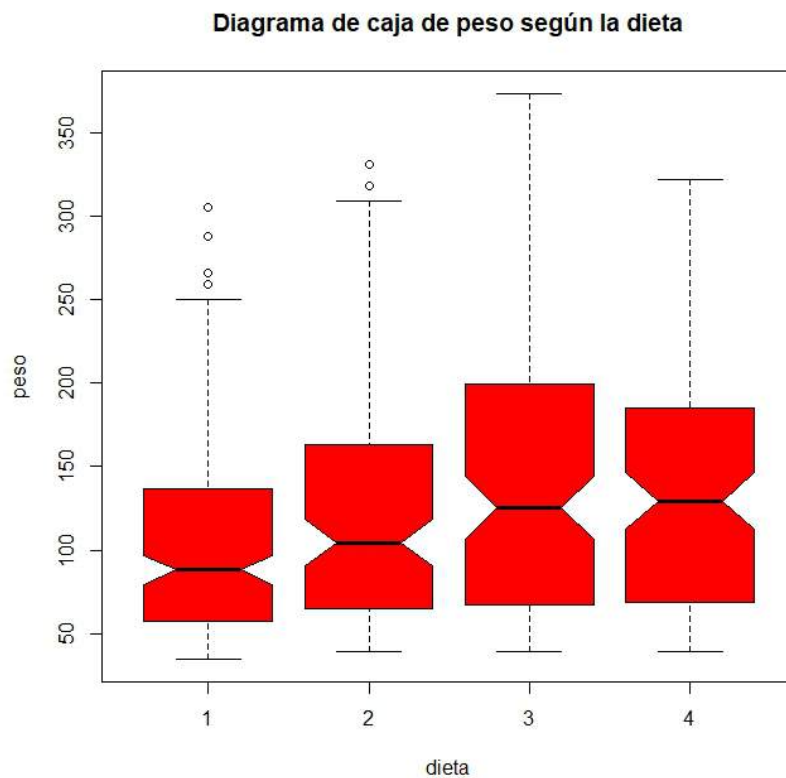


Figura 10.4: Diagrama de caja de peso según dieta con notch=TRUE



# Capítulo 11

## Matrices de datos cuantitativos, covarianzas y correlaciones, y representación gráfica de datos multidimensionales

### 11.1. Matrices de datos cuantitativos

En este capítulo vamos a ver datos que necesitan más de una variable para representarlos. Estos datos se agrupan en filas y en columnas.

Cuadro 11.1: Representación de datos cuantitativos

	Variable 1	Variable 2	...	Variable p
Individuo 1	x11	x12	...	x1p
Individuo 2	x21	x22	...	x2p
...	...	...	...	...
Individuo n	xn1	xn2	...	xnp

Vamos a representar la matriz tanto por filas como por columnas.

#### 11.1.1. Estadísticos multidimensionales

Vector de medias:

$$\vec{X} = (x_{o1}, x_{o2}, \dots, x_{op}), x_{oj} = \frac{1}{n} \sum_{i=1}^n x_{ij}$$

Vector de varianzas:

$$s^2x = (s_1^2, s_2^2, \dots, s_p^2), s_j^2 = \frac{1}{n} \sum_{i=1}^n (x_{ij} - x_{oj})^2$$

Vector de varianzas muestrales:

$$\vec{s}_x^2 = (\vec{s}_1^2, \vec{s}_2^2, \dots, \vec{s}_p^2)$$

$$\bar{s}_j^2 = \frac{1}{n-1} \sum_{i=1}^n (x_{ij} - \bar{x}_{oj})^2 = \frac{n}{n-1} s_j^2$$

Vamos a ver cómo hallar en R los estadísticos anteriores:

Listing 11.1: Código estadísticos multidimensionales

```

1 X=data.frame(V1=c(1,1,2,3),V2=c(-1,0,3,0),V3=c(3,3,0,1))
2 > X
3   V1 V2 V3
4  1  1 -1  3
5  2  1  0  3
6  3  2  3  0
7  4  3  0  1
8 #Media
9 sapply(X,mean)
10  V1  V2  V3
11 1.75 0.50 1.75
12 #Varianza muestral
13 sapply(X,var)
14  V1  V2  V3
15 0.9166667 3.0000000 2.2500000
16 #Desviación típica muestral
17 sapply(X,sd)
18  V1  V2  V3
19 0.9574271 1.7320508 1.5000000
20 #Verdaderas
21 #Varianza
22 var_ver=function(x){var(x)*(length(x)-1)/length(x)}
23 sapply(X,var_ver)
24  V1  V2  V3
25 0.6875 2.2500 1.6875
26 #Desviación típica
27 sd_ver=function(x){sqrt(var_ver(x))}
28 sapply(X,sd_ver)
29  V1  V2  V3
30 0.8291562 1.5000000 1.2990381

```

Es muy típico tipificar una tabla multidimensional, para ello la restaremos su media y la dividiremos por su desviación típica. Para tipificar una tabla de datos multidimensional en R vamos a utilizar la función `scale(X,center=...,scale=...)`, donde:

- **X**: Matriz o tabla de datos.
- **center**: Vector que restamos a la columnas. `center=TRUE` (defecto), restamos el valor de medias. `center=FALSE`, no restamos nada.
- **scale**: Vector por el que dividimos la matriz. `scale=TRUE` (defecto), dividimos por las desviaciones típicas muestrales. `scale=FALSE`, no dividimos por nada.

Vamos a proceder a tipificar nuestra tabla de datos:



Listing 11.2: Código tipificación tabla de datos

```

1 #Tipificacion tabla de datos
2 scale(X)
3           V1           V2           V3
4 [1, ] -0.7833495 -0.8660254  0.8333333
5 [2, ] -0.7833495 -0.2886751  0.8333333
6 [3, ]  0.2611165  1.4433757 -1.1666667
7 [4, ]  1.3055824 -0.2886751 -0.5000000
8 attr(,"scaled:center")
9   V1   V2   V3
10 1.75 0.50 1.75
11 attr(,"scaled:scale")
12   V1   V2   V3
13 0.9574271 1.7320508 1.5000000
14 apply(scale(X),2,mean)
15           V1           V2           V3
16 -4.163336e-17  0.000000e+00  0.000000e+00
17 apply(scale(X),2,var)
18 V1 V2 V3
19  1  1  1
20 scale(X,center=TRUE,scale=FALSE)
21           V1   V2   V3
22 [1, ] -0.75 -1.5  1.25
23 [2, ] -0.75 -0.5  1.25
24 [3, ]  0.25  2.5 -1.75
25 [4, ]  1.25 -0.5 -0.75
26 attr(,"scaled:center")
27   V1   V2   V3
28 1.75 0.50 1.75

```

## 11.2. Covarianzas y correlaciones

En este capítulo vamos a ver las covarianzas y las correlaciones de una matriz de datos.

### 11.2.1. Covarianza entre dos variables

**Covarianza:**

$$s_{ij} = \frac{1}{n} \sum_{k=1}^n ((x_{ki} - \bar{x}_{oi})(x_{kj} - \bar{x}_{oj})) = \frac{1}{n} \sum_{k=1}^n x_{ki}x_{kj} - \bar{x}_{oi}\bar{x}_{oj}$$

**Covarianza muestral**

$$\vec{s}_{ij} = \frac{1}{n-1} \sum_{k=1}^n ((x_{ki} - \bar{x}_{oi})(x_{kj} - \bar{x}_{oj})) = \frac{n}{n-1} s_{ij}$$

Relación entre ambas:

$$s_{ij} = s_{ji}, \vec{s}_{ij} = \vec{s}_{ji}, s_{ii} = s_i^2, \vec{s}_{ii} = \vec{s}_i^2$$

Ejemplo de cálculo de covarianza:

$$X = \begin{pmatrix} 1 & -1 & 3 \\ 1 & 0 & 3 \\ 2 & 3 & 0 \\ 3 & 0 & 1 \end{pmatrix}$$

$$s_{12} = \frac{1}{4}(1 \cdot (-1) + 1 \cdot 0 + 2 \cdot 3 + 3 \cdot 0) - 1,75 \cdot 0,5 = 1,25 - 0,875 = 0,375$$

$$\vec{s}_{12} = \frac{4}{3}s_{12} = 0,5$$

Vamos a proceder a calcular la covarianza en R:

Listing 11.3: Código de cálculo de covarianza

```

1 X=data.frame(V1=c(1,1,2,3),V2=c(-1,0,3,0),V3=c(3,3,0,1))
2 > X
3   V1 V2 V3
4   1  -1  3
5   1   0  3
6   2   3  0
7   3   0  1
8 #Covarianza entre las dos primeras columnas
9 #Muestral
10 cov(X$V1,X$V2)
11 [1] 0.5
12 #Verdadera
13 cov(X$V1,X$V2)*3/4
14 [1] 0.375

```

### 11.2.2. Matrices covarianzas y covarianzas muestrales

$$s = \begin{pmatrix} s_{11} & s_{12} & \cdots & s_{1p} \\ s_{21} & s_{22} & \cdots & s_{2p} \\ \cdots & \cdots & \cdots & \cdots \\ s_{p1} & s_{p2} & \cdots & s_{pp} \end{pmatrix}, \vec{s} = \begin{pmatrix} \vec{s}_{11} & \vec{s}_{12} & \cdots & \vec{s}_{1p} \\ \vec{s}_{21} & \vec{s}_{22} & \cdots & \vec{s}_{2p} \\ \cdots & \cdots & \cdots & \cdots \\ \vec{s}_{p1} & \vec{s}_{p2} & \cdots & \vec{s}_{pp} \end{pmatrix}$$

Vamos a ver cómo podemos realizar en R las matrices covarianzas y covarianzas muestrales:

Listing 11.4: Código matrices covarianzas y covarianzas muestrales

```

1 #Matrices covarianzas y covarianzas muestrales
2 #Muestral
3 cov(X)

```

```

4           V1          V2          V3
5 V1  0.9166667  0.500000  -1.083333
6 V2  0.5000000  3.000000  -2.166667
7 V3 -1.0833333  -2.166667  2.250000
8 #Verdadera
9 n=dim(X) [1]
10 ((n-1)/n)*cov(X)
11           V1          V2          V3
12 V1  0.6875  0.375  -0.8125
13 V2  0.3750  2.250  -1.6250
14 V3 -0.8125  -1.625  1.6875

```

### 11.2.3. Correlación entre dos variables

$$r_{ij} = \text{COR}(x_{oi}, x_{oj}) = \frac{s_{ij}}{s_i \cdot s_j}$$

$$\vec{r}_{ij} = \frac{\vec{s}_{ij}}{\vec{s}_i \cdot \vec{s}_j} = r_{ij}$$

### 11.2.4. Propiedades de la correlación entre dos variables

- **Simetría:**  $r_{ij} = r_{ji}$ .
- **Correlación de una variable consigo misma:**  $r_{ii} = 1$ .
- **Valore entre -1 y 1:**  $-1 \leq r_{ij} \leq 1$ .
- **Relación perfecta:** Si  $r_{ij} = \pm 1$ , existe una relación lineal perfecta entre las variables  $x_{oi}$  y  $x_{oj}$ . O sea, existen valores a y b tal que  $x_{oi} = a \cdot x_{oj} + b$ .

La correlación es una covarianza normalizada.

**Ejemplo de correlación entre dos variables:**

$$X = \begin{pmatrix} 1 & -1 & 3 \\ 1 & 0 & 3 \\ 2 & 3 & 0 \\ 3 & 0 & 1 \end{pmatrix}$$

$$s_{12} = 0,375, s_1 = \frac{\sqrt{11}}{4} = 0,8292, s_2 = \frac{3}{2} = 1,5$$

$$r_{12} = \frac{0,375}{0,8292 \cdot 1,5} = 0,3015$$

### 11.2.5. Matrices de correlaciones

$$r = \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1p} \\ r_{21} & r_{22} & \cdots & r_{2p} \\ \cdots & \cdots & \cdots & \cdots \\ r_{p1} & r_{p2} & \cdots & r_{pp} \end{pmatrix}$$

Está formada por las correlaciones  $ij$ .

Vamos a calcular las correlaciones en R, para ello vamos a utilizar la función `cor`:

Listing 11.5: Código cálculo correlaciones

```

1 #Cálculo correlaciones
2 #Matriz correlaciones
3 cor(X)
4           V1           V2           V3
5 V1  1.0000000  0.3015113 -0.7543365
6 V2  0.3015113  1.0000000 -0.8339504
7 V3 -0.7543365 -0.8339504  1.0000000
8 cov(scale(X))
9           V1           V2           V3
10 V1  1.0000000  0.3015113 -0.7543365
11 V2  0.3015113  1.0000000 -0.8339504
12 V3 -0.7543365 -0.8339504  1.0000000

```

## 11.3. Representación gráfica de datos multidimensionales

En este capítulo vamos a ver cómo representar datos multidimensionales de una manera sencilla.

### 11.3.1. La función plot

Listing 11.6: Código dos variables en función plot

```

1 #Representación de dos en función plot
2 iris.pet=iris[,c("Petal.Length", "Petal.Width")]
3 plot(iris.pet, pch=20, xlab="Largo", ylab="Ancho")

```

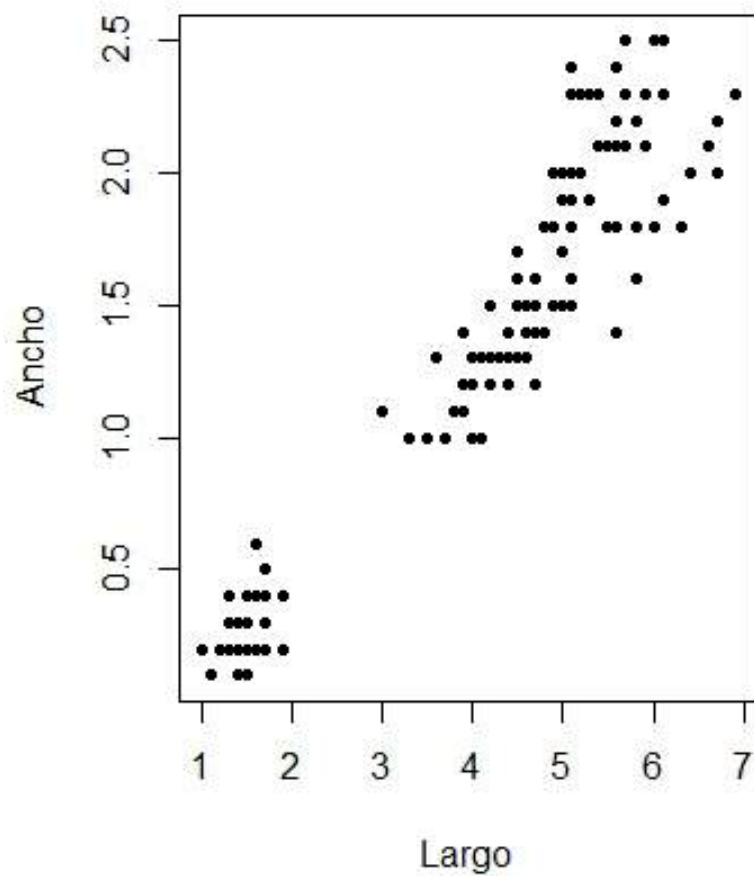


Figura 11.1: Dos variables en una función plot

### 11.3.2. La función `scatterplot3d`

Listing 11.7: Código tres variables en función `scatterplot3d`

```
1 #Representación de 3 variables
2 > library(scatterplot3d)#Instalar paquete antes
3 scatterplot3d(iris[,1:3],pch=20)
```

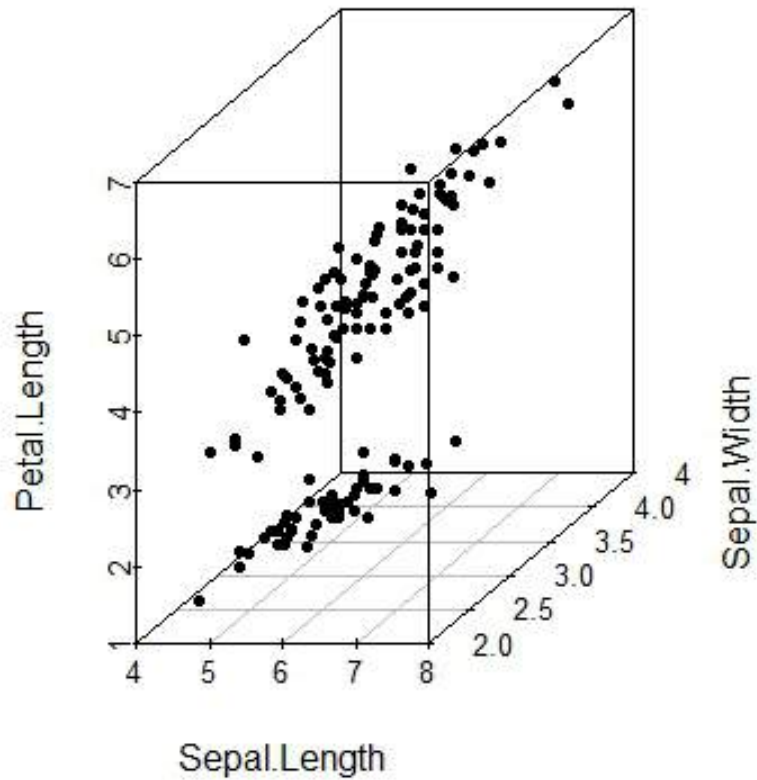


Figura 11.2: Tres variables en la función scatterplot3d

### 11.3.3. Diagramas de dispersión con plot

Listing 11.8: Código diagramas de dispersión con plot

```
1 #Diagramas de dispersión con plot
2 plot(iris[,1:4])
```

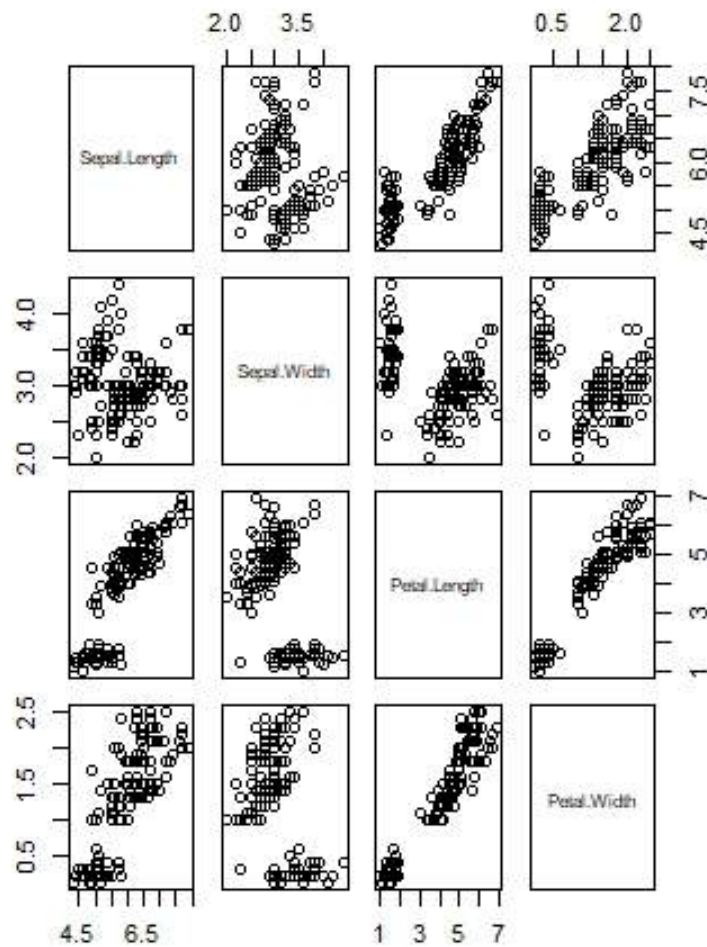


Figura 11.3: Diagramas de dispersión con plot

Se puede hacer este mismo diagrama de dispersión distinguiendo una variable cualitativa de la siguiente forma:

Listing 11.9: Código diagramas de dispersión con plot distinguido variable

```
1 plot(iris[,1:4], col=iris$Species)
```

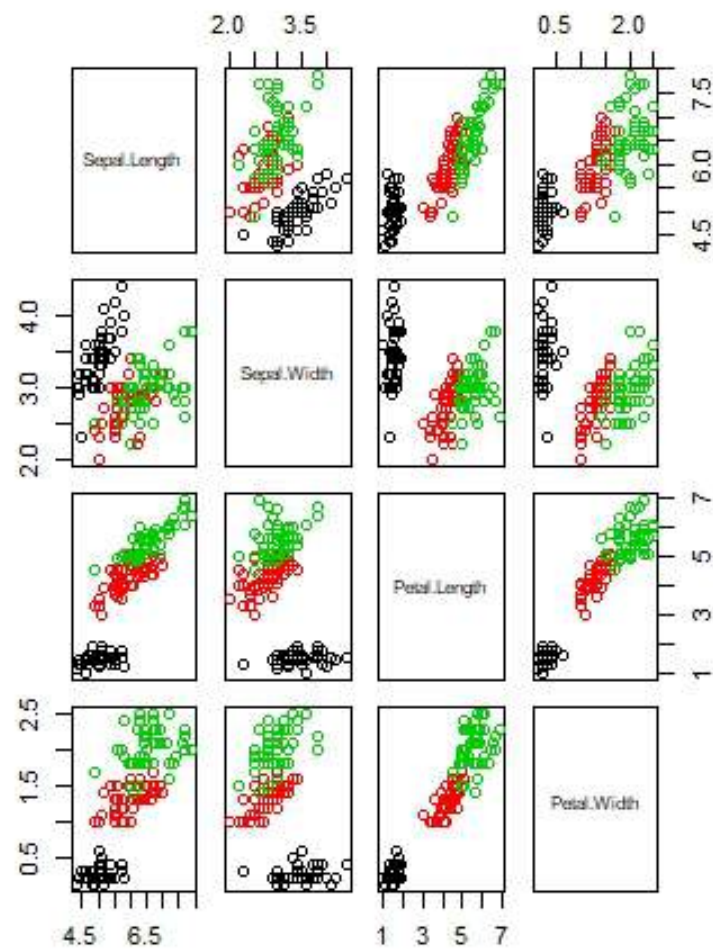


Figura 11.4: Diagramas de dispersión con plot distinguiendo variable

### 11.3.4. Diagramas de dispersión con la función `pairs`

Por otro lado, podemos utilizar para los diagramas de dispersión la función `pairs`:

Listing 11.10: Código diagrama dispersión con función `pairs`

```

1 #Pairs
2 pairs(iris[,1:4], col=iris$Species)

```



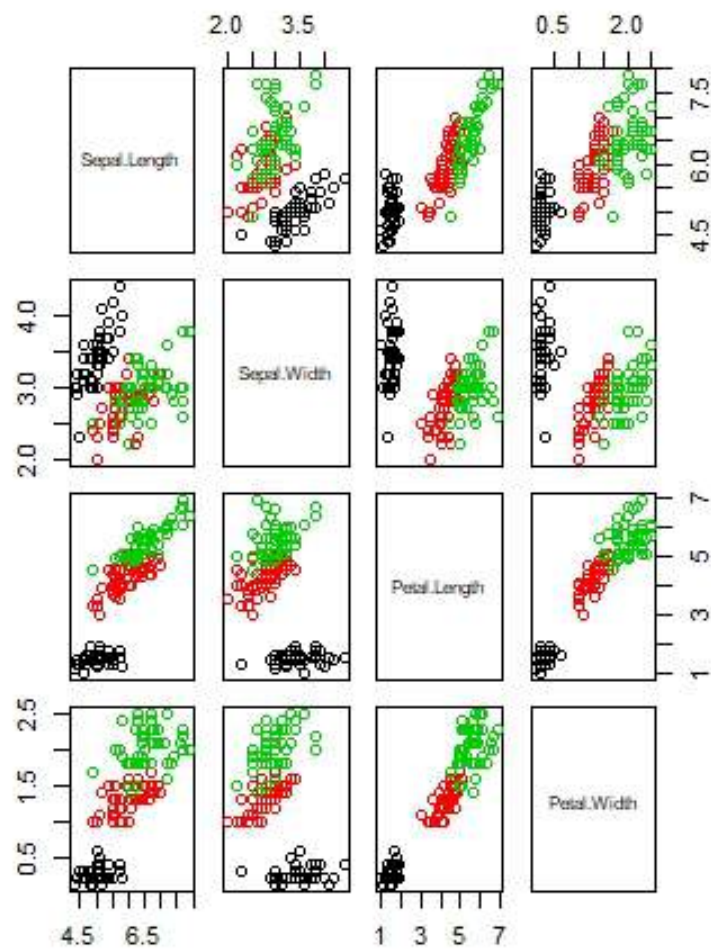


Figura 11.5: Diagramas de dispersión con pairs

Esta función se puede aplicar a una matriz.

### 11.3.5. Diagramas de dispersión con spm

Los diagramas de dispersión de esta función contienen mucha más información que los dos anteriores. Para ello, requerimos la instalación del paquete [car](#).

Listing 11.11: Código diagrama dispersión con función spm

```

1 > library(car)
2 #Diagramas de dispersión con spm
3 spm(iris[,1:4])

```

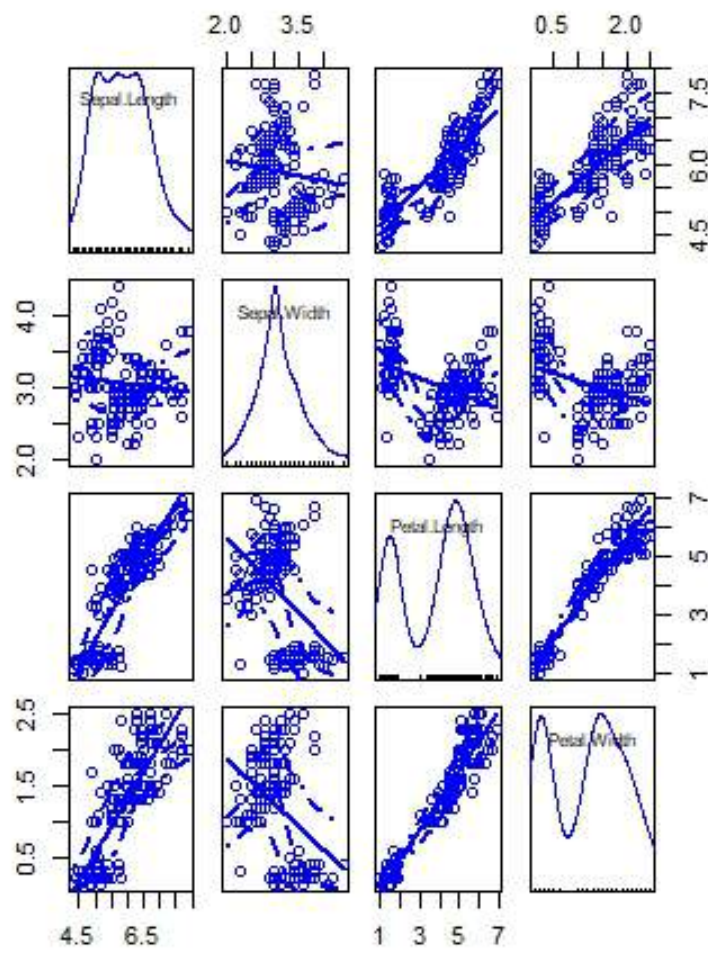


Figura 11.6: Diagramas de dispersión con spm

# Capítulo 12

## Agrupamiento de datos con R, estadísticos para datos agrupados e histogramas

### 12.1. Agrupamiento de datos con R

En este capítulo nos vamos a centrar cómo agrupar datos cuantitativos con R y a determinar sus frecuencias.

#### 12.1.1. Agrupamiento de datos

Es frecuente cuando trabajamos en estadística con datos cuantitativos, agruparlos por distintas razones.

- Datos continuos imposibles de medir exactamente.
- Datos discretos con muchos posibles valores.
- Gran número de datos e interesa estudiar sus frecuencias.

Para agrupar datos:

- a). Se decide el número de intervalos.
- b). Se decide su amplitud.
- c). Se calculan los extremos.
- d). Se calcula la marca de clase.

#### Número de intervalos

R dispone de las funciones `nclass.Sturges`, `nclass.scott` y `nclass.FD` para calcular este número usando las reglas de Sturges, Scott o Freedman-Diaconis, respectivamente.

## Amplitud

- Todos los intervalos de la misma longitud.
- La amplitud  $A$  es el cociente entre el rango de datos y el número de intervalos, redondeado por exceso.

## Extremos

Intervalos de la forma:

$$[L_1, L_2), [L_2, L_3), \dots, [L_k, L_{k+1})$$

$$L_1 = \min(x) - 0,5 \cdot \text{precisión}$$

$$L_i = L_1 + (i - 1)A$$

Vamos a proceder a realizar un agrupamiento de datos en R:

Listing 12.1: Código agrupamiento de datos

```

1 #Datos a agrupar
2 > iris$Petal.Length
3 [1] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 1.5 1.6 1.4 1.1
   1.2 1.5 1.3 1.4
4 [19] 1.7 1.5 1.7 1.5 1.0 1.7 1.9 1.6 1.6 1.5 1.4 1.6 1.6 1.5
   1.5 1.4 1.5 1.2
5 [37] 1.3 1.4 1.3 1.5 1.3 1.3 1.3 1.6 1.9 1.4 1.6 1.4 1.5 1.4
   4.7 4.5 4.9 4.0
6 [55] 4.6 4.5 4.7 3.3 4.6 3.9 3.5 4.2 4.0 4.7 3.6 4.4 4.5 4.1
   4.5 3.9 4.8 4.0
7 [73] 4.9 4.7 4.3 4.4 4.8 5.0 4.5 3.5 3.8 3.7 3.9 5.1 4.5 4.5
   4.7 4.4 4.1 4.0
8 [91] 4.4 4.6 4.0 3.3 4.2 4.2 4.2 4.3 3.0 4.1 6.0 5.1 5.9 5.6
   5.8 6.6 4.5 6.3
9 [109] 5.8 6.1 5.1 5.3 5.5 5.0 5.1 5.3 5.5 6.7 6.9 5.0 5.7 4.9
   6.7 4.9 5.7 6.0
10 [127] 4.8 4.9 5.6 5.8 6.1 6.4 5.6 5.1 5.6 6.1 5.6 5.5 4.8 5.4
   5.6 5.1 5.1 5.9
11 [145] 5.7 5.2 5.0 5.2 5.4 5.1
12 #Agrupación datos
13 nclass.scott(iris$Petal.Length)
14 [1] 6
15 nclass.Sturges(iris$Petal.Length)
16 [1] 9
17 nclass.FD(iris$Petal.Length)
18 [1] 5
19 k=nclass.FD(iris$Petal.Length)
20 A=diff(range(iris$Petal.Length))/k
21 > A
22 [1] 1.18
23 A=1.2#Redondear a la décima, en exceso

```

```

24 #Extremos de los intervalos
25 m=min(iris$Petal.Length)
26 L=m-0.05+A*(0:k)
27 > L
28 [1] 0.95 2.15 3.35 4.55 5.75 6.95
29 #Marcas de clase
30 marcas=(L[0:k]+L[1:k+1])/2
31 > marcas
32 [1] 1.55 2.75 3.95 5.15 6.35

```

### 12.1.2. Agrupamiento de datos con R

R agrupa los datos codificándolos en un factor cuyos niveles son las clases en las que hemos agrupado los valores y asignando cada dato a su clase.

```
cut(x,breaks=...,labels=...,right=...)
```

- **x** es un vector numérico.
- **breaks** es un vector numérico con los extremos de los intervalos en los que queremos agrupar.
- **labels** es un vector con las etiquetas de los intervalos.
- **right** indica cómo queremos el extremo derecho de los intervalos.

Retorna una lista con los elementos de **x** codificados. Una vez agrupados los datos, podemos calcular sus frecuencias.

Vamos a terminar de agrupar los datos del ejemplo anterior:

Listing 12.2: Código agrupamiento de datos con R

```

1 #Terminar de agrupar
2 etiq=c("muy corto","corto","medio","largo","muy largo")
3 Largo_cut=cut(iris$Petal.Length,breaks=L,labels=etiq,right=
4 FALSE)
5 > Largo_cut
6 [1] muy corto muy corto muy corto muy corto muy corto muy
7 corto muy corto
8 [8] muy corto muy corto muy corto muy corto muy corto muy
9 corto muy corto
10 [15] muy corto muy corto muy corto muy corto muy corto muy
11 corto muy corto
12 [22] muy corto muy corto muy corto muy corto muy corto muy
13 corto muy corto
14 [29] muy corto muy corto muy corto muy corto muy corto muy
15 corto muy corto
16 [36] muy corto muy corto muy corto muy corto muy corto muy
17 corto muy corto

```

```

11 [43] muy corto muy corto muy corto muy corto muy corto muy
    corto muy corto
12 [50] muy corto largo      medio      largo      medio      largo
    medio
13 [57] largo      corto      largo      medio      medio      medio
    medio
14 [64] largo      medio      medio      medio      medio      medio
    medio
15 [71] largo      medio      largo      largo      medio      medio
    largo
16 [78] largo      medio      medio      medio      medio      medio
    largo
17 [85] medio      medio      largo      medio      medio      medio
    medio
18 [92] largo      medio      corto      medio      medio      medio
    medio
19 [99] corto      medio      muy largo largo      muy largo largo
    muy largo
20 [106] muy largo medio      muy largo muy largo muy largo largo
    largo
21 [113] largo      largo      largo      largo      largo      muy
    largo muy largo
22 [120] largo      largo      largo      muy largo largo      largo
    muy largo
23 [127] largo      largo      largo      muy largo muy largo muy
    largo largo
24 [134] largo      largo      muy largo largo      largo      largo
    largo
25 [141] largo      largo      largo      muy largo largo      largo
    largo
26 [148] largo      largo      largo
27 Levels: muy corto corto medio largo muy largo
28 #Calculamos sus frecuencias para agruparlas en un data.frame
29 f_abs=as.vector(table(Largo_cut))
30 f_abs_acum=cumsum(table(Largo_cut))
31 f_rel=as.vector(prop.table(table(Largo_cut)))
32 f_rel_acum=cumsum(prop.table(table(Largo_cut)))
33 tabla_frec=data.frame(marcas, f_abs, f_abs_acum, f_rel, f_rel_acum
    )
34      marcas f_abs f_abs_acum      f_rel f_rel_acum
35 muy corto  1.55   50          50 0.3333333 0.3333333
36 corto     2.75    3          53 0.0200000 0.3533333
37 medio     3.95   34          87 0.2266667 0.5800000
38 largo     5.15   47         134 0.3133333 0.8933333
39 muy largo  6.35   16         150 0.1066667 1.0000000

```

## 12.2. Estadísticos para datos agrupados

En este capítulo vamos a ver cómo realizar cálculos estadísticos de datos agrupados, como por ejemplo de encuestas. Cuando tratamos con datos agrupados:

- Si disponemos de los datos en bruto, se calculan los estadísticos de la forma habitual.
- En caso contrario, se aproximan los estadísticos de los datos reales mediante estadísticos de los datos agrupados.

Mismas fórmulas que con datos sin agrupar pero **cambiando cada clase por su marca**.

### 12.2.1. Fórmulas de los estadísticos

Supongamos que tenemos  $k$  clases, con marcas  $X_1, \dots, X_k$  y frecuencias absolutas  $n_1, \dots, n_k$  con un total de  $n$  datos. Entonces:

**Media:**

$$\bar{x} = \frac{\sum_{i=1}^k n_i \cdot X_i}{n}$$

**Varianza:**

$$s^2 = \frac{\sum_{i=1}^k n_i \cdot X_i^2}{n} - \bar{x}^2$$

**Varianza muestral:**

$$\bar{s}^2 = \frac{n}{n-1} s^2$$

**Desviación típica:**

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

**Desviación típica muestral:**

$$\sqrt{\bar{s}} = \sqrt{\bar{s}^2}$$

**Moda** → **Intervalo modal:** Clase con mayor frecuencia.

Para realizar estos cálculos en R vamos a utilizar el data frame [Chile](#), que cuenta las intenciones de voto en el plebiscito de Chile de 1988.

- **region:** Región del país.
- **population:** Población del municipio o ciudad.
- **sex:** Sexo.
- **age:** Edad.
- **education:** Nivel educativo.
- **income:** Ingresos mensuales.
- **statusquo:** Nivel de apoyo al status quo.
- **vote:** Intención de voto.

Listing 12.3: Código fórmulas estadísticas aplicadas al data frame Chile

```

1 > library(car)
2 > data(Chile)
3 > head(Chile)
4   region population sex age education income statusquo vote
5 1      N    175000  M  65          P   35000    1.00820    Y
6 2      N    175000  M  29          PS    7500   -1.29617    N
7 3      N    175000  F  38          P   15000    1.23072    Y
8 4      N    175000  F  49          P   35000   -1.03163    N
9 5      N    175000  F  23          S   35000   -1.10496    N
10 6      N    175000  F  28          P    7500   -1.04685    N
11 #Quitar datos perdidos
12 Chile2=na.omit(Chile)
13 #Agrupamiento de datos
14 k=nclass.scott(Chile2$age)
15 > k
16 [1] 14
17 k=nclass.scott(Chile2$age)
18 diff(range(Chile2$age))/k
19 [1] 3.714286
20 A=4
21 #Creación tabla de frecuencias
22 #Creación tabla de frecuencias
23 Tabla_frec_agrup=function(x,k,a,p){
24   L=min(x)-p/2+A*(0:k)
25   x_int=cut(x,breaks = L,right = FALSE)
26   intervalos=levels(x_int)
27   marcas=(L[1]+L[2]/2+A*(0:(k-1)))
28   f.abs=as.vector(table(x_int))
29   f.rel=f.abs/length(x)
30   f.abs.cum=cumsum(f.abs)
31   f.rel.cum=cumsum(f.rel)
32   tabla_x=data.frame(intervalos ,marcas ,f.abs ,f.abs.cum ,f.rel ,f
33     .rel.cum)
34   tabla_x
35 }
36 tabla=Tabla_frec_agrup(Chile2$age,k,A,1)
37 > tabla
38   intervalos  marcas  f.abs  f.abs.cum      f.rel  f.rel.cum
39 1 [ 17.5 ,21.5)  28.25   310      310  0.12751954  0.1275195
40 2 [ 21.5 ,25.5)  32.25   302      612  0.12422871  0.2517483
41 3 [ 25.5 ,29.5)  36.25   238      850  0.09790210  0.3496503
42 4 [ 29.5 ,33.5)  40.25   207     1057  0.08515014  0.4348005
43 5 [ 33.5 ,37.5)  44.25   259     1316  0.10654052  0.5413410
44 6 [ 37.5 ,41.5)  48.25   204     1520  0.08391608  0.6252571
45 7 [ 41.5 ,45.5)  52.25   193     1713  0.07939120  0.7046483

```



```

45 8 [ 45.5 ,49.5) 56.25 134 1847 0.05512135 0.7597696
46 9 [ 49.5 ,53.5) 60.25 111 1958 0.04566022 0.8054299
47 10 [ 53.5 ,57.5) 64.25 115 2073 0.04730564 0.8527355
48 11 [ 57.5 ,61.5) 68.25 136 2209 0.05594406 0.9086796
49 12 [ 61.5 ,65.5) 72.25 98 2307 0.04031263 0.9489922
50 13 [ 65.5 ,69.5) 76.25 79 2386 0.03249691 0.9814891
51 14 [ 69.5 ,73.5) 80.25 45 2431 0.01851090 1.0000000
52 #Cálculo estadísticos
53 Total=sum(tabla$f.abs)
54 > Total
55 [1] 2431
56 #Media
57 Media=sum(tabla$f.abs*tabla$marcas)/Total
58 > Media
59 [1] 47.08176
60 Media.sin.agrupar=mean(Chile2$age)
61 > Media.sin.agrupar
62 [1] 38.29
63 #Varianza
64 Varianza=sum(tabla$f.abs*tabla$marcas^2)/Total - Media^2
65 > Varianza
66 [1] 216.2115
67 Varianza.sin.agrupar=(Total - 1)/Total*var(Chile2$age)
68 > Varianza.sin.agrupar
69 [1] 215.0377
70 #Desviación típica
71 Desv.tip=sqrt(Varianza)
72 > Desv.tip
73 [1] 14.70413
74 #Intervalo modal
75 Int.modal=tabla$intervalos [which(tabla$f.abs==max(tabla$f.abs)
76 )]
77 > Int.modal
78 [1] [ 17.5 ,21.5)
79 14 Levels: [ 17.5 ,21.5) [ 21.5 ,25.5) [ 25.5 ,29.5) [ 29.5 ,33.5) ...
80 [ 69.5 ,73.5)

```

### 12.2.2. Cuantiles de datos agrupados

Para calcular el cuantil  $Q_p$  de orden  $p$ , se requiere el **Intervalo crítico**, que es el primer intervalo con frecuencia relativa acumulada  $\geq p$ . Sea este  $[L_c, L_{c+1})$ . Entonces el cuantil de orden  $p$  viene dado por:

$$Q_p = L_c + A_c \cdot \frac{pn - N_{c-1}}{n_c}$$

Donde:

- $A_c$  es la amplitud del intervalo crítico.
- $n$  es el total de datos.
- $N_{c-1}$  es la frecuencia absoluta acumulada del intervalo anterior al crítico.
- $nc$  es la frecuencia absoluta del intervalo crítico.

Vamos a seguir con el ejemplo anterior para sumarle estos datos estadísticos:

Listing 12.4: Código fórmulas estadísticas aplicadas al data frame Chile

```

1 #Mediana
2 crit=min(which(tabla$f.rel.cum>=0.5))
3 > crit
4 [1] 5
5 > tabla$intervalos[ crit ]
6 [1] [33.5,37.5)
7 14 Levels: [17.5,21.5) [21.5,25.5) [25.5,29.5) [29.5,33.5) ...
      [69.5,73.5)
8 Lc=tabla$marcas[ crit ]-A/2
9 > Lc
10 [1] 42.25
11 n=Total
12 > n
13 [1] 2431
14 nc=tabla$f.abs[ crit ]
15 > nc
16 [1] 259
17 Nc1=tabla$f.abs.cum[ crit -1 ]
18 > Nc1
19 [1] 1057
20 Mediana=Lc+A*(0.5*n-Nc1)/nc
21 > Mediana
22 [1] 44.69788
23 median(Chile2$age)
24 [1] 36

```

## 12.3. Histogramas

En este capítulo vamos a ver cómo generar histogramas con R y cómo mejorarlos.

Los histogramas son unos diagramas de barras con los que se describen gráficamente los datos agrupados. Cada barra representa una clase cuya frecuencia determina el área de la barra.

- **Histogramas de frecuencias absolutas:** La altura de cada barra es la necesaria para que el área sea igual a la frecuencia absoluta de cada clase. **Histogramas de frecuencias relativas:** La altura de cada barra es la necesaria para que el área sea igual a la frecuencia relativa de la clase.

- **Histogramas de frecuencias acumuladas (absolutas o relativas):** Las alturas de las barras son iguales a las frecuencias acumuladas de las clases.

### 12.3.1. Histogramas en R

La función para dibujar histogramas con R es `hist(x,breaks=...,freq=...,right=...,...)` donde:

- `x` es el vector de datos.
- `breaks` establece los valores de los extremos de los intervalos, el número de intervalos o el método para calcular las clases.
- `freq` igualado a TRUE produce el histograma de frecuencias absolutas. Igualado a FALSE de frecuencias relativas.
- `right` igualado a FALSE deja los intervalos abiertos por la derecha.
- Otros parámetros usuales de la función `plot`.

El resultado de `hist` es una list con las siguientes componentes:

- `breaks`: Da los extremos de los intervalos.
- `mids`: Da los puntos medios de los intervalos.
- `counts`: Da las frecuencias absolutas de los intervalos.
- `density`: Da las densidades de los intervalos.

Para ver los histogramas, vamos a volver a utilizar el data frame `Chile`:

Listing 12.5: Código histogramas

```

1 data(Chile)
2 head(Chile)
3   region population sex age education income statusquo vote
4 1      N    175000  M  65          P   35000    1.00820    Y
5 2      N    175000  M  29          PS    7500   -1.29617    N
6 3      N    175000  F  38          P   15000    1.23072    Y
7 4      N    175000  F  49          P   35000   -1.03163    N
8 5      N    175000  F  23          S   35000   -1.10496    N
9 6      N    175000  F  28          P    7500   -1.04685    N
10 Chile2=na.omit(Chile)
11 k=14
12 A=4
13 L=min(Chile2$age)-0.5+A*(0:14)
14 #Histograma
15 hist(Chile2$age,breaks=L,right=FALSE,main="Histograma de
    frecuencias absolutas",xlab="Intervalos",ylab="Frecuencias
    absolutas")

```

```
16 h=hist(Chile2$age,breaks=L, right=FALSE, plot=FALSE)
17 h$breaks
18 [1] 17.5 21.5 25.5 29.5 33.5 37.5 41.5 45.5 49.5 53.5 57.5
    61.5 65.5 69.5 73.5
19 h$mids
20 [1] 19.5 23.5 27.5 31.5 35.5 39.5 43.5 47.5 51.5 55.5 59.5
    63.5 67.5 71.5
21 h$counts
22 [1] 310 302 238 207 259 204 193 134 111 115 136  98  79  45
23 #Histograma de frecuencias absolutas
24 hist_abs=function(x,L){
25   h=hist(x,breaks=L, right=FALSE, freq=FALSE,
26         xaxt="n", yaxt="n", col="lightgray",
27         main="Histograma de frecuencias absolutas",
28         xlab="Intervalos y marcas de clase",
29         ylab="Frecuencias absolutas")
30   axis(1,at=L)
31   text(h$mids,h$density/2,labels=h$counts,col="blue" )
32 }
33 hist_abs(Chile2$age,L)
34 #Histograma de frecuencias absolutas acumuladas
35 hist_abs.cum=function(x,L){
36   h=hist(x,breaks=L, right=FALSE, plot=FALSE)
37   h$density=cumsum(h$density)
38   plot(h,freq=FALSE,xaxt="n", yaxt="n", col="lightgray",
39        main="Histograma de frecuencias absolutas acumuladas",
40        xlab="Intervalos",ylab="Frec.absolutas acumuladas")
41   axis(1,at=L)
42   text(h$mids,h$density/2,labels=cumsum(h$counts),col="blue" )
43 }
44 hist_abs.cum(Chile2$age,L)
```

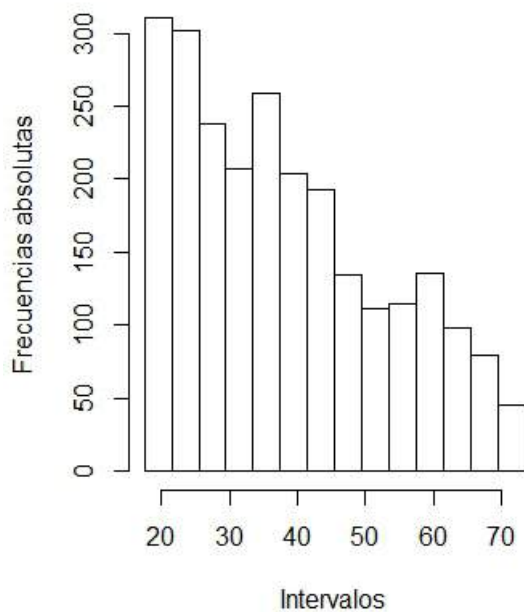
**Histograma de frecuencias absolutas**

Figura 12.1: Histograma de frecuencias absolutas

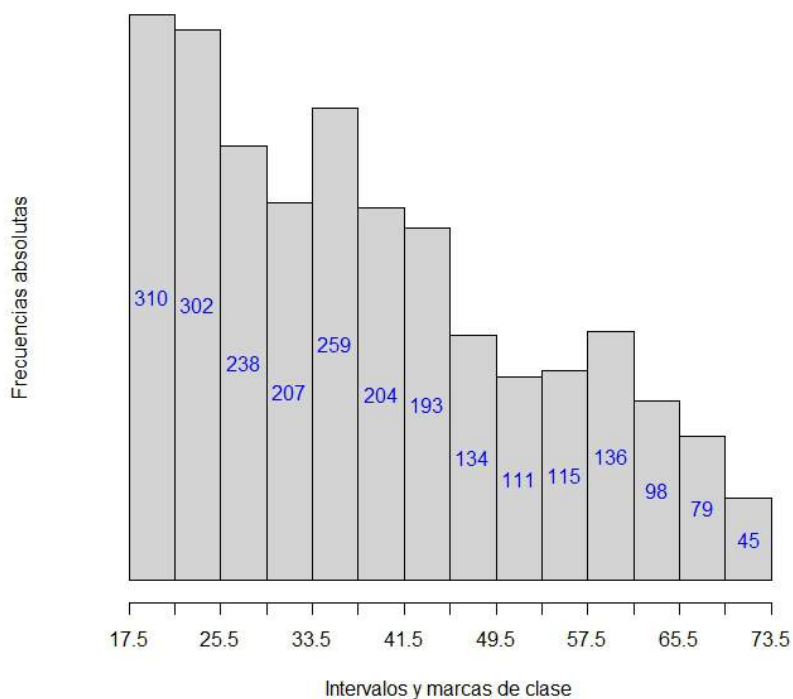
**Histograma de frecuencias absolutas**

Figura 12.2: Histograma de frecuencias absolutas con colores y datos frecuencias

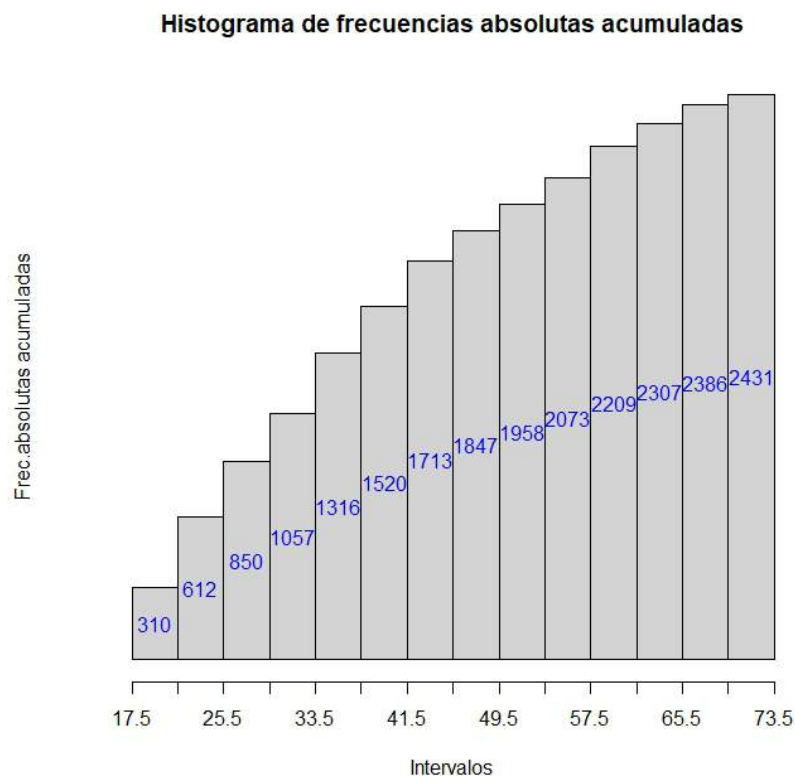


Figura 12.3: Histograma de frecuencias absolutas acumuladas

### 12.3.2. Densidad de una variable

La densidad de una variable es una curva tal que el área comprendida entre el eje de abscisas y la curva sobre un intervalo es igual a la fracción de individuos de la población que caen dentro de ese intervalo.

Para estimar la densidad de la distribución a partir de una muestra hay que usar la función `density`.

Se suele incluir en los histogramas de frecuencias relativas.

#### Función de distribución de una variable

La función de distribución de una variable nos da, en cada punto, la frecuencia relativa acumulada por la variable sobre la población en ese punto.

Para estimar la función de distribución hay que ir acumulando los valores dados por `density`.

Se suele incluir en los histogramas de frecuencias relativas acumuladas.

Vamos a ver la densidad del ejemplo anterior:

Listing 12.6: Código cálculo densidad

```

1 #Densidad
2 plot(density(Chile2$age), type="l", main="Densidad de la
   variable edad", xlab="Edad", ylab="Densidad")

```

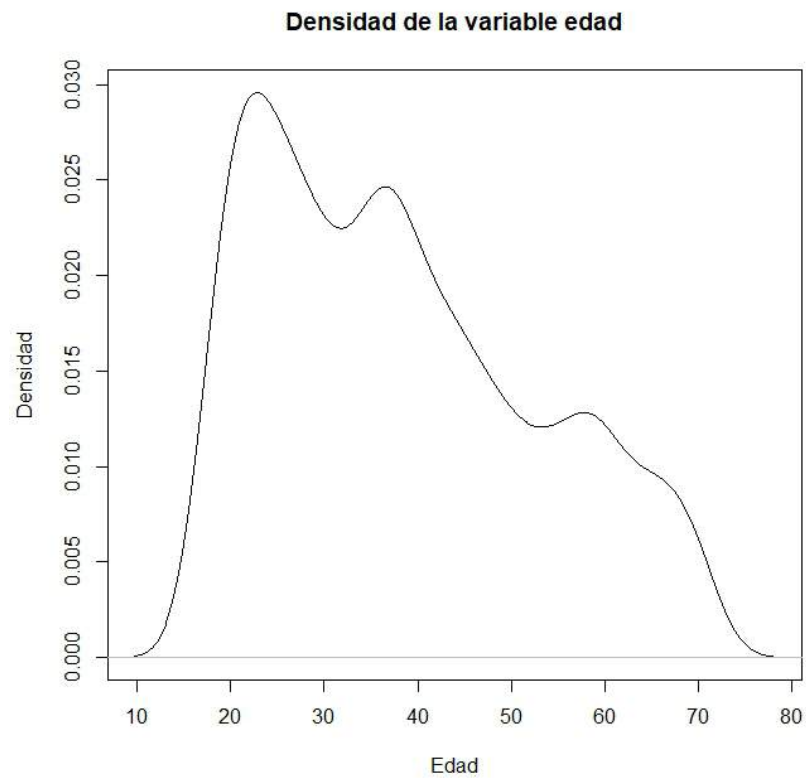


Figura 12.4: Densidad de la variable edad

Se podrá dibujar la curva de densidad junto con el histograma de frecuencias acumuladas.





# Índice de figuras

1.1. Página web RProject . . . . .	1
1.2. CRAN de España . . . . .	2
1.3. Instalación de R . . . . .	2
1.4. Idioma instalación R . . . . .	2
1.5. Localización de R . . . . .	3
1.6. Tareas de R . . . . .	3
1.7. Web RStudio . . . . .	4
1.8. RStudio Desktop . . . . .	4
1.9. Plataformas disponibles . . . . .	5
1.10. Interfaz RStudio . . . . .	5
1.11. Paneles . . . . .	6
1.12. Pestaña ayuda . . . . .	7
1.13. Editor RStudio . . . . .	8
1.14. Programa sencillo a ejecutar . . . . .	9
1.15. Codificación programa . . . . .	9
1.16. Nombre programa . . . . .	10
1.17. Salvar Workspace . . . . .	10
1.18. Variables salvadas . . . . .	10
1.19. Paquetes pendientes de cargar . . . . .	11
1.20. Instalar paquetes . . . . .	12
1.21. Instalar paquetes- Consola . . . . .	13
1.22. Paquete ya cargado . . . . .	13
2.1. Suma básica . . . . .	15
2.2. Operación con y sin paréntesis . . . . .	16
2.3. Entorno funciones . . . . .	20
2.4. Funciones números complejos . . . . .	21
3.1. Representación gráfica . . . . .	24
3.2. Recta regresión ejemplo . . . . .	27
3.3. Importar datos . . . . .	27
3.4. Importar datos web . . . . .	28
3.5. Datos ya importados . . . . .	28
3.6. Regresión nueva . . . . .	30
3.7. Gráfica función logarítmica . . . . .	31
3.8. Representación más lineal . . . . .	32
3.9. Representación semilogarítmica . . . . .	33
3.10. Representación de la curva . . . . .	35
3.11. Función exponencial . . . . .	36

3.12. Representación más lineal . . . . .	37
3.13. Curva tras reconversión parámetros . . . . .	39
6.1. Data frame iris . . . . .	70
6.2. Editor de datos . . . . .	79
7.1. Función plot(x,y) . . . . .	91
7.2. Función plot(x) . . . . .	92
7.3. Función plot(f) . . . . .	92
7.4. Figura modificada márgenes y aspecto exterior . . . . .	94
7.5. Número con estilo de punto . . . . .	94
7.6. Primera modificación puntos . . . . .	95
7.7. Segunda modificación puntos . . . . .	96
7.8. Primera modificación del gráfico . . . . .	97
7.9. Segunda modificación del gráfico . . . . .	97
7.10. Tercera modificación del gráfico . . . . .	98
7.11. Cuarta modificación del gráfico . . . . .	99
7.12. Quinta modificación del gráfico . . . . .	99
7.13. Sexta modificación del gráfico . . . . .	100
7.14. Séptima modificación del gráfico . . . . .	101
7.15. Gráfico modificado tipo l . . . . .	102
7.16. Gráfica modificada ejes . . . . .	103
7.17. Gráfica con punto añadido . . . . .	104
7.18. Gráfica con punto añadido y recta añadida . . . . .	105
7.19. Gráfica con punto añadido, recta añadida y curva añadida . . . . .	106
7.20. Gráfica con punto añadido, recta añadida, curva añadida y texto añadido	107
7.21. Gráfica con leyenda añadida . . . . .	108
8.1. Diagrama de barras de las frecuencias relativas . . . . .	118
8.2. Diagrama de barras bidimensional . . . . .	119
8.3. Diagrama de barras bidimensional datos enfrentados y leyenda . . . . .	119
8.4. Diagrama circular . . . . .	120
8.5. Gráfico mosaico . . . . .	121
8.6. Gráfico mosaico de la tabla HairEyeColor . . . . .	122
8.7. Diagrama mosaico ejemplo . . . . .	125
8.8. Diagrama de barras frecuencias relativas color ojos . . . . .	125
8.9. Diagrama de barras frecuencias relativas color cabello . . . . .	126
8.10. Diagrama de barras frecuencias relativas color cabello en cada ojos . . . . .	126
8.11. Diagrama de barras frecuencias relativas color ojos en cada color cabello	127
8.12. Gráfico mosaico de la tabla HairEyeColor . . . . .	127
10.1. Ejemplo de diagrama de caja . . . . .	136
10.2. Diagrama de caja de peso . . . . .	137
10.3. Diagrama de caja de peso según la dieta . . . . .	138
10.4. Diagrama de caja de peso según dieta con notch=TRUE . . . . .	139
11.1. Dos variables en una función plot . . . . .	147
11.2. Tres variables en la función scatterplot3d . . . . .	148
11.3. Diagramas de dispersión con plot . . . . .	149
11.4. Diagramas de dispersión con plot distinguiendo variable . . . . .	150

---

11.5. Diagramas de dispersión con pairs . . . . .	151
11.6. Diagramas de dispersión con spm . . . . .	152
12.1. Histograma de frecuencias absolutas . . . . .	163
12.2. Histograma de frecuencias absolutas con colores y datos frecuencias . . .	163
12.3. Histograma de frecuencias absolutas acumuladas . . . . .	164
12.4. Densidad de la variable edad . . . . .	165



# Índice de cuadros

2.1. Tabla de operaciones básicas con comandos . . . . .	15
2.2. Funciones básicas . . . . .	16
4.1. Signos lógicos . . . . .	49
6.1. Estructura general data frame . . . . .	69
8.1. Tabla de frecuencias del ejemplo . . . . .	109
8.2. Tabla de respuestas Si o no de hombre y mujer . . . . .	111
8.3. Tabla bidimensional de frecuencias . . . . .	111
8.4. Tabla con tres variables . . . . .	112
8.5. Tabla tridimensional . . . . .	112
11.1. Representación de datos cuantitativos . . . . .	141



# Listings

1.1. Código programa a ejecutar . . . . .	8
2.1. Código operaciones básicas . . . . .	16
2.2. Código de operaciones en grados y radianes . . . . .	17
2.3. Notación científica por defecto de R . . . . .	17
2.4. Código para cifras significativas . . . . .	17
2.5. Código función round . . . . .	18
2.6. Código funciones redondeo . . . . .	18
2.7. Código de definición de variable . . . . .	18
2.8. Código de cambio de valor de variable . . . . .	19
2.9. Código de función sencilla . . . . .	19
2.10. Código de función de dos variables . . . . .	19
2.11. Código de función con dos definiciones . . . . .	19
2.12. Código borrar . . . . .	20
2.13. Código números complejos . . . . .	20
2.14. Código operaciones números complejos . . . . .	21
2.15. Código del conjugado, módulo y argumento del número complejo . . . . .	22
3.1. Código de recogida de datos en vectores . . . . .	23
3.2. Código data.frame . . . . .	23
3.3. Código acceso datos columna edad . . . . .	24
3.4. Código dibujar gráficas . . . . .	24
3.5. Código método lineal . . . . .	25
3.6. Código summary . . . . .	25
3.7. Código cálculo recta de regresión completo . . . . .	26
3.8. Código importar datos . . . . .	29
3.9. Código recta regresión . . . . .	29
3.10. Código función logarítmica . . . . .	31
3.11. Código representación más lineal . . . . .	31
3.12. Código representación más lineal . . . . .	32
3.13. Código de regresión lineal . . . . .	33
3.14. Código curva . . . . .	34
3.15. Código función exponencial . . . . .	35
3.16. Código recta más lineal . . . . .	36
3.17. Código regresión lineal . . . . .	37
3.18. Código curva tras reconversión de parámetros . . . . .	38
4.1. Código función c . . . . .	41
4.2. Código de funciones concatenadas con c . . . . .	41
4.3. Código vectores función sep . . . . .	42
4.4. Código función seq . . . . .	43
4.5. Código función scan . . . . .	44

4.6. Código separación scan . . . . .	45
4.7. Código progresión aritmética . . . . .	46
4.8. Código definición función . . . . .	46
4.9. Código función sapply . . . . .	46
4.10. Código multiplicación entradas . . . . .	46
4.11. Código funciones específicas . . . . .	47
4.12. Código entrada vector . . . . .	48
4.13. Código condicionales . . . . .	49
4.14. Código vector con entradas vacías . . . . .	50
4.15. Código nuevo vector sin entradas vacías . . . . .	50
4.16. Código borrar datos a mano . . . . .	51
4.17. Código funciones no admiten parámetro . . . . .	51
4.18. Código diferencias factor de vector . . . . .	51
4.19. Código de definición de factores . . . . .	52
4.20. Código factor ordenado por niveles . . . . .	52
4.21. Código vectores homogéneos . . . . .	53
4.22. Código creación nuevas listas . . . . .	54
4.23. Código extracción objetos de la list . . . . .	54
5.1. Código construcción matrices . . . . .	57
5.2. Código concatenación vectores y matrices . . . . .	58
5.3. Código entrada matriz . . . . .	59
5.4. Código distinción filas y columnas . . . . .	60
5.5. Código matriz diagonal . . . . .	60
5.6. Código funciones básicas dimensiones . . . . .	61
5.7. Código funciones básicas comunes vectores . . . . .	61
5.8. Código función elemento a elemento . . . . .	62
5.9. Código operaciones algebraicas con matrices . . . . .	63
5.10. Código solución sistema compatible determinado . . . . .	64
5.11. Código función valores y vectores propios . . . . .	65
5.12. Código descomposición canónica . . . . .	67
6.1. Código visualización data frame . . . . .	70
6.2. Código mostrar primeras y últimas filas . . . . .	70
6.3. Código dimensiones data frame . . . . .	72
6.4. Código extracción datos data frame . . . . .	74
6.5. Código importar data frame desde fichero y web . . . . .	75
6.6. Código de creación data frame . . . . .	77
6.7. Código función fix . . . . .	78
6.8. Código modificadores data frame . . . . .	79
6.9. Código modificar variable . . . . .	81
6.10. Código añadir filas . . . . .	82
6.11. Código modificador columnas . . . . .	82
6.12. Código selección de trozos . . . . .	83
6.13. Código de la función select . . . . .	85
6.14. Código función subset . . . . .	86
6.15. Código función sapply . . . . .	87
6.16. Código función aggregate . . . . .	88
6.17. Código ejemplo paquete alr4 . . . . .	89
6.18. Código añadir variables de un data frame al entorno global . . . . .	89



7.1. Código usos función plot . . . . .	91
7.2. Código usos función plot . . . . .	92
7.3. Código usos función plot . . . . .	92
7.4. Código modificación aspecto exterior . . . . .	93
7.5. Código modificación aspecto de los puntos . . . . .	94
7.6. Código modificación aspecto de los puntos . . . . .	95
7.7. Código modificando el tipo de gráfico . . . . .	96
7.8. Código modificando el tipo de gráfico . . . . .	97
7.9. Código modificando el tipo de gráfico . . . . .	98
7.10. Código modificando el tipo de gráfico . . . . .	98
7.11. Código modificando el tipo de gráfico . . . . .	99
7.12. Código modificando el tipo de gráfico . . . . .	100
7.13. Código modificando el tipo de gráfico . . . . .	100
7.14. Código modificación gráfico estilo l . . . . .	101
7.15. Código modificando ejes de coordenadas . . . . .	102
7.16. Código añadiendo puntos . . . . .	103
7.17. Código añadir recta . . . . .	104
7.18. Código añadir curva . . . . .	105
7.19. Código añadir texto . . . . .	106
7.20. Código añadir leyenda . . . . .	107
8.1. Código tabla unidimensional de frecuencia . . . . .	110
8.2. Código tabla bidimensional . . . . .	111
8.3. Código frecuencias relativas globales y marginales . . . . .	111
8.4. Código tabla tridimensional . . . . .	112
8.5. Código de tabla de frecuencias a partir de data frames . . . . .	116
8.6. Código del diagrama de barras . . . . .	117
8.7. Código función barplot tabla bidimensional . . . . .	118
8.8. Código diagrama circular . . . . .	119
8.9. Código gráfico mosaico . . . . .	120
8.10. Código gráfico mosaico HairEyeColor . . . . .	121
8.11. Código ejemplo completo . . . . .	122
10.1. Código medidas frecuencias pases . . . . .	131
10.2. Código medidas de tendencia central . . . . .	132
10.3. Código cálculo cuantiles . . . . .	133
10.4. Código medidas de dispersión . . . . .	134
10.5. Código resumen estadístico . . . . .	135
10.6. Código diagramas de cajas . . . . .	137
10.7. Código diagrama de caja con parámetros modificados . . . . .	138
11.1. Código estadísticos multidimensionales . . . . .	142
11.2. Código tipificación tabla de datos . . . . .	143
11.3. Código de cálculo de covarianza . . . . .	144
11.4. Código matrices covarianzas y covarianzas muestrales . . . . .	144
11.5. Código cálculo correlaciones . . . . .	146
11.6. Código dos variables en función plot . . . . .	146
11.7. Código tres variables en función scatterplot3d . . . . .	147
11.8. Código diagramas de dispersión con plot . . . . .	148
11.9. Código diagramas de dispersión con plot distinguido variable . . . . .	149
11.10. Código diagrama dispersión con función pairs . . . . .	150

---

11.11 Código diagrama dispersión con función spm . . . . .	151
12.1. Código agrupamiento de datos . . . . .	154
12.2. Código agrupamiento de datos con R . . . . .	155
12.3. Código fórmulas estadísticas aplicadas al data frame Chile . . . . .	158
12.4. Código fórmulas estadísticas aplicadas al data frame Chile . . . . .	160
12.5. Código histogramas . . . . .	161
12.6. Código cálculo densidad . . . . .	164

PASCAL

ENERO 2018

Ult. actualización 4 de mayo de 2019

TEX lic. LPPL & powered by **TEFLON** CC-ZERO

Este documento esta realizado bajo licencia Creative Commons “CC0 1.0 Universal”.

