

Introducción a R y RStudio

Sam Rogers. Traducido por Sabela Muñoz

Índice

| | |
|--|-----------|
| Resumen del curso | 3 |
| Introducción a RStudio | 3 |
| Consola de R | 3 |
| Script | 3 |
| Entorno e Historia | 3 |
| Archivos, Gráficos, Paquetes, Ayuda y Visualizador | 6 |
| Sintaxis básica | 9 |
| Uso como calculadora | 9 |
| Práctica | 10 |
| Variables | 11 |
| Asignación de nombres y puntuación | 11 |
| Scripts | 11 |
| Establecer el directorio de trabajo | 11 |
| Práctica | 12 |
| Vectores | 12 |
| Obtención de los elementos de un vector | 13 |
| Práctica | 14 |
| Funciones | 14 |
| Práctica | 15 |
| Otro tipo de datos | 15 |
| Valores lógicos | 16 |
| Matrices | 16 |
| Arrays | 17 |
| Listas | 18 |
| Práctica | 21 |
| Gestión de datos | 22 |
| Qué no hacer | 22 |
| Guardando los datos | 22 |
| Pensando en filas y columnas | 22 |
| Poniendo nombre a las columnas | 23 |
| Formateando los valores | 23 |
| Fechas | 24 |
| Números | 24 |
| Colores, subrayado, negrita y celdas combinadas | 24 |
| Archivos CSV | 24 |
| Datos no disponibles | 25 |

| | |
|---|-----------|
| Base de datos | 25 |
| Acceso a elementos de una base de datos | 26 |
| Leyendo los datos | 27 |
| Práctica | 28 |
| Funciones más avanzadas | 29 |
| Más tipos de datos | 29 |
| Matrices | 29 |
| Lista | 29 |
| Un objeto de distinto tipo: t test | 31 |
| Funciones de la familia “apply” | 32 |
| Práctica (Opcional) | 33 |
| Gráficos en R | 33 |
| Diagrama de dispersión | 33 |
| Histogramas | 35 |
| Diagrama de cajas | 37 |
| También hecho en R | 40 |
| El conjunto de Mandelbrot | 40 |
| Círculos concéntricos | 41 |
| Doble corazón | 42 |
| Práctica | 42 |
| Más recursos | 42 |
| Recursos para el autoaprendizaje | 42 |
| Libros | 43 |
| Gestión y limpieza de datos | 44 |
| Otra información | 44 |
| Agradecimientos | 44 |

Este taller es parte del programa de formación de *Statistics for Australian Grains Industry* en la Universidad de Adelaide, que está cofinanciado por la entidad *Australian Grains Research and Development Corporation*, proyecto UA00164. El material debe utilizarse estrictamente para fines de formación y queda prohibida su distribución y modificación.

Resumen del curso

En este curso hacemos una breve introducción al lenguaje de R utilizando la interfaz RStudio. En primer lugar, daremos una [visión general de RStudio](#). Seguidamente, explicaremos la sintaxis y [los comandos básicos de R](#) para posteriormente trabajar con los [distintos tipos de objetos en R](#). A continuación, daremos algunas pautas para [gestionar datos en R](#) y para [leer y escribir bases de datos](#). En la sección final, haremos una breve introducción a la creación de [gráficos en R](#).

Nota: Uno de los requerimientos del curso es tener R y RStudio instalado en el equipo. Por favor, sigue las instrucciones de la guía *Instalación de R y RStudio.pdf* en la carpeta “Materiales” si necesitas instalar alguno de estos programas.

Introducción a RStudio

RStudio está dividido en 4 paneles, cada uno de ellos tiene una o más ventanas:

Consola de R

La consola de R se encuentra por defecto en el panel de abajo a la izquierda. En la consola es donde se ejecutan los comandos de R.

- El símbolo > indica que R está preparado para ejecutar un comando.
- El símbolo + al principio de la línea indica que R está esperando recibir más información para poder ejecutar el comando. Necesitamos completar el comando o presionar escape (Esc) para volver a >. Por ejemplo:

```
> max(c(1,2)
+           #(R está esperando completar el comando - un paréntesis ) o Esc)
```

- Podemos escribir directamente en la consola de R o en el panel denominado **Script** y presionar el botón **Run** para ejecutar el código en la consola de R.

Script

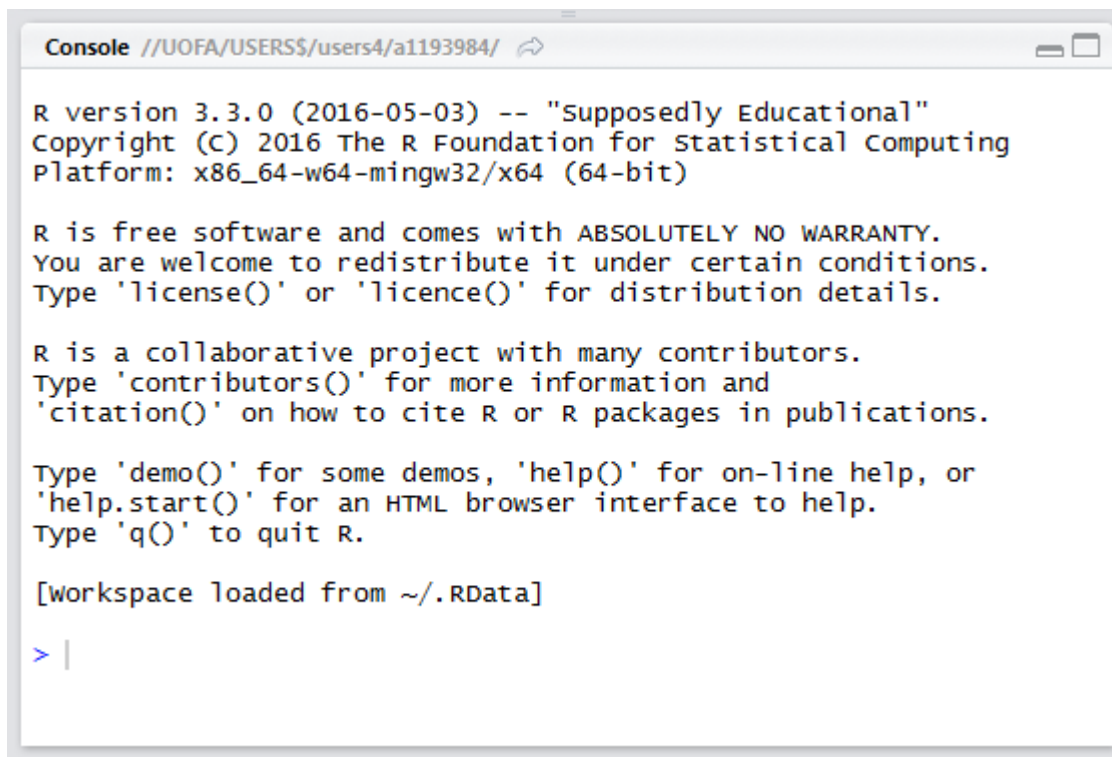
El Script se encuentra por defecto en el panel de arriba a la izquierda. En el Script es donde escribimos comandos y líneas de código. Las líneas de código no se ejecutan en este panel sino en la consola. Podemos guardar el Script en un archivo cuya extensión es .R.

- **Nota:** Si es la primera vez que abrimos RStudio, la ventana del Script no se encuentra visible. Pulsa el botón señalado en la Figura 3 para expandir la ventana del Script.

Entorno e Historia

Las ventanas de Entorno e Historia están localizadas en el panel de arriba a la derecha.

- La ventana del Entorno (Environment) muestra los datos que se han importado o los objetos que se han generado
- La ventana Historia (History) nos proporciona una lista de todo lo que se ha hecho hasta el momento. Por ejemplo si ejecutamos



The screenshot shows a console window titled "Console //UOFA/USERS\$/users4/a1193984/". The text inside the console is the standard R startup message, including the version (3.3.0), copyright (2016), platform (x86_64-w64-mingw32/x64), and instructions on how to use R, such as typing 'license()', 'contributors()', 'citation()', 'demo()', 'help()', 'help.start()', and 'q()'.

```
R version 3.3.0 (2016-05-03) -- "Supposedly Educational"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[workspace loaded from ~/.RData]

> |
```

Figura 1: Consola al empezar

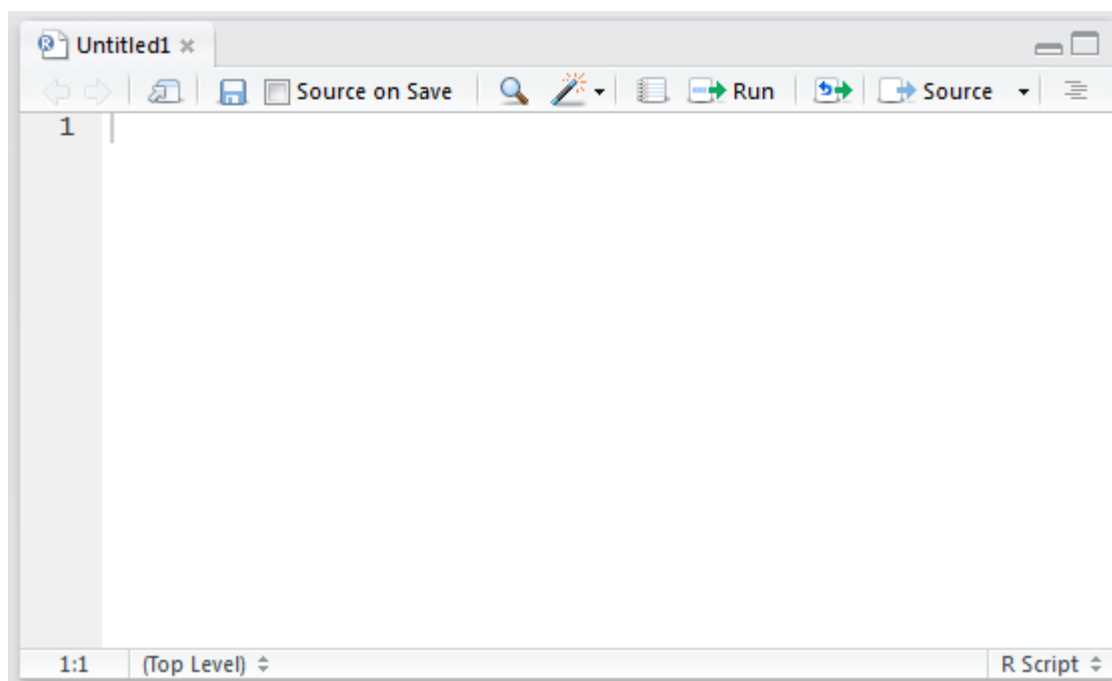


Figura 2: Script vacío

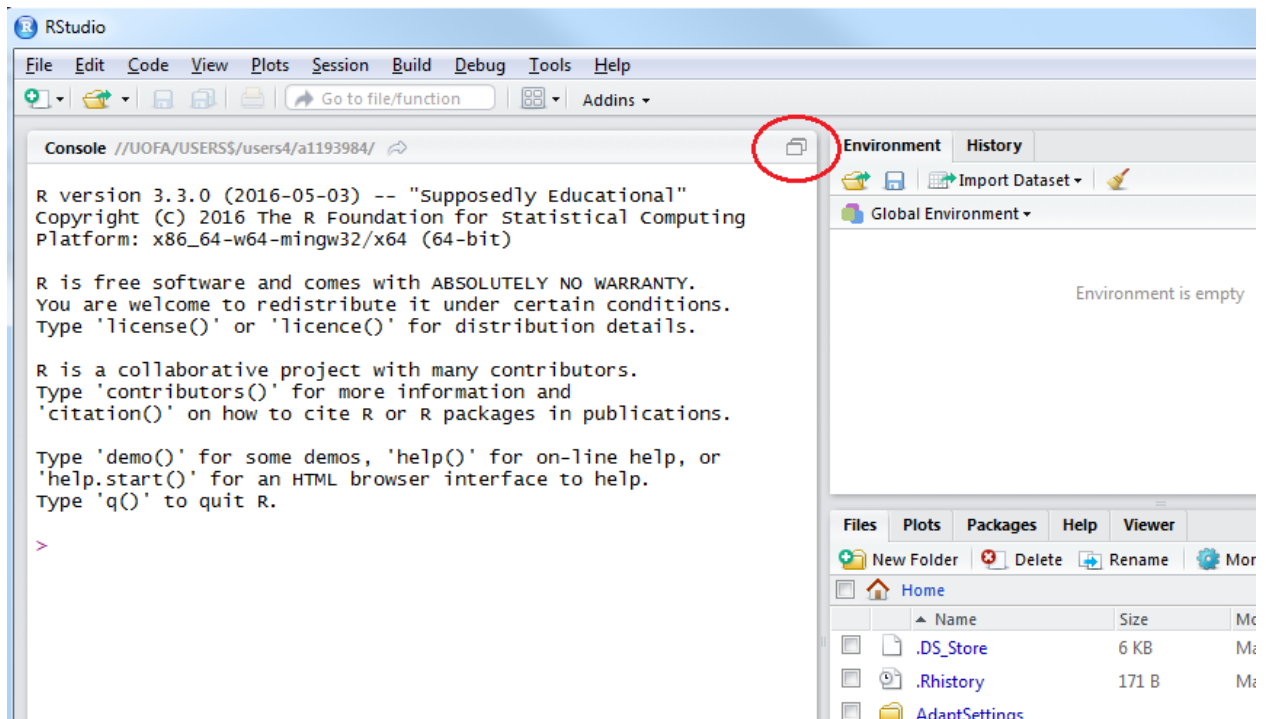


Figura 3: Establecer la ventana del Script

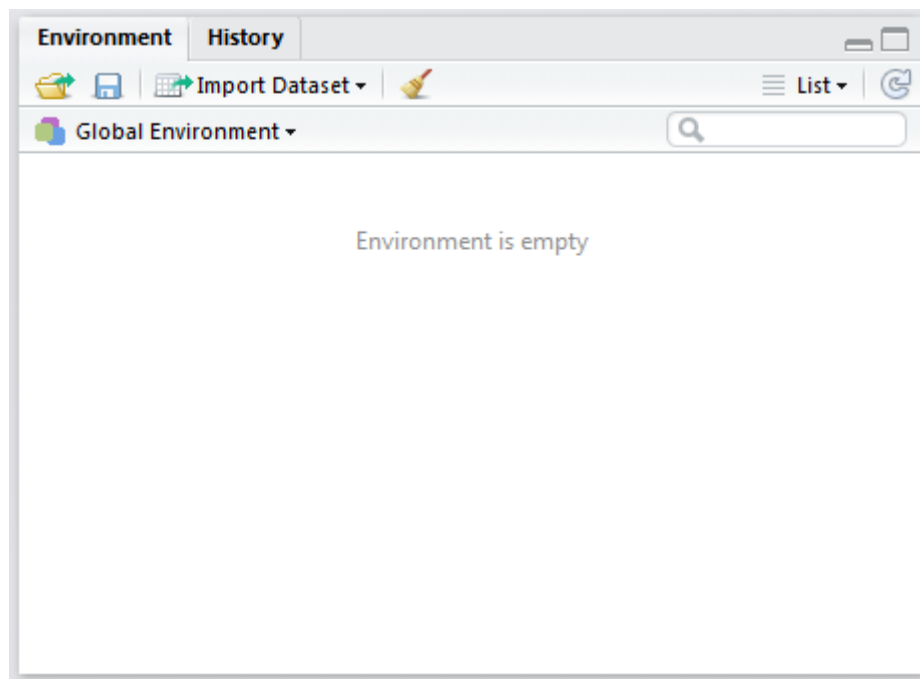


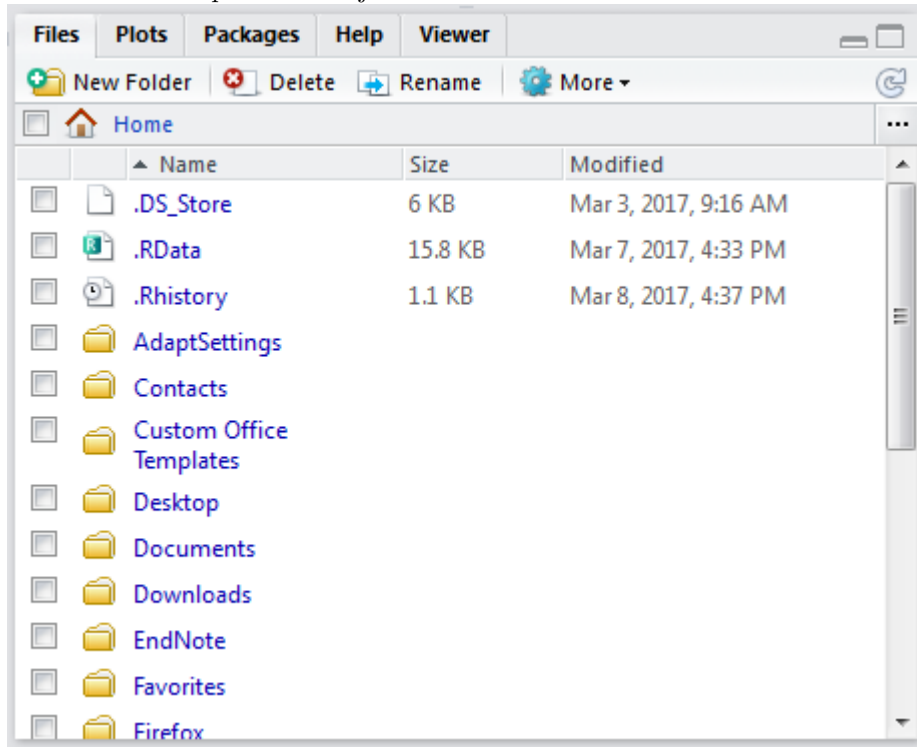
Figura 4: Entorno al empezar

```
A <- letters[1:24]
```

dicha línea aparecerá en la historia.

Archivos, Gráficos, Paquetes, Ayuda y Visualizador

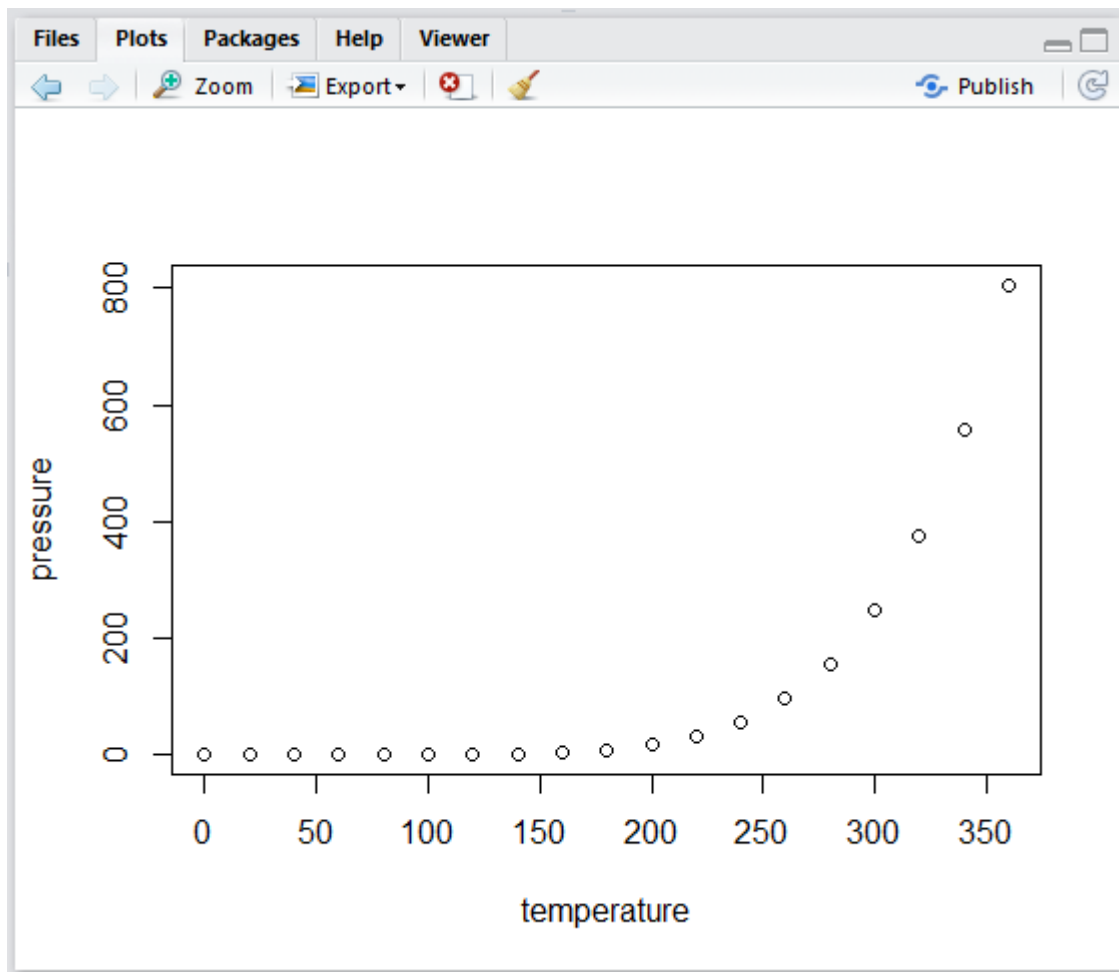
La ventana de Archivos, Gráficos, Paquetes, Ayuda y Visualizador (*Files, Plots, Packages, Help and Viewer*) se localizan en el panel de abajo a la derecha.



De estas ventanas, las tres más útiles son:

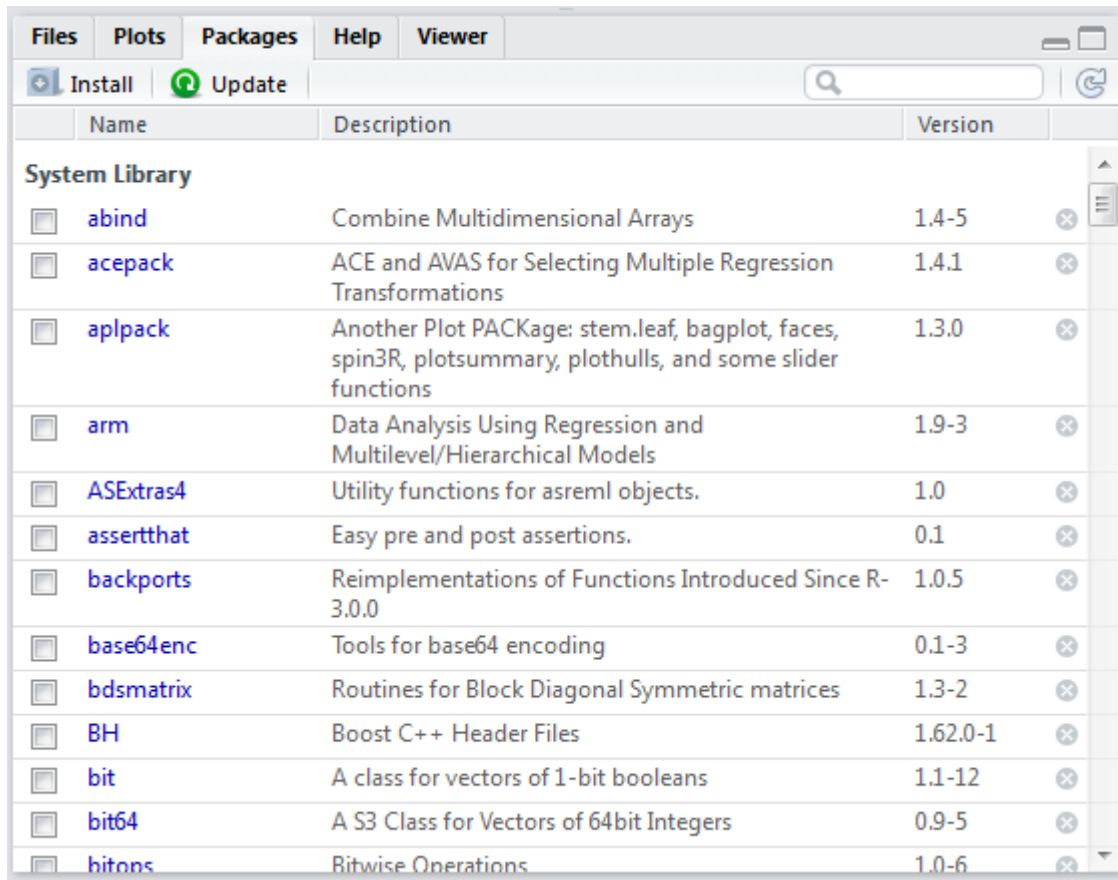
1. **Gráficos** es la ventana donde se generan los gráficos. Por ejemplo:

```
plot(pressure)
```



- Selecciona *Export* para guardar las gráficas, *Zoom* para verlas más grandes y *Clear all* para borrar todas las gráficas producidas hasta el momento. Utiliza las flechas para visualizar las gráficas que has ido generando.

2. **Paquetes (*Packages*)** muestra los paquetes de R que se encuentran instalados en tu librería de R.

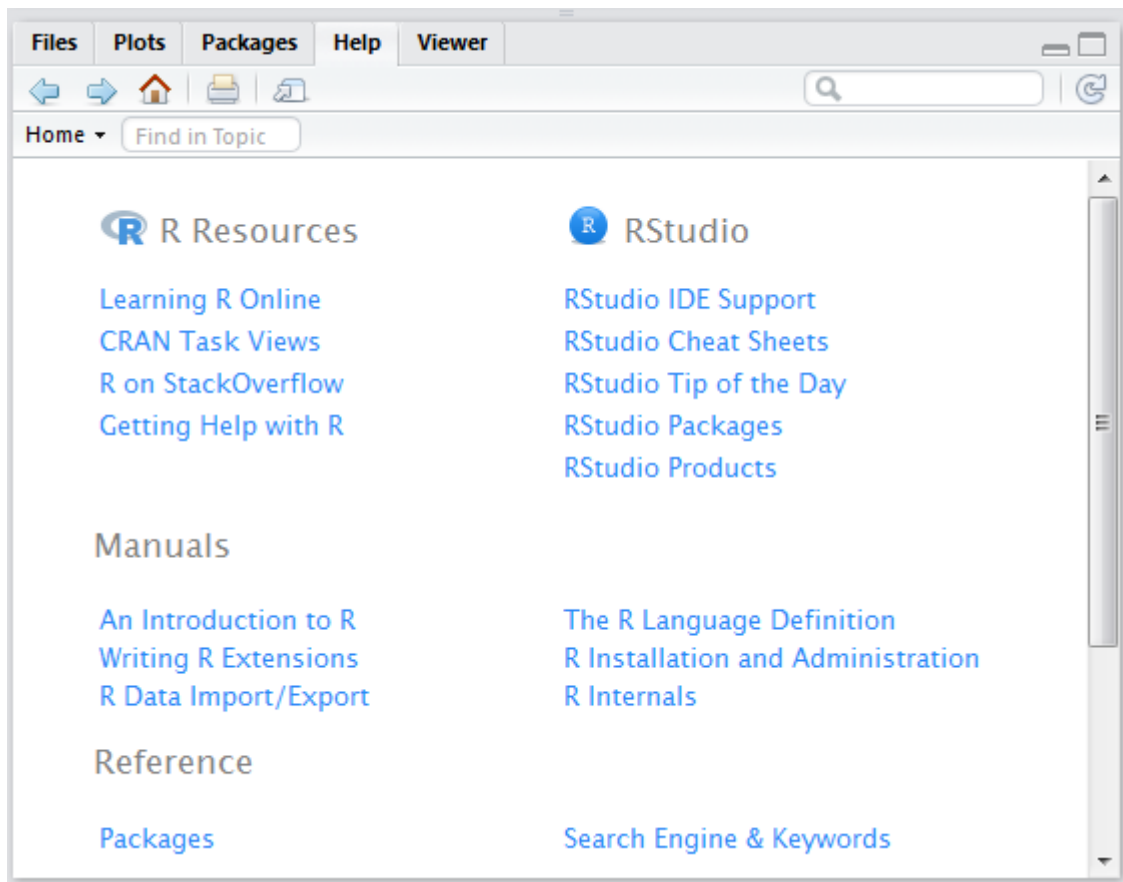


- Para poder usar cada una de las funciones de un paquete, necesitamos primero instalar el paquete y luego cargarlo en R.
 - Para instalar el paquete pulsa el botón instalar **Install** arriba a la izquierda. En la ventana que aparece, selecciona el repositorio de CRAN (*CRAN: Comprehensive R Archive Network*) en la pestaña de instalar desde. Escribe el nombre del paquete bajo el apartado **Packages** y pulsa instalar.
 - Por defecto, R instala los paquetes desde internet desde el repositorio de CRAN. No obstante, es posible instalar paquetes desde carpetas comprimidas (archivos zip) que se encuentren en tu ordenador.
 - Para cargar el paquete pulsa la casilla correspondiente al paquete que quieres cargar de la lista de paquetes que aparecen instalados en tu librería de R
- La instalación de paquetes también se puede llevar a cabo desde la consola de R:

```
install.packages('lattice')
# ahora cárgalo
library(lattice) #o
require(lattice) #mismo resultado
```

3. La ventana de **Ayuda**

Podemos buscar ayuda sobre cómo utilizar algunas funciones de los paquetes usando la barra de búsqueda en la ventana de Ayuda. También podemos buscar documentación de ayuda mediante los hiperenlaces azules.



- De nuevo también podemos buscar ayuda desde la consola, por ejemplo:

```
?read.table
help(read.table)

#o si no estamos seguro de lo que estamos buscando, podemos buscar
help.search("data input") #o
??"data input"           #Mismo resultado. Nota las comillas

#Para abrir ayuda online en RStudio
help.start()
```

Sintaxis básica

Uso como calculadora

R es esencialmente una calculadora. Podemos utilizar el programa para realizar cálculos simples como:

```
#un simple cálculo
2+5
```

```
[1] 7
```

```
#uso de paréntesis para determinar el orden de las operaciones
(2+5)*(6-3)
```

```
[1] 21
```

```
#respuesta diferente a la anterior  
2+5*6-3
```

```
[1] 29
```

```
#División con /  
10/(2+3)
```

```
[1] 2
```

```
#Elevar al cuadrado  
3^2
```

```
[1] 9
```

Habréis notado que el símbolo `#` está apareciendo continuamente. Este símbolo es importante en R ya que se usa para comentar código (escribir notas explicativas para ti o tus compañeros). Nada que siga al símbolo `#` será evaluado en R. Esto es importante a la hora de leer datos en R, ya que si tu base de datos contiene este símbolo es muy posible que se produzcan errores o que la base de datos no se lea de forma correcta.

```
#¿Qué resultado va a dar la siguiente operación?  
2+5##*20-3
```

Asimismo, R también tiene muchas funciones matemáticas y estadísticas. Por ejemplo:

```
sqrt(9) #Raíz cuadrada  
log(20) #Logaritmo en base e  
exp(20) #Exponencial  
abs(-10) #Valor absoluto  
round(3.1298376198712, digits=5) #Redondeo a cinco cifras decimales  
  
#R usa notación científica. Por ejemplo:  
1.2e3    # = 1200  
1.2e-2   # = 0.012
```

Más que ser una simple calculadora, R es también un lenguaje de programación muy potente. A menudo, una de las razones por las que usamos un lenguaje de programación en vez de una calculadora es para automatizar algunos de los procesos y evitar repeticiones innecesarias. En este curso, vamos a introducir este lenguaje de programación. No obstante, para aquellos que estén interesados en profundizar en su aprendizaje se recomiendan los siguientes [recursos](#)

Práctica

- Calcula el resultado de $8/2$
- Calcula el resultado de $(210*1290)/(\text{sqrt}(1367^3))$
- Encuentra la solución de las siguientes expresiones:
 - `sqrt(9)`
 - `log(20)`
 - `exp(20)`
 - `abs(-10)`
 - `round(3.1298376198712, digits=5)`
- Calcula el volumen de un tanque de agua de 4,28 metros de altura y con un radio de 1,19 metros.
 - **Recuerda:** la fórmula del volumen es $V = \pi \times \text{radio}^2 \times \text{altura}$.
 - **Nota:** π se pronuncia como pi y es un número especial que representa la relación entre la longitud de una circunferencia y su diámetro. El número π se escribe simplemente en R como `pi`.

Variables

Puede que queramos usar algunos de los resultados obtenidos con anterioridad en cálculos nuevos. En vez de repetir $(2+5)*(20-3)$ cada vez que lo necesitemos, podemos crear una *variable* y *asignarle* dicho resultado. La flecha de asignación (\leftarrow) se utiliza para asignar valores a variables. También es posible utilizar el símbolo $=$ pero la práctica estándar en R es usar la flecha \leftarrow que es lo que recomendamos y utilizaremos en este curso. Por ejemplo:

```
a <- 2 #a 'se le asigna' el valor 2
a
```

```
[1] 2
```

```
2*a
```

```
[1] 4
```

Habréis notado que R no muestra el resultado cuando se le asigna un valor a una variable. Para mostrar dicho resultado es necesario ejecutar dicha variable. Podemos sobrescribir variables asignándoles valores de otros cálculos u otras variables.

```
a <- 4
city <- "Adelaide" #Asignar una palabra o cadena ('string') de caracteres.

#y un poco más difícil
b <- 6
c <- -1
ans <- (-b+sqrt(b^2-4*a*c))/(2*a)
ans
```

```
[1] 0.1513878
```

Asignación de nombres y puntuación

R es muy particular en algunos aspectos. Es importante tener en cuenta que diferencia entre letras mayúsculas y minúsculas.

```
A <- 5
a
```

```
[1] 4
```

Scripts

Podemos guardar una lista de comandos que queremos volver a ejecutar en un archivo de texto llamado *script*. Estos archivos tienen una extensión `.R`. Podemos ejecutar todo el archivo de una vez o utilizar el comando `source()` con el mismo objetivo. También podemos ejecutarlo línea a línea utilizando el botón Run que aparece arriba a la derecha de la ventana del Script o con la combinación de teclas (Ctrl+Enter en Windows, Cmd+Enter en Mac).

Establecer el directorio de trabajo

Es un buen hábito guardar todos los archivos que estáis utilizando para un proyecto o un experimento en una sola carpeta y establecer esa carpeta como directorio de trabajo. Se puede comprobar o cambiar la carpeta en la que R está trabajando con los comandos `getwd()` o `setwd("<ruta del archivo>")`, respectivamente. Se puede realizar la misma operación manualmente en RStudio pulsando **Session > Set Working Directory > Choose directory...** en la barra superior de herramientas.

Práctica

- Comprueba tu directorio de trabajo, crea una nueva carpeta en tu ordenador para este curso y cambia el directorio de trabajo a dicha carpeta.
 - Asígnale los resultados de tus cálculos anteriores a variables. Puedes nombrar estas variables como consideres, aunque una de las pautas de la programación es asignar nombres que sean lógicos para dichas variables.
 - **Nota:** Puedes utilizar la flecha arriba para volver a comandos anteriores.
-

Vectores

Podemos asignar un conjunto de valores a un vector. De hecho, las variables de la sección anterior que contenían un solo valor son vectores de longitud 1. Se usa el operador `c()` para crear vectores.

```
# Creando vectores: c()
vec.num <- c(1,3,2,7,9,15)
vec.num
```

```
[1] 1 3 2 7 9 15
```

```
#Podemos crear vectores de texto
vec.txt <- c("a","b","c")
vec.txt
```

```
[1] "a" "b" "c"
```

```
vec.txt <- c("Wheat","Barley","Oats")
vec.txt
```

```
[1] "Wheat" "Barley" "Oats"
```

Existen vectores guardados en R:

```
LETTERS
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q"
[18] "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
letters
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
[18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

También podemos crear vectores usando el operador `rep()` (para repetir valores) y `seq()` (para crear secuencia de valores):

```
#Una secuencia del 1 al 21 con un salto de 3
vec.seq <- seq(1,21,3)
vec.seq
```

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

```
#Repetir el vector c(1,6) cinco veces
vec.rep <- rep(c(1,6), times=5)
vec.rep
```

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

```
#Repetir cada elemento del vector 5 veces
vec.rep <- rep(c(1,6), each=5)
vec.rep
```

```
[1] 1 1 1 1 1 6 6 6 6 6
#Los vectores pueden ser tan largos como uno quiera
rep(rep(c(1,6), each=5), times=10)

[1] 1 1 1 1 1 6 6 6 6 6 1 1 1 1 1 6 6 6 6 6 1 1 1 1 1 6 6 6 6 6 1 1 1 1 1
[36] 6 6 6 6 6 1 1 1 1 1 6 6 6 6 6 1 1 1 1 1 6 6 6 6 6 1 1 1 1 1 6 6 6 6 6
[71] 1 1 1 1 1 6 6 6 6 6 1 1 1 1 1 6 6 6 6 6 1 1 1 1 1 6 6 6 6 6
```

Obtención de los elementos de un vector

Los elementos de un vector se guardan en el vector de acuerdo a la posición que ocupan dentro del mismo. Por ejemplo:

```
vec.num <- c(5, 1, 8, 0, 1) #5 se encuentra en la primera posición,
                             #1 se encuentra en la segunda posición,
                             #8 se encuentra en la tercera posición,
                             #0 se encuentra en la cuarta posición y
                             #1 se encuentra en la quinta posición
```

Usamos los corchetes[x] después del nombre del vector para acceder al elemento del vector que se encuentra en la posición x

```
vec.num[1]
```

```
[1] 5
```

```
vec.num[2]
```

```
[1] 1
```

```
vec.num[c(1,3)] #el primer y el tercer elemento
```

```
[1] 5 8
```

```
vec.num[-2] #todos los elementos excepto el segundo
```

```
[1] 5 8 0 1
```

```
vec.num[1:4] #elementos del 1 al cuatro
```

```
[1] 5 1 8 0
```

```
vec.num[6] #no existe el sexto elemento, como resultado se obtiene NA
```

```
[1] NA
```

Podemos acceder al elemento que ocupa la posición x en el vector y asignarle un valor diferente mediante la flecha de asignación <-

```
#Asignamos el valor 4 al segundo elemento del vector vec.num
vec.num[2] <- 4
vec.num
```

```
[1] 5 4 8 0 1
```

R nos permite realizar operaciones matemáticas con vectores siempre y cuando dichos vectores tengan la misma longitud y sean del mismo tipo.

```
a <- c(1,1)
b <- c(2,1)
a+b
```

```
[1] 3 2
```

```
c <- c("one", "two")
a+c   #Falla porque no podemos añadir números y texto
c <- c(1, 2, 3)
b+c   #Funciona, pero es mejor evitarlo
```

Práctica

- Escribe un vector con la siguiente información: 1 en la primera posición, 3 en la segunda posición, 11 en la tercera posición; llama al vector **z**.
- Accede al tercer elemento del vector y reemplázalo con el valor 5.
- Calcula la media de **z** (suma todos los valores y divide el resultado por el número de valores).
- Reemplaza el primer elemento de **z** por **NA**.
- Crea un vector repitiendo el patrón de tus iniciales tres veces usando el operador **rep()**. Por ejemplo, John Adam Smith sería "J" "A" "S" "J" "A" "S" "J" "A" "S".
- Crea un vector con tres elementos y en cada elemento escribe el nombre de un país que te gustaría visitar. Accede al primer país. **Nota:** puedes incluir espacios e incluso acentos en el texto siempre y cuando los elementos del vector vengan entre comillas (" "). Elige un nombre (lógico) para dicho vector.
- Crea un vector con la secuencia del 1 al 5, llama al vector **r**. Crea otro vector con la secuencia del 2 al 10 con saltos de dos en dos y llama al vector **h**. Calcula el volumen de 5 tanques de distinto radio y altura usando la formula anterior del volumen ($V = \pi \times r^2 \times h$) y guárdalo con el nombre **V**.

Funciones

R, *grosso modo*, se compone de *objetos* y *funciones* que actúan sobre dichos objetos. No todas las funciones pueden actuar sobre todos los objetos. Esto puede sonar abstracto y para entenderlo mejor vamos a relacionarlo con ejemplos de nuestra vida diaria. Por ejemplo, muchos de nosotros tenemos un objeto llamado coche, y podemos aplicar varias funciones como como conducir, aparcarse, reparar, limpiar, etc. No obstante, no podemos aplicar funciones que no tienen sentido en el contexto de un coche como es nadar, respirar, volar. Volviendo a un ejemplo de R y como veíamos antes, no tiene sentido sumar vectores de texto.

R tiene muchas funciones guardadas, y podemos acceder a muchas más instalando paquetes nuevos. Ya hemos usado algunas de estas funciones como **sqrt()**, **round()** y **seq()**. Una función es simplemente una manera de abstraer un conjunto de operaciones realizadas por un ordenador de manera que no tenemos que realizar todos los pasos de forma repetitiva cada vez que queremos llevar a cabo el mismo conjunto de operaciones. En la práctica anterior, calculamos la media de **z**, que era 3. Esto fue relativamente fácil puesto que el vector **z** solo tenía tres elementos, así que fue sencillo realizar $(1+3+5)/3$ en R. Pero ¿qué ocurriría si tuviésemos un vector de 30 elementos, o 300 o 3000? Ciertamente, no nos gustaría escribir los elementos uno a uno. En este tipo de situaciones las funciones resultan bastante útiles.

```
z <- c(1,3,5)
#Calcula la media manualmente
(1+3+5)/3
```

```
[1] 3
```

```
#El mismo resultado
mean(z)
```

```
[1] 3
```

```
#Secuencia del 1 al 345
z <- 1:345
```

```
mean(z)
```

```
[1] 173
```

A continuación, podemos encontrar algunos ejemplos de funciones.

```
x <- c(1,3,2,7,9,15)
mean(x)    #media de x
var(x)     #varianza de x
max(x)     #máximo de x
min(x)     #mínimo de x
length(x)  #longitud de x (es decir, cuántos elementos tiene el vector)
sum(x)     #suma de todos los elementos de x
sort(x)    #ordena los elementos de x
class(x)   #clase de x (es decir, numérico, texto, lógico, etc.)
```

```
#Algunas funciones no proporcionan un resultado si el vector contiene NA
```

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

```
mean(b)
```

```
[1] NA
```

```
#Usamos na.rm=TRUE para ignorar los valores NA
```

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

```
[1] 2
```

Los elementos que se encuentran entre paréntesis en una función se llaman los *argumentos* de una función. Las funciones pueden tener muchos argumentos, y para separarlos utilizamos comas. Podemos pasar los argumentos a la función usando el nombre del argumento seguido del símbolo = (por ejemplo, `na.rm =`), o dichos argumentos pueden proporcionarse en el orden que la función espera recibirlos. Para conocer dicho orden, se puede consultar la documentación de ayuda para dicha función. La documentación de ayuda también proporciona información sobre el objeto que devuelve la función.

Muchas funciones están guardadas en *paquetes* (que son simplemente una colección de funciones), y se distribuyen vía Internet o en carpetas comprimidas. A continuación, proporcionamos un ejemplo de cómo instalar un paquete del repositorio de CRAN.

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

Práctica

Usando los vectores creados en la sección anterior:

- Calcular la media de `V`.
- Encontrar la longitud del vector `z`
- Encontrar la clase o tipo del vector donde se guardaron los países que querías visitar
- Encontrar la desviación estándar del vector `z`. (**Nota:** puedes buscar la función desviación estándar en el apartado de búsqueda de la ventana ayuda o mediante `??` y el comando que desees utilizar)
- Encuentra, instala y carga un paquete que te permita utilizar una función para leer archivos de Excel

Otro tipo de datos

Hemos introducido varios tipos de datos en R, principalmente valores numéricos (`numeric`), palabras de texto (`char`), y vectores (`vectors`). En esta sección, vamos a introducir otro tipo de datos.

Valores lógicos

Los valores lógicos pueden tomar el valor de verdadero (TRUE o T), falso (FALSE o F), o NA. Estos valores son útiles cuando realizamos comparaciones.

```
10>3
```

```
[1] TRUE
```

```
#x toma el valor de 5  
x <- 5  
#Podemos comprobar el valor de x  
#¿Toma x el valor 29?  
x == 29
```

```
[1] FALSE
```

Otro comando bastante útil para realizar comparaciones lógicas es `which()`. Este comando nos dice la localización de los elementos que dan como resultado TRUE en la comparación o comprobación. Por ejemplo, podríamos preguntarle a R “¿Qué valores de x son mayores que 5?” usando:

```
x <- round(runif(10, 1, 10), 0) # genera de forma aleatoria 10 enteros entre 1 y 10  
x <- sample(1:10, 10, replace = T) #Alternativa para generar 10 enteros entre 1 y 10  
x
```

```
[1] 6 9 1 10 1 9 6 5 2 7
```

```
which(x > 5) #Nos da la posición de los elementos que son mayores que 5
```

```
[1] 1 2 4 6 7 10
```

```
x[x>5] #Nos da el valor de los elementos que son mayores que 5
```

```
[1] 6 9 10 9 6 7
```

La tabla que se muestra a continuación nos explica las operaciones correspondientes a los operadores lógicos.

| Operadores lógicos | Operación |
|--------------------|-------------------------|
| <, <= | Menos que (o igual que) |
| >, >= | Mayor que (o igual que) |
| == | Igual que |
| ! | No |
| & | Y |
| | O |
| && and | No-vectorizado |

Matrices

Una matriz es una colección de elementos que se disponen en columnas y filas. En otras palabras, podemos pensar en una matriz como un conjunto de vectores filas puestos uno bajo el otro o un conjunto de vectores columnas puestos uno al lado de otro. De esta manera, las matrices tienen dimensión dos. Podemos acceder al vector que ocupa la fila x mediante `[x,]` o al vector que ocupa la columna y mediante `[,y]`.

```
m1 <- matrix(c(1,2,3,4,5,6), nrow = 3)  
m1
```

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



```
#Por defecto, R rellena las matrices verticalmente en columnas
#Usando byrow = T damos la orden a R de rellenarlas horizontalmente en filas.
m2 <- matrix(c(1:12), nrow = 3, byrow = T)
m2
```

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

```
#Podemos acceder al elemento de la segunda fila y primera columna
m2[2,1]
```

```
[1] 5
```

```
#Accede a la segunda columna. Nota que R no distingue entre vectores filas y vectores columnas.
#Para R, una vez accedemos a la fila o a la columna todo son vectores.
```

```
m2[,2]
```

```
[1] 2 6 10
```

```
#Obtenemos los elementos de la primera fila en la primera y tercera columna.
```

```
m2[1, c(1,3)]
```

```
[1] 1 3
```

Algunas de las funciones que usábamos con los vectores también pueden usarse con matrices:

```
mean(m2) #media
```

```
[1] 6.5
```

```
max(m2) #máximo
```

```
[1] 12
```

```
dim(m2) #Nos da la dimensión de la matriz (3 filas por 4 columnas en este caso)
```

```
[1] 3 4
```

```
m2[m2>3] #qué elementos de la matriz son mayores que 3
```

```
[1] 5 9 6 10 7 11 4 8 12
```

Arrays

Para generalizar los conceptos de vector y matriz, un vector es un *array* de dimensión 1 y una matriz es un *array* de dimensión 2. Podemos crear arrays con tantas dimensiones como queramos. Los arrays no son objetos muy comunes porque las listas y las bases de datos son más versátiles, es por ello que los mencionamos muy brevemente.

```
A <- array(letters[1:24], dim = c(4,2,3)) #3 4*2 matrices
A
```

```
, , 1
```

```
      [,1] [,2]
[1,] "a"  "e"
[2,] "b"  "f"
[3,] "c"  "g"
[4,] "d"  "h"
```

```
, , 2
```

```
      [,1] [,2]  
[1,] "i"  "m"  
[2,] "j"  "n"  
[3,] "k"  "o"  
[4,] "l"  "p"
```

```
, , 3
```

```
      [,1] [,2]  
[1,] "q"  "u"  
[2,] "r"  "v"  
[3,] "s"  "w"  
[4,] "t"  "x"
```

Listas

Una lista es una estructura de datos con múltiples componentes donde cada componente de la lista puede contener un tipo de datos diferente. Por ejemplo, podemos tener una lista de tres componentes y en la primera componente tener un vector, en la segunda componente una matriz y en la tercera componente una palabra. Las componentes pueden tener distintas longitudes, es por ello que este tipo de objetos resulta de gran versatilidad.

```
vec.lst <- list() #crea una lista de longitud cero  
vec.lst
```

```
list()
```

```
vec.lst <- vector(mode="list",length=2) # crea una lista de longitud dos (dos componentes)  
vec.lst
```

```
[[1]]  
NULL
```

```
[[2]]  
NULL
```

```
names(vec.lst) <- c("a","b")  
vec.lst
```

```
$a  
NULL
```

```
$b  
NULL
```

```
vec.lst$a <- c(1,2)  
vec.lst
```

```
$a  
[1] 1 2
```

```
$b  
NULL
```

```

vec.lst <- list(x=runif(10),y=letters[1:15])
vec.lst

$x
[1] 0.29328365 0.06027509 0.35937782 0.18541754 0.95515606 0.82383518
[7] 0.56652808 0.30265074 0.93319464 0.38848837

$y
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o"
#longitud de la lista
length(vec.lst)

[1] 2
names(vec.lst)

[1] "x" "y"
lapply(vec.lst,length) #longitud de cada una de las componentes de la lista

$x
[1] 10

$y
[1] 15
#Accede a los elementos de la lista
vec.lst[[2]] #Accede a la segunda componente de la lista

[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o"
vec.lst[[2]][1] #Accede al primer elemento de la segunda componente de la lista

[1] "a"
#Elementos de una lista
vec.lst[1] #una lista que contiene solo la primera componente de la lista original vec.lst

$x
[1] 0.29328365 0.06027509 0.35937782 0.18541754 0.95515606 0.82383518
[7] 0.56652808 0.30265074 0.93319464 0.38848837
attributes(vec.lst[1])

$names
[1] "x"
class(vec.lst[1])

[1] "list"
class(vec.lst[[1]])

[1] "numeric"
class(vec.lst[[2]])

[1] "character"
#Añadiendo componentes a una lista
vec.lst$z <- c(1,2,3,4)

```

```
vec.lst
```

```
$x
```

```
[1] 0.29328365 0.06027509 0.35937782 0.18541754 0.95515606 0.82383518  
[7] 0.56652808 0.30265074 0.93319464 0.38848837
```

```
$y
```

```
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o"
```

```
$z
```

```
[1] 1 2 3 4
```

```
vec.lst[[4]] <- seq(1,20)
```

```
vec.lst
```

```
$x
```

```
[1] 0.29328365 0.06027509 0.35937782 0.18541754 0.95515606 0.82383518  
[7] 0.56652808 0.30265074 0.93319464 0.38848837
```

```
$y
```

```
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o"
```

```
$z
```

```
[1] 1 2 3 4
```

```
[[4]]
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

```
names(vec.lst)[4] <- 'new'
```

```
vec.lst
```

```
$x
```

```
[1] 0.29328365 0.06027509 0.35937782 0.18541754 0.95515606 0.82383518  
[7] 0.56652808 0.30265074 0.93319464 0.38848837
```

```
$y
```

```
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o"
```

```
$z
```

```
[1] 1 2 3 4
```

```
$new
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

```
vec.lst$mat <- matrix(rnorm(60),2,3)
```

```
vec.lst
```

```
$x
```

```
[1] 0.29328365 0.06027509 0.35937782 0.18541754 0.95515606 0.82383518  
[7] 0.56652808 0.30265074 0.93319464 0.38848837
```

```
$y
```

```
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o"
```

```
$z
```

```
[1] 1 2 3 4
```

```

$new
[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20

$mat
      [,1]      [,2]      [,3]
[1,]  1.6821041 0.5090884 0.5758447
[2,] -0.6236012 0.2462884 0.3222023

```

A continuación, mostramos una imagen para ilustrar los distintos tipos de objetos que hemos estudiado junto con las bases de datos que estudiaremos en la sección siguiente.

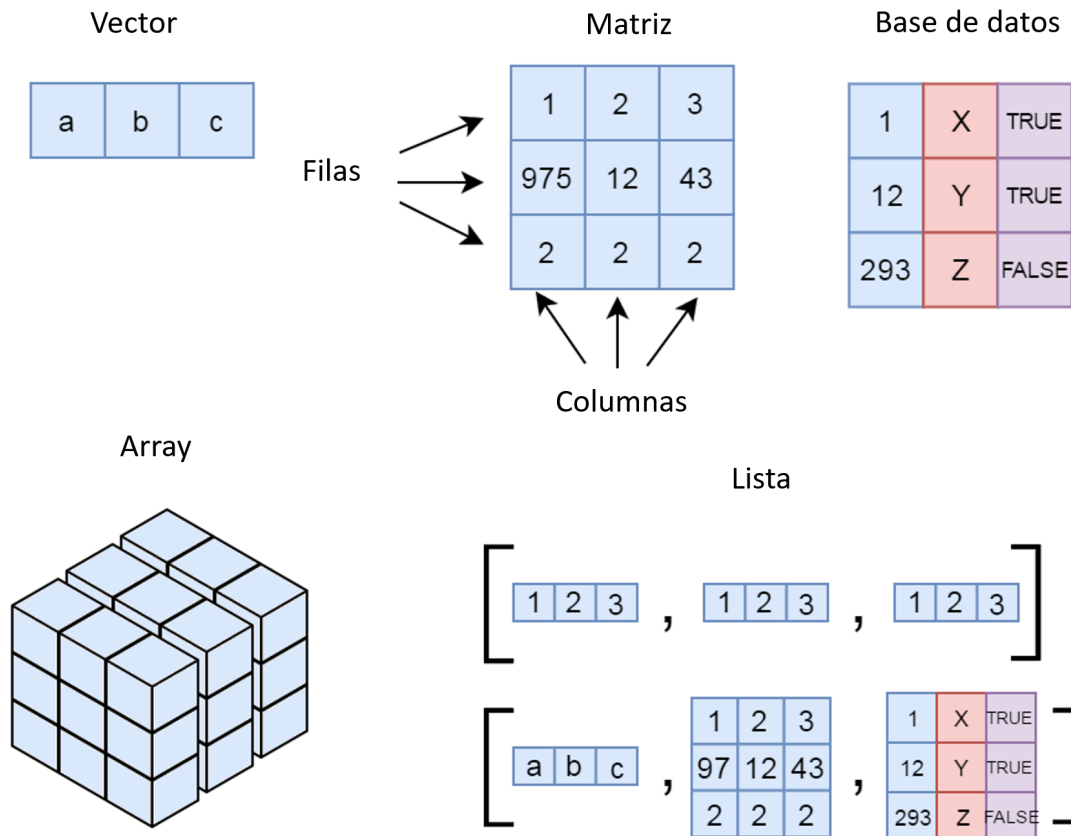


Figura 5: Tipos de datos

Práctica

- Usando el vector `V` de la sección anterior, comprueba si el último valor de `V` es mayor que 500
- Usa el comando `which()` para encontrar las posiciones de los volúmenes que son mayores que 100 y encuentra qué valores de volúmenes son.
- Crea una matriz de 3 filas y 2 columnas donde la primera fila contiene los números 2, 2, la segunda fila 4,5 y la tercera fila 8,9 y llámala `m2`
- Muestra todos los elementos de la primera fila, y todos los elementos de la segunda columna
- Crea un vector `z` con la tercera fila de la matriz, y de este vector extrae el elemento que es mayor que 8
- Crea una lista con dos objetos. En la primera componente de la lista guarda un vector y en la segunda una matriz. Puede ser un vector o una matriz que hayas creado anteriormente. (**Nota:** Si has olvidado qué has creado, puedes usar `ls()` para recordar todos los objetos que has creado hasta el momento)

- Accede al primer elemento del vector que has guardado en la primera componente de la lista.
-

Gestión de datos

En esta sección hablaremos sobre la gestión de datos con la intención de facilitar su análisis en R. Los consejos que daremos tienen el objetivo de facilitar el análisis de los datos que se han recogido. Existen otras fuentes de recursos que hablan sobre este tema y que pueden consultarse en la última sección de este documento sobre [más recursos](#) para reforzar el autoaprendizaje.

Lo más importante que hay que tener en cuenta a la hora de introducir datos en Excel es **¡ser consistente!** Veremos qué significa esto en las siguientes secciones.

Qué no hacer

Estos datos de hospitales son un ejemplo de lo que no hacer. Descárgate el archivo en <http://bit.ly/2q5uFR2>.

Ahora hemos visto un ejemplo de qué *no* hacer. ¿Cuáles son las buenas prácticas que deberíamos tener en cuenta?

Guardando los datos

- Guarda los datos en un archivo grande ya que es más fácil de tratar que muchos pequeños.
- Guarda siempre una copia de tus datos originales *antes* de hacer cualquier cambio en los mismos.
 - Realiza los cambios en R, y guarda el archivo script (este archivo contendrá una lista ordenada de todos los pasos que has seguido para llevar a cabo la limpieza de los datos).
- Guarda tus datos en formatos no patentados siempre que sea posible.
 - Usa extensiones .csv o .txt antes que .xls/.xlsx, Microsoft accede a archivos de bases de datos en estos formatos. Si un software en particular no existe a los 1/5/20 años, todavía quieres que te sea posible acceder a tus datos.
- Incluye solo los *datos* en el archivo Excel. No incluyas gráficos, tablas etc.
 - Evita columnas con comentarios o notas tanto como sea posible.
 - Este tipo de notas deben ir en una hoja Excel separada donde se hagan este tipo de descripciones o en un archivo por separado.
- Ten un archivo que explique y describa los datos
 - ¡¡No infravalores tu habilidad para olvidar lo que significan los datos o anotaciones de la muestras!! Por ejemplo, ¡¡aquellas parcelas que destruyeron las plagas!!
- Guarda los datos en una carpeta que sea independiente y explicativa de lo que guarda.
 - Si por alguna razón no te pudieran contactar, ¿podría alguien acceder a los datos y saber que significan o debería esta persona indagar en tus papeles, e-mails y archivos para poder saber dónde se encuentran los datos?
- **¡Haz una copia de seguridad!**

Pensando en filas y columnas

Para nuestros objetivos (recogida y el análisis de datos), generalmente queremos usar un formato donde las columnas contienen variables o descriptores, y las filas contienen observaciones individuales. Una variable contiene los valores que se han medido de un mismo atributo (como puede ser la altura, la temperatura, la duración) en todos los elementos de la muestra. Una observación contiene todos los valores que se han medido del mismo elemento de la muestra (como pueden ser una persona, una planta o una propiedad) de todos los atributos que se han medido en el experimento.

Queremos evitar ir añadiendo columnas tanto como sea posible después de haber tomado los datos. De manera que si por alguna razón hubiese que tomar otra muestra, queremos que nuestros datos hayan sido recogidos de manera que sea fácil añadir los valores de esta nueva muestra a la base de datos. Por ejemplo,

añadiendo simplemente una fila al final de nuestros datos y rellenando todos los valores de los distintos atributos medidos en el experimento para esa muestra.

Poniendo nombre a las columnas

Poner nombre a las columnas puede resultar difícil. Los nombres deben tener un significado lógico y ser concisos. A ser posible no deberían incluir símbolos de puntuación o espacios. Por ejemplo:

- `max_temp_celsius` en vez de `max temp celsius` (incluye espacios)
- `airport_faa_code` no `airport/faa code` (incluye caracteres especiales)

R trabaja mejor con bases de datos basados en filas (también llamados formato “largo”). Es decir, aquellas bases de datos que tienen más filas que columnas donde las columnas hacen referencia a las variables y las observaciones a las filas. R es una herramienta muy potente para gestionar y manipular datos, y podemos cambiar fácilmente de un formato “largo” a uno “ancho”.

A veces tenemos muchas variables o columnas, que hacen que la base de datos sea más amplia de lo que debería. En este caso, puede que sea mejor condensar las columnas en una sola columna y tener una medida que condense la información. Aquí tenemos un ejemplo de algunos datos de temperaturas que se tomaron diariamente de una estación de México en 2010.

| | id | year | month | element | day1 | day2 | day3 | day4 | day5 | day6 | day7 | day8 |
|----|---------|------|-------|---------|------|------|------|------|------|------|------|------|
| 1 | MX17004 | 2010 | 1 | tmax | NA | NA | NA | NA | NA | NA | NA | NA |
| 2 | MX17004 | 2010 | 1 | tmin | NA | NA | NA | NA | NA | NA | NA | NA |
| 3 | MX17004 | 2010 | 2 | tmax | NA | 27.3 | 24.1 | NA | NA | NA | NA | NA |
| 4 | MX17004 | 2010 | 2 | tmin | NA | 14.4 | 14.4 | NA | NA | NA | NA | NA |
| 5 | MX17004 | 2010 | 3 | tmax | NA | NA | NA | NA | 32.1 | NA | NA | NA |
| 6 | MX17004 | 2010 | 3 | tmin | NA | NA | NA | NA | 14.2 | NA | NA | NA |
| 7 | MX17004 | 2010 | 4 | tmax | NA | NA | NA | NA | NA | NA | NA | NA |
| 8 | MX17004 | 2010 | 4 | tmin | NA | NA | NA | NA | NA | NA | NA | NA |
| 9 | MX17004 | 2010 | 5 | tmax | NA | NA | NA | NA | NA | NA | NA | NA |
| 10 | MX17004 | 2010 | 5 | tmin | NA | NA | NA | NA | NA | NA | NA | NA |

Estos datos tienen las variables (temperatura máxima y mínima) y las observaciones (días) en sentido contrario. Así es como debería estar:

| | id | year | month | day | tmax | tmin |
|----|---------|------|-------|-----|------|------|
| 1 | MX17004 | 2010 | 1 | 30 | 27.8 | 14.5 |
| 2 | MX17004 | 2010 | 2 | 2 | 27.3 | 14.4 |
| 3 | MX17004 | 2010 | 2 | 3 | 24.1 | 14.4 |
| 4 | MX17004 | 2010 | 2 | 11 | 29.7 | 13.4 |
| 5 | MX17004 | 2010 | 2 | 23 | 29.9 | 10.7 |
| 6 | MX17004 | 2010 | 3 | 5 | 32.1 | 14.2 |
| 7 | MX17004 | 2010 | 3 | 10 | 34.5 | 16.8 |
| 8 | MX17004 | 2010 | 3 | 16 | 31.1 | 17.6 |
| 9 | MX17004 | 2010 | 4 | 27 | 36.3 | 16.7 |
| 10 | MX17004 | 2010 | 5 | 27 | 33.2 | 18.2 |

Ahora, si quisiésemos anotar el tiempo de mañana, simplemente tendríamos que añadir otra fila al final y rellenar los detalles en esa fila.

Formateando los valores

La forma en la que Excel formatea los datos (especialmente números y fechas) es a menudo un efecto visual. Es decir, lo muestra de la forma que nosotros queremos, pero lo guarda de forma distinta para sí mismo. Esto puede llevar a errores cuando leemos los datos en R, ya que R sí que lee este tipo de datos como Excel lo ha guardado para sí mismo. A continuación, mostramos algunas de los problemas de formato de datos más comunes.

Fechas

Es necesario que entendamos que la manera en que Excel “ve” las fechas es diferente a la manera en que lo hace R. Excel guarda las fechas como números: el número de días desde el 1 de enero del 1900. R trata las fechas como un objeto fecha (**Date**) que tiene un significado especial para R. No obstante, esto no resulta un problema. R puede tratar con fechas de Excel *siempre y cuando sepa qué esperar*. Normalmente, esto se soluciona guardando nuestros datos en un archivo con extensión csv así como existen otras herramientas de R para gestionar y manipular fechas.

Números

Así como ocurre con las fechas, el formato de número es un efecto visual de Excel. Cuando guardemos archivos para R, es conveniente eliminar todo excepto la coma decimal (incluyendo signos como el dólar, símbolos de porcentaje, etc.) ya que esto hará pensar a R que los números son texto.

Colores, subrayado, negrita y celdas combinadas

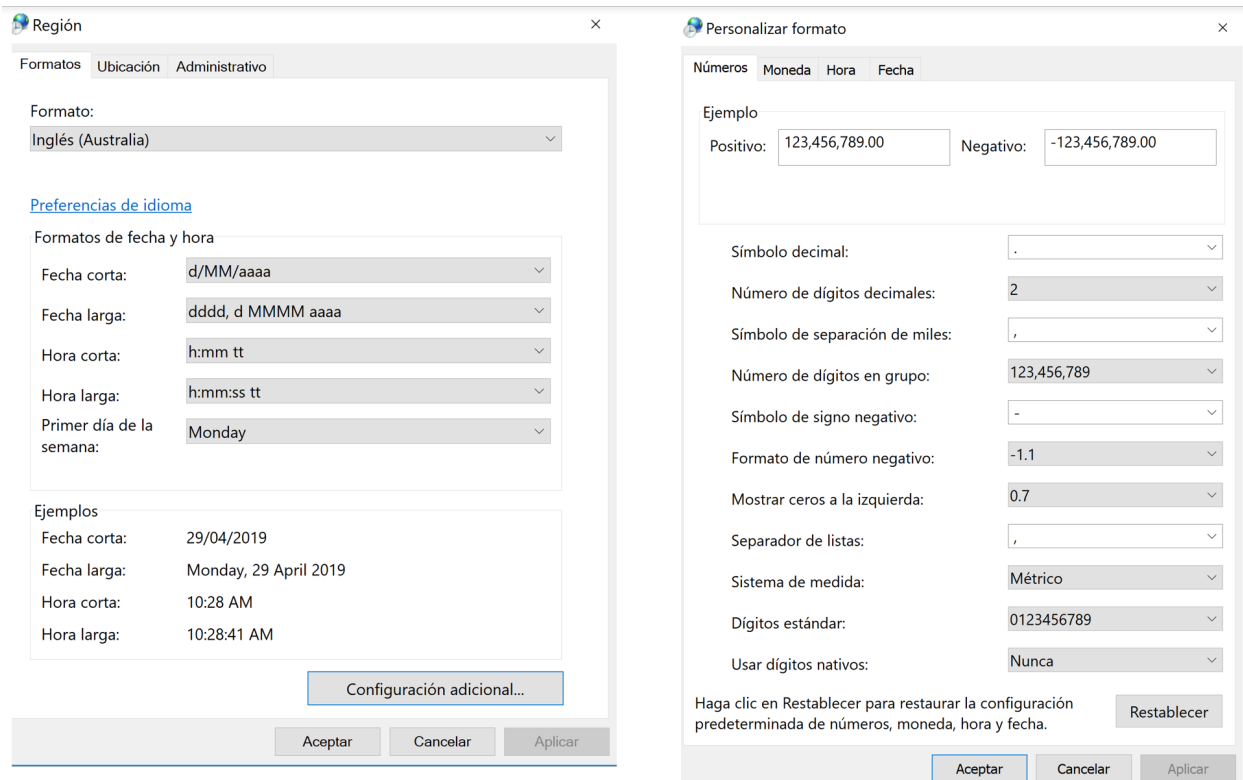
Ninguno de los efectos visuales como los colores, subrayados, negritas o celdas combinadas van a aparecer en R. Usar esto en Excel no suele crear un problema para R, pero ten en cuenta que dichos efectos no van a ser leídos por R.

Archivos CSV

CSV es un formato de texto basado en separar los valores por comas. Guardar un archivo de Excel (.xlsx) como un csv guardará los valores como se han mostrado en vez de como están guardados en Excel para sí mismo. Esto nos da una mejor idea de cómo R va a leer dicho archivo. Además, tienen la ventaja de que ocupan menos espacio que los archivos xls (aunque xlsx son normalmente más pequeños de nuevo). CSV es además un formato no patentado, por tanto, existe menos riesgo de no poder abrirlos en el futuro.

En España utilizamos la coma para delimitar la posición decimal de los números y los puntos para la separación de miles. En los países anglosajones es justo al revés. Esto hace que podamos tener problemas con los archivos CSV donde la coma ha sido utilizada para separar las celdas ya que esta es nuestro símbolo para delimitar la posición decimal y por tanto puede que al abrir un archivo CSV el resultado no sea el esperado. Las celdas en España se separan por ;. Las funciones `read.csv` y `write.csv` se utilizan para leer y escribir archivos separados por comas y `read.csv2` y `write.csv2` se utilizan para archivos separados por puntos y coma.

Opcionalmente se puede adoptar la notación anglosajona y cambiar el formato de los números temporalmente en tu ordenador. Para ello vamos al panel de control y bajo el apartado Reloj y Región podemos cambiar los formatos de fecha, hora y número. Seleccionamos la opción inglés (Australia) en el apartado formato y posteriormente hacemos clic en configuración adicional. Una vez en esta ventana seleccionamos el punto como símbolo decimal y la coma como símbolo de separación de miles. Esto debería ser suficiente para evitar problemas al abrir archivos CSV.



Datos no disponibles

¿Qué deberíamos hacer si en nuestra base de datos faltan algunas observaciones?

Vamos a echar un vistazo al ejemplo <http://bit.ly/2qXkwH4>

Si faltan datos en nuestro experimento, la mejor alternativa es dejar la celda en blanco o usar la palabra NA. Lo más importante que tenemos que recordar es: ¡ser consistente! Si empezamos a guardar los datos con un sistema, mantendremos este sistema en todo el archivo de datos.

Base de datos

Aprender a cómo gestionar vuestros datos, introducirlos en vuestro ordenador y leerlos es uno de los aspectos más importantes que necesitaréis dominar. Cuando se leen datos en R desde una hoja de Excel o similar, R normalmente lo interpreta como un objeto llamado base de datos (data frame). El objeto data frame tiene estructura de tabla con filas y columnas. Las filas contienen las distintas observaciones del estudio y las columnas contienen los valores de las distintas variables. Uno de los puntos más importantes que debemos saber sobre las bases de datos es que las columnas pueden contener distintos tipos de datos. Por ejemplo, la primera columna puede contener números, la segunda texto y la tercera columna puede contener datos de tipo lógico. No obstante, todas las columnas deben tener la misma longitud. Volveremos a este tema después.

```
a <- 1:4
b <- c("Dog", "Observation 2", "Parachute", "Singapore")
c <- c(TRUE, TRUE, FALSE, NA)
mydf <- data.frame(a, b, c)
mydf
```

| a | b | c |
|---|-----|------|
| 1 | Dog | TRUE |

```
2 2 Observation 2 TRUE
3 3 Parachute FALSE
4 4 Singapore NA
```

Recordad que los nombres de las columnas deberían tener un significado lógico y ser concisos. Además, *pueden* contener espacios, pero es mejor si no lo hacen. Nunca incluyáis el símbolo # en ningún sitio de vuestra base de datos ya que, como hemos comentado anteriormente, este es el símbolo que R utiliza para los comentarios y por tanto causará errores en la lectura de los datos.

```
colnames(mydf)
```

```
[1] "a" "b" "c"
```

```
colnames(mydf) <- c("Numbers", "Words", "Boolean")
mydf
```

| | Numbers | Words | Boolean |
|---|---------|---------------|---------|
| 1 | 1 | Dog | TRUE |
| 2 | 2 | Observation 2 | TRUE |
| 3 | 3 | Parachute | FALSE |
| 4 | 4 | Singapore | NA |

R tiene algunas bases de datos de ejemplo.

```
#Utiliza data() para ver todas las bases de datos que existen en R
head(iris) #Muestra las primeras 6 filas
```

| | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|---|--------------|-------------|--------------|-------------|---------|
| 1 | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 2 | 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 3 | 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 4 | 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 5 | 5.0 | 3.6 | 1.4 | 0.2 | setosa |
| 6 | 5.4 | 3.9 | 1.7 | 0.4 | setosa |

```
#Nos muestra un resumen de los datos
```

```
summary(iris)
```

| | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width |
|----------|--------------|---------------|---------------|---------------|
| Min. | :4.300 | Min. :2.000 | Min. :1.000 | Min. :0.100 |
| 1st Qu.: | 5.100 | 1st Qu.:2.800 | 1st Qu.:1.600 | 1st Qu.:0.300 |
| Median | :5.800 | Median :3.000 | Median :4.350 | Median :1.300 |
| Mean | :5.843 | Mean :3.057 | Mean :3.758 | Mean :1.199 |
| 3rd Qu.: | 6.400 | 3rd Qu.:3.300 | 3rd Qu.:5.100 | 3rd Qu.:1.800 |
| Max. | :7.900 | Max. :4.400 | Max. :6.900 | Max. :2.500 |

Species

```
setosa :50
versicolor:50
virginica :50
```

Acceso a elementos de una base de datos

Podemos acceder a los elementos de una base de datos de la misma forma que lo hacíamos con las matrices (usando [x,y]), pero las bases de datos también tienen otra propiedad muy útil. Podemos acceder a las columnas de una base de datos utilizando el símbolo \$ y el nombre de la columna. Por ejemplo:

```
mydf[,1] #Nos muestra todos los elementos de la columna 1
```

```
[1] 1 2 3 4
```

```
mydf$Words #Nos muestra los elementos de la columna Words
```

```
[1] Dog          Observation 2 Parachute    Singapore  
Levels: Dog Observation 2 Parachute Singapore
```

Notad como el último comando nos muestra los niveles (**Levels**) de la columna. Esto es porque la columna es un *factor*. Los factores son vectores de texto pero con una estructura. R reconoce que existen distintas categorías y todos los datos con la palabra “Dog” pertenecen a la misma categoría. Si esta columna fuera tratada como un vector de texto, cada uno de estos elementos se tratarían de forma individual y la misma palabra no se reconocería como la misma categoría.

```
#La función str (abreviación de estructura) nos proporciona toda la información  
#sobre las columnas de nuestra base de datos.  
str(iris) #Nota "Factor" en la última línea
```

```
'data.frame':  150 obs. of  5 variables:  
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...  
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...  
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...  
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...  
 $ Species      : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
```

```
iris[c(1:3, 148:150),] #Principio y final de la base de datos
```

| | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|-----|--------------|-------------|--------------|-------------|-----------|
| 1 | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 2 | 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 3 | 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 148 | 6.5 | 3.0 | 5.2 | 2.0 | virginica |
| 149 | 6.2 | 3.4 | 5.4 | 2.3 | virginica |
| 150 | 5.9 | 3.0 | 5.1 | 1.8 | virginica |

Los factores tienen niveles (“levels”). Podemos comprobar cuáles son estos niveles mediante el comando `levels()`. Podemos cambiar el nombre de los niveles accediendo a ellos o mediante alguna comprobación lógica.

```
levels(iris$Species)
```

```
[1] "setosa"      "versicolor" "virginica"
```

```
levels(iris$Species)[1] <- "spuria" #Reemplaza el primer nivel  
levels(iris$Species)
```

```
[1] "spuria"      "versicolor" "virginica"
```

```
#Encuentra los niveles que hemos cambiado anteriormente y reemplaza su nombre con el nombre original  
levels(iris$Species)[levels(iris$Species)=="spuria"] <- "setosa"  
levels(iris$Species)
```

```
[1] "setosa"      "versicolor" "virginica"
```

Leyendo los datos

Podemos leer los datos desde la ventana “[Environment](#)” pulsando Import Dataset. No obstante, se suele utilizar las funciones `read.table` y `read.csv`. Podemos leer los datos desde archivos locales guardados en el

ordenador, o incluso de archivos guardados en la Web. Aquí mostramos algunos ejemplos de cómo leer datos.

```
#Lee tu propia base de datos
#Encabezado (header) indica si quieres leer los nombres de las columnas.
#sep=', ' valores de cada línea del archivo están separados por este carácter.
fiber <- read.table(file = "Data/Energydigestability1.csv", header=T, sep=',')
str(fiber)

'data.frame':  30 obs. of  5 variables:
 $ PlantGroup  : Factor w/ 5 levels "apple","carrot",...: 5 5 5 5 5 5 1 1 1 1 ...
 $ EnergyDigest: num  91.5 91.5 90.5 89 87 86.5 70 78.5 78 74.5 ...
 $ CrudeFiber   : num   2 1.8 1.8 1.8 1.8 1.8 5.7 5.1 5.5 5.9 ...
 $ AverageCF    : num   1.87 1.87 1.87 1.87 1.87 1.87 5.74 5.74 5.74 5.74 ...
 $ Date         : Factor w/ 30 levels "1/07/2015","10/07/2015",...: 8 9 10 11 13 14 15 16 17 18 ...

#Mismo resultado que antes, pero no hay necesidad de argumentos extras
fiber <- read.csv(file = "Data/Energydigestability1.csv")
str(fiber)

'data.frame':  30 obs. of  5 variables:
 $ PlantGroup  : Factor w/ 5 levels "apple","carrot",...: 5 5 5 5 5 5 1 1 1 1 ...
 $ EnergyDigest: num  91.5 91.5 90.5 89 87 86.5 70 78.5 78 74.5 ...
 $ CrudeFiber   : num   2 1.8 1.8 1.8 1.8 1.8 5.7 5.1 5.5 5.9 ...
 $ AverageCF    : num   1.87 1.87 1.87 1.87 1.87 1.87 5.74 5.74 5.74 5.74 ...
 $ Date         : Factor w/ 30 levels "1/07/2015","10/07/2015",...: 8 9 10 11 13 14 15 16 17 18 ...

# Se pueden leer datos directamente de un archivo.xlsx, pero se necesita Java
# library(xlsx)
# fiber <- read.xlsx("Data/Energydigestability1.xlsx")
# str(fiber)
```

Una vez leídas las bases de datos podemos hacer cambios sobre ellas o analizarlas. También podemos volverlas a guardar en nuestro ordenador.

```
write.table(x = fiber, file = "Data/Energydigestability1.csv")

#El mismo resultado que la línea anterior
write.csv(x = fiber, file = "Data/Energydigestability1.csv")
```

Práctica

Vamos a hacer algunos ejercicios relacionados con las bases de datos:

- Descárgate el [archivo: http://bit.ly/2qSa547](http://bit.ly/2qSa547)
- Lee el archivo y guárdalo en R con el nombre `temp`
- Investiga la estructura de esta base de datos
- Si tienes un factor en la base de datos, cambia el nombre de uno de los niveles
- Accede a la tercera columna de tus datos
- Accede al quinto elemento de la primera columna
- Escribe la base de datos `temp` en un archivo de tu ordenador que se llame `temp.csv`

Funciones más avanzadas

Más tipos de datos

Hemos hablado de crear matrices, arrays y listas. En la práctica raramente creamos estos objetos, sino que es más común encontrárnoslos como el objeto que nos devuelve una función. A continuación, vamos a ver algunos ejemplos.

Matrices

Uno de los ejemplos de cuando una matriz es devuelta como salida en una función es la matriz de correlación. Por ejemplo, podemos utilizar la función `cor()` si tenemos múltiples columnas en una base de datos y queremos estimar la correlación entre columnas. La función `cor()` nos devuelve una matriz. Considera la base de datos `iris`.

```
cors <- cor(iris[, -5]) #Elimina la columna con el nombre de las especies
cors
```

```
          Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000   -0.1175698    0.8717538    0.8179411
Sepal.Width     -0.1175698    1.0000000   -0.4284401   -0.3661259
Petal.Length     0.8717538   -0.4284401    1.0000000    0.9628654
Petal.Width      0.8179411   -0.3661259    0.9628654    1.0000000
```

```
class(cors)
```

```
[1] "matrix"
```

Lista

Ahora que hemos calculado la matriz de correlación, observamos que la longitud de los pétalos (`Petal.Length`) tiene una correlación muy alta con la amplitud de los mismos (`Petal.Width`). Podríamos usar esta información para establecer un modelo de regresión y poder predecir la longitud de un pétalo dado una amplitud específica del mismo. Los modelos de regresión son creados en R con la función `lm()`.

```
model <- lm(formula=Petal.Length~Petal.Width, data=iris)
```

Podemos establecer modelos entre variables utilizando la notación (`~`). En este caso, la variable dependiente y se encuentra a la izquierda de la tilde (`~`), y la variable independiente x a la derecha. En el ejemplo anterior, estamos determinando la relación entre la variable dependiente `Petal.Length` y la variable independiente `Petal.Width`. Vamos a ver qué produce la función `lm()`.

```
model
```

```
Call:
```

```
lm(formula = Petal.Length ~ Petal.Width, data = iris)
```

```
Coefficients:
```

```
(Intercept)  Petal.Width
      1.084         2.230
```

```
str(model) #Muestra la estructura del objeto
```

```
List of 12
```

```
$ coefficients : Named num [1:2] 1.08 2.23
 ..- attr(*, "names")= chr [1:2] "(Intercept)" "Petal.Width"
$ residuals    : Named num [1:150] -0.1295 -0.1295 -0.2295 -0.0295 -0.1295 ...
 ..- attr(*, "names")= chr [1:150] "1" "2" "3" "4" ...
$ effects      : Named num [1:150] -46.02591 20.74803 -0.20895 -0.00895 -0.10895 ...
```

```

..- attr(*, "names")= chr [1:150] "(Intercept)" "Petal.Width" "" "" ...
$ rank          : int 2
$ fitted.values: Named num [1:150] 1.53 1.53 1.53 1.53 1.53 ...
..- attr(*, "names")= chr [1:150] "1" "2" "3" "4" ...
$ assign        : int [1:2] 0 1
$ qr            :List of 5
..$ qr          : num [1:150, 1:2] -12.2474 0.0816 0.0816 0.0816 0.0816 ...
.. ..- attr(*, "dimnames")=List of 2
.. .. ..$ : chr [1:150] "1" "2" "3" "4" ...
.. .. ..$ : chr [1:2] "(Intercept)" "Petal.Width"
.. ..- attr(*, "assign")= int [1:2] 0 1
..$ qraux: num [1:2] 1.08 1.1
..$ pivot: int [1:2] 1 2
..$ tol   : num 1e-07
..$ rank  : int 2
..- attr(*, "class")= chr "qr"
$ df.residual : int 148
$ xlevels      : Named list()
$ call         : language lm(formula = Petal.Length ~ Petal.Width, data = iris)
$ terms        :Classes 'terms', 'formula' language Petal.Length ~ Petal.Width
.. ..- attr(*, "variables")= language list(Petal.Length, Petal.Width)
.. ..- attr(*, "factors")= int [1:2, 1] 0 1
.. .. ..- attr(*, "dimnames")=List of 2
.. .. .. ..$ : chr [1:2] "Petal.Length" "Petal.Width"
.. .. .. ..$ : chr "Petal.Width"
.. ..- attr(*, "term.labels")= chr "Petal.Width"
.. ..- attr(*, "order")= int 1
.. ..- attr(*, "intercept")= int 1
.. ..- attr(*, "response")= int 1
.. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
.. ..- attr(*, "predvars")= language list(Petal.Length, Petal.Width)
.. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
.. .. ..- attr(*, "names")= chr [1:2] "Petal.Length" "Petal.Width"
$ model        :'data.frame': 150 obs. of 2 variables:
..$ Petal.Length: num [1:150] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
..$ Petal.Width : num [1:150] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
..- attr(*, "terms")=Classes 'terms', 'formula' language Petal.Length ~ Petal.Width
.. .. ..- attr(*, "variables")= language list(Petal.Length, Petal.Width)
.. .. ..- attr(*, "factors")= int [1:2, 1] 0 1
.. .. .. ..- attr(*, "dimnames")=List of 2
.. .. .. .. ..$ : chr [1:2] "Petal.Length" "Petal.Width"
.. .. .. .. ..$ : chr "Petal.Width"
.. .. ..- attr(*, "term.labels")= chr "Petal.Width"
.. .. ..- attr(*, "order")= int 1
.. .. ..- attr(*, "intercept")= int 1
.. .. ..- attr(*, "response")= int 1
.. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
.. .. ..- attr(*, "predvars")= language list(Petal.Length, Petal.Width)
.. .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
.. .. .. ..- attr(*, "names")= chr [1:2] "Petal.Length" "Petal.Width"
- attr(*, "class")= chr "lm"

```

```
names(model) #Los nombres de las variables en la lista que devuelve la función
```

```
[1] "coefficients" "residuals" "effects" "rank"
```

```
[5] "fitted.values" "assign"          "qr"              "df.residual"
[9] "xlevels"       "call"            "terms"           "model"
```

```
#Información útil
summary(model)
```

Call:

```
lm(formula = Petal.Length ~ Petal.Width, data = iris)
```

Residuals:

```
      Min       1Q   Median       3Q      Max
-1.33542 -0.30347 -0.02955  0.25776  1.39453
```

Coefficients:

```
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.08356     0.07297   14.85  <2e-16 ***
Petal.Width  2.22994     0.05140   43.39  <2e-16 ***
---

```

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

Residual standard error: 0.4782 on 148 degrees of freedom

Multiple R-squared: 0.9271, Adjusted R-squared: 0.9266

F-statistic: 1882 on 1 and 148 DF, p-value: < 2.2e-16

Un objeto de distinto tipo: t test

¿Y si llevamos a cabo un test t? Con el objetivo de mostrar cómo funciona, utilizaremos el ejemplo que se muestra en la documentación de `t.test`

```
t.test(1:10, y = 7:20, var.equal = TRUE) #Se asume distinta varianza por defecto
```

Two Sample t-test

data: 1:10 and 7:20

t = -5.1473, df = 22, p-value = 3.691e-05

alternative hypothesis: true difference in means is not equal to 0

95 percent confidence interval:

```
-11.223245 -4.776755
```

sample estimates:

mean of x mean of y

```
5.5      13.5
```

```
t.test(1:10, y = c(7:20, 200)) #No significativo debido al valor atípico 200
```

Welch Two Sample t-test

data: 1:10 and c(7:20, 200)

t = -1.6329, df = 14.165, p-value = 0.1245

alternative hypothesis: true difference in means is not equal to 0

95 percent confidence interval:

```
-47.242900 6.376233
```

sample estimates:

mean of x mean of y

```
5.50000 25.93333
```

```
#Fórmula para comparar la variable extra en dos grupos distintos de la base de datos sleep
#mu es el valor de la diferencia de medias que queremos comprobar (0 por defecto)
```

```
test <- t.test(extra ~ group, data = sleep, mu=1)
test
```

Welch Two Sample t-test

```
data: extra by group
t = -3.0385, df = 17.776, p-value = 0.007141
alternative hypothesis: true difference in means is not equal to 1
95 percent confidence interval:
 -3.3654832  0.2054832
sample estimates:
mean in group 1 mean in group 2
      0.75      2.33
```

```
class(test) #test es de tipo "htest", un objeto test de hipótesis
```

```
[1] "htest"
```

```
typeof(test) #Pero R lo trata como una lista
```

```
[1] "list"
```

```
names(test)
```

```
[1] "statistic" "parameter" "p.value"    "conf.int"  "estimate"
[6] "null.value" "alternative" "method"     "data.name"
```

```
str(test)
```

List of 9

```
$ statistic : Named num -3.04
..- attr(*, "names")= chr "t"
$ parameter : Named num 17.8
..- attr(*, "names")= chr "df"
$ p.value   : num 0.00714
$ conf.int  : num [1:2] -3.365 0.205
..- attr(*, "conf.level")= num 0.95
$ estimate  : Named num [1:2] 0.75 2.33
..- attr(*, "names")= chr [1:2] "mean in group 1" "mean in group 2"
$ null.value : Named num 1
..- attr(*, "names")= chr "difference in means"
$ alternative: chr "two.sided"
$ method     : chr "Welch Two Sample t-test"
$ data.name  : chr "extra by group"
- attr(*, "class")= chr "htest"
```

Funciones de la familia “apply”

Existen funciones especiales en R que podemos usar para aplicar otras funciones a los datos. Uno de los casos comunes es cuando queremos calcular un valor para cada categoría de un factor. Afortunadamente, R tiene algunas herramientas para hacer esto de forma sencilla. Este tipo de funciones se conocen como funciones de la familia `*apply` (`apply`, `lapply`, `sapply`, `vapply`, `tapply`, etc.). En nuestra opinión la función más útil de esta familia es la función `tapply`.

Los argumentos de esta función son una variable, un factor y una función que queremos aplicar sobre dicha variable para los distintos grupos definidos por cada nivel del factor. Por ejemplo, para calcular la media de la variable `Petal.Length` de la base de datos `iris` para las distintas especies utilizamos:


```
#The function tapply
tapply(X = iris$Petal.Length, INDEX = iris$Species, FUN = mean) #media de cada grupo
```

```
setosa versicolor virginica
1.462      4.260      5.552
```

```
#¿Cuántos elementos en cada grupo?
```

```
tapply(X = iris$Petal.Length, INDEX = iris$Species, FUN = length)
```

```
setosa versicolor virginica
50      50      50
```

Práctica (Opcional)

Realiza los siguientes ejercicios usando la base de datos “Energy Digestibility”:

- Crea una matriz de correlación llamada **cors** para calcular la correlación entre las columnas de la base de datos “Energy Digestibility”.
- Crea un modelo lineal entre dos variables que estén altamente correlacionadas y guárdalo bajo el nombre de **modelo**. Muestra los coeficientes del modelo en la consola.
- Calcula la media de la variable **Crude Fiber** para cada grupo de plantas (usando **tapply**).
- Calcula la desviación estándar de **Crude Fiber** para cada grupo de plantas usando la función **tapply**.
- ¿De qué clase son los resultados devueltos por la función **tapply**?
- Lleva a cabo un test t para ver si existen diferencias significativas en la media de la variable **EnergyDigest** cuando comparamos patatas con guisantes. **Pista:** Necesitarás usar el lenguaje lógico que aprendiste en secciones anteriores.

Gráficos en R

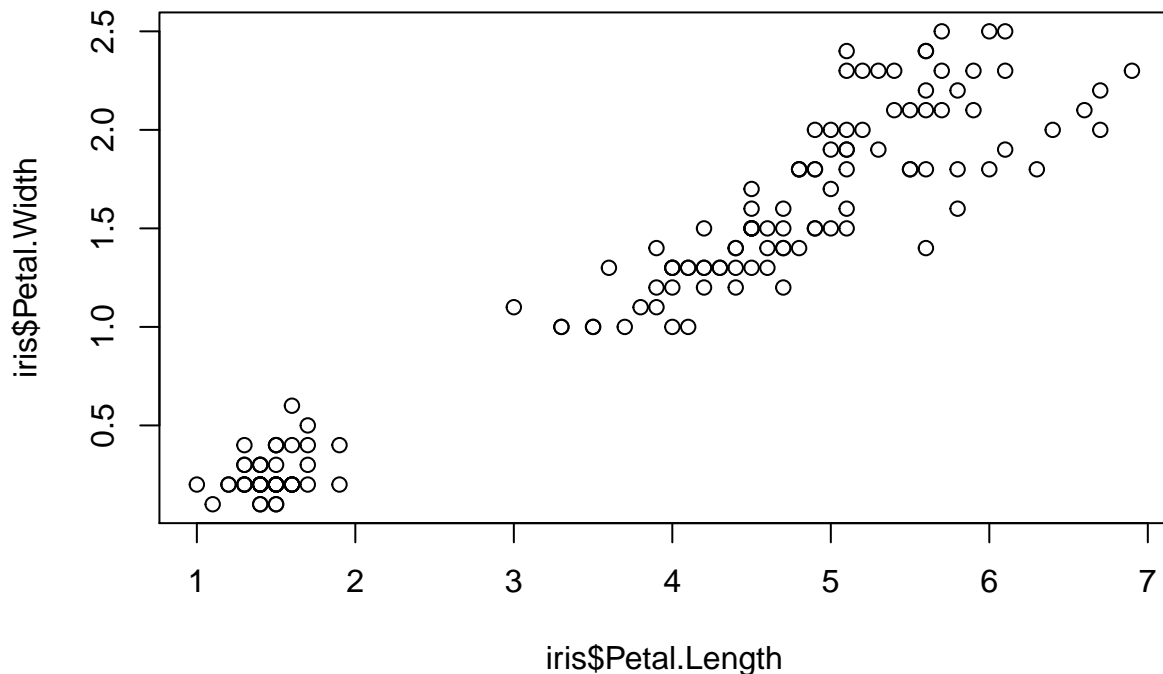
R tiene un gran potencial para realizar gráficos. En este curso introducimos brevemente algunos de los gráficos básicos usando la base de datos **iris**. Utiliza **demo(graphics)** y **demo(image)** para ver algunos ejemplos de demostración.

Diagrama de dispersión

El diagrama de dispersión o gráfico de puntos se utiliza generalmente para representar dos variables numéricas de datos emparejados. Cada variable se representa en un eje.

```
#Ejemplo de diagrama de dispersión
```

```
plot(x = iris$Petal.Length, y = iris$Petal.Width)
```



```
#Establece los nombres de los ejes, el símbolo para representar los puntos,
#el tamaño de los puntos, colores etc.
plot(iris$Petal.Length, iris$Petal.Width, pch=20, cex=0.8,
xlab="Petal length (cm)", ylab="Petal width (cm)",
main="Fisher/Anderson Iris data",
col=c("red", "blue", "green")[as.numeric(iris$Species)])

# Añade líneas horizontales y verticales para las medias
# El argumento lty especifica el estilo de la línea
# 0=en blanco, 1=sólida, 2=guiones, 3=puntos, 4=puntos y guiones, 5=guiones largos,
#6=guiones dobles
abline(v=mean(iris$Petal.Length), lty=2, col="red")
abline(h=mean(iris$Petal.Width), lty=4, col="red")

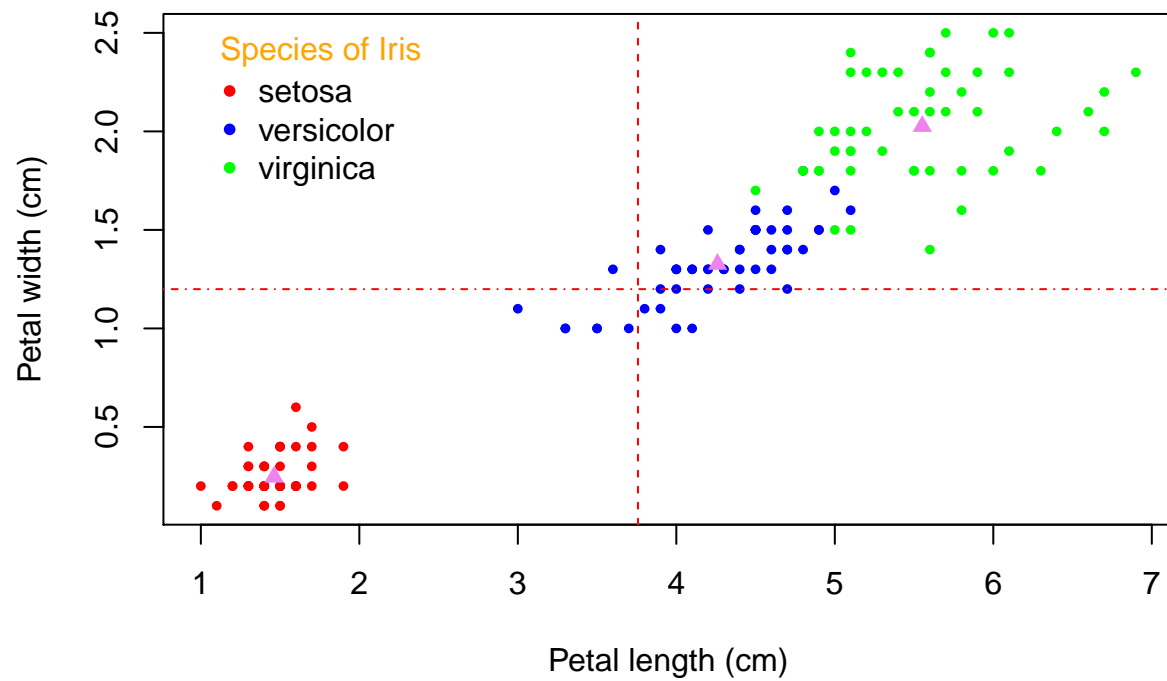
#Añade el centroide para cada una de las especies
mpl <- tapply(iris$Petal.Length, iris$Species, mean)
mpw <- tapply(iris$Petal.Width, iris$Species, mean)

#Añade los puntos
points(mpl,mpw,pch=17, col="violet")

#Añade texto
#pos=4 justificación por la izquierda
text(1,2.4,"Species of Iris",col='orange',pos=4)
#Añade la leyenda
legend(1, 2.4, levels(iris$Species), pch=20, bty="n",
```

```
col=c("red", "blue", "green"))
```

Fisher/Anderson Iris data

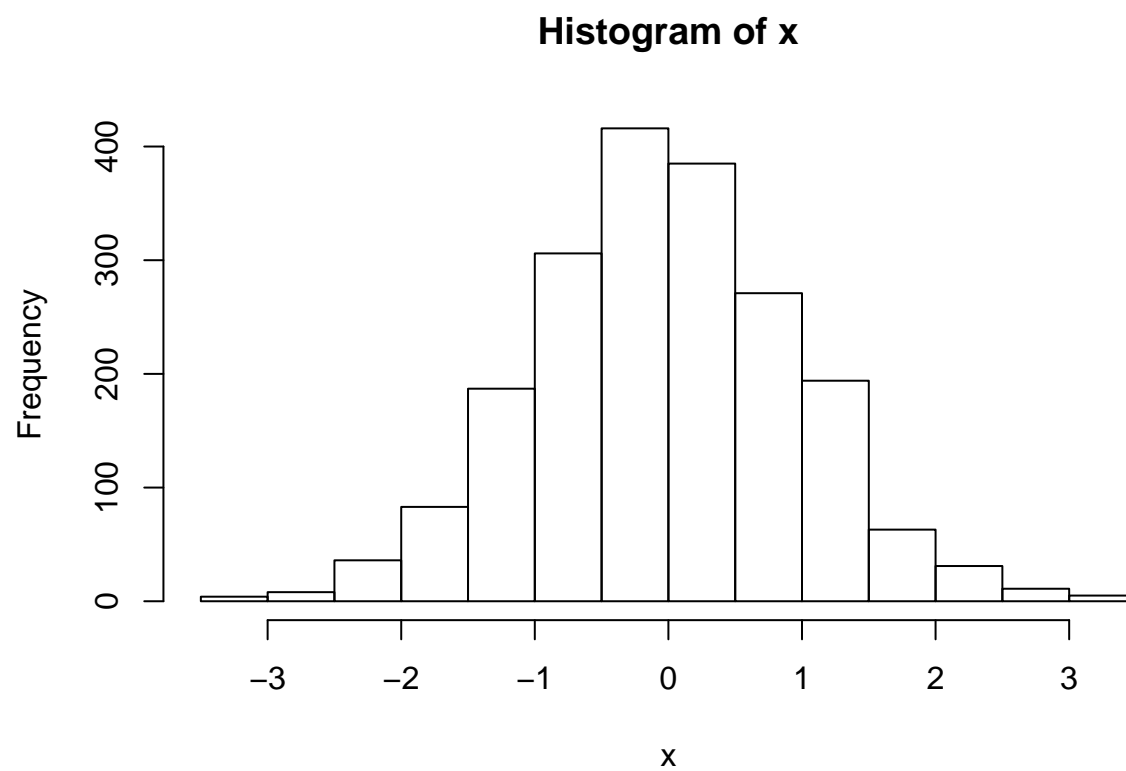


Histogramas

Los histogramas se usan para mostrar la distribución o frecuencia de una variable numérica. Se pueden generar fácilmente de la siguiente forma:

```
#Generar 2000 puntos aleatorios de una distribución normal estándar
x <- rnorm(2000, 0, 1)

#Crea el histograma
hist(x)
```



```
#Decora el gráfico  
hist(x, main="", col="royalblue", xlab="yield")
```

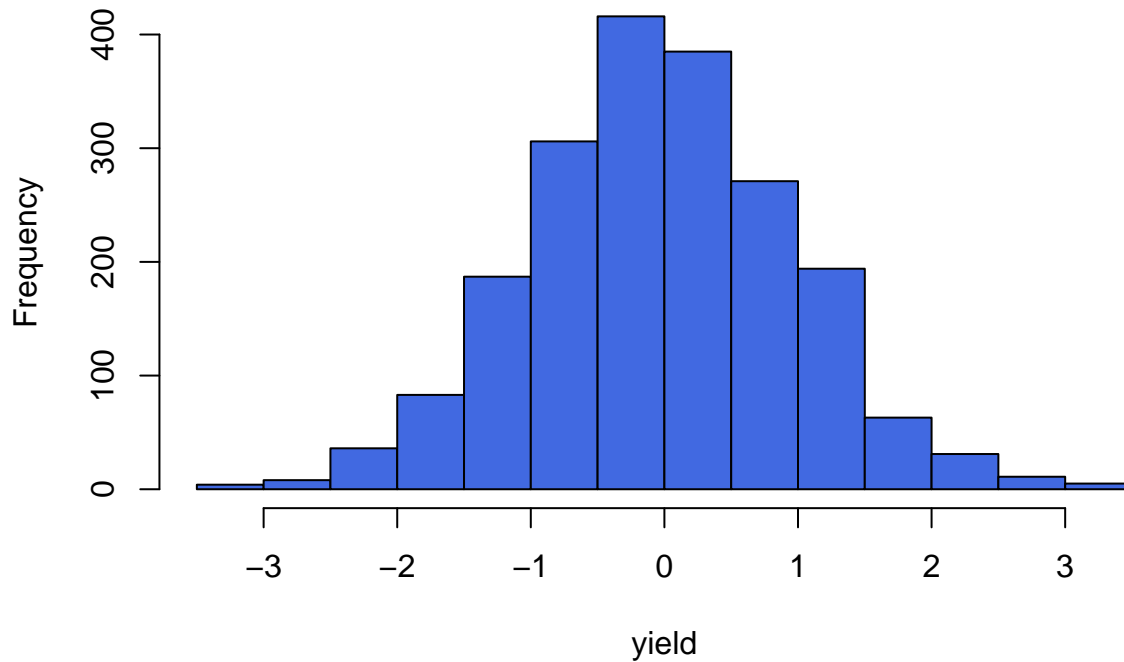
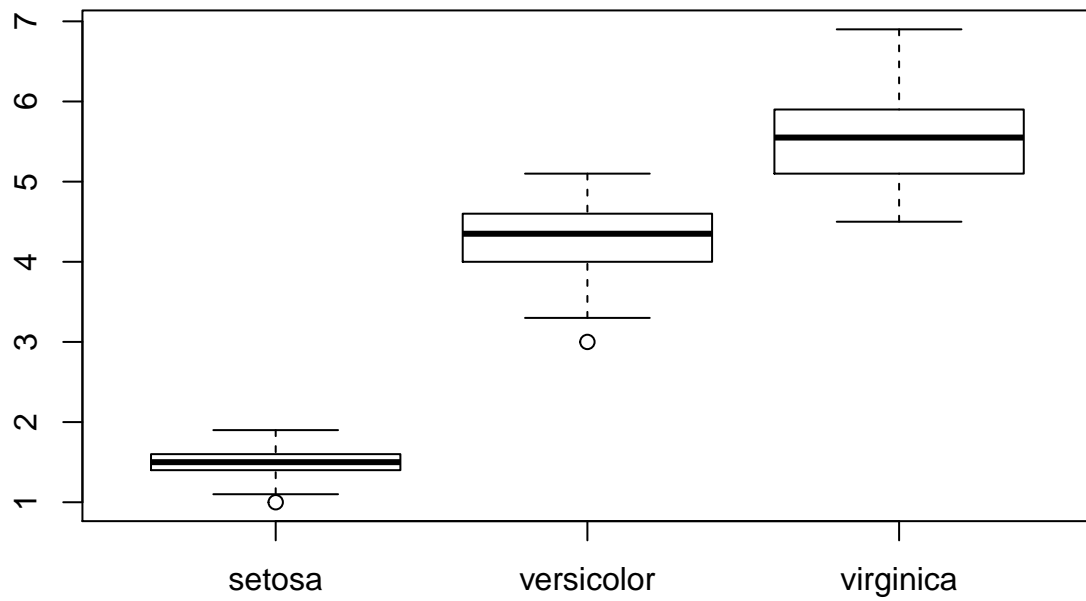


Diagrama de cajas

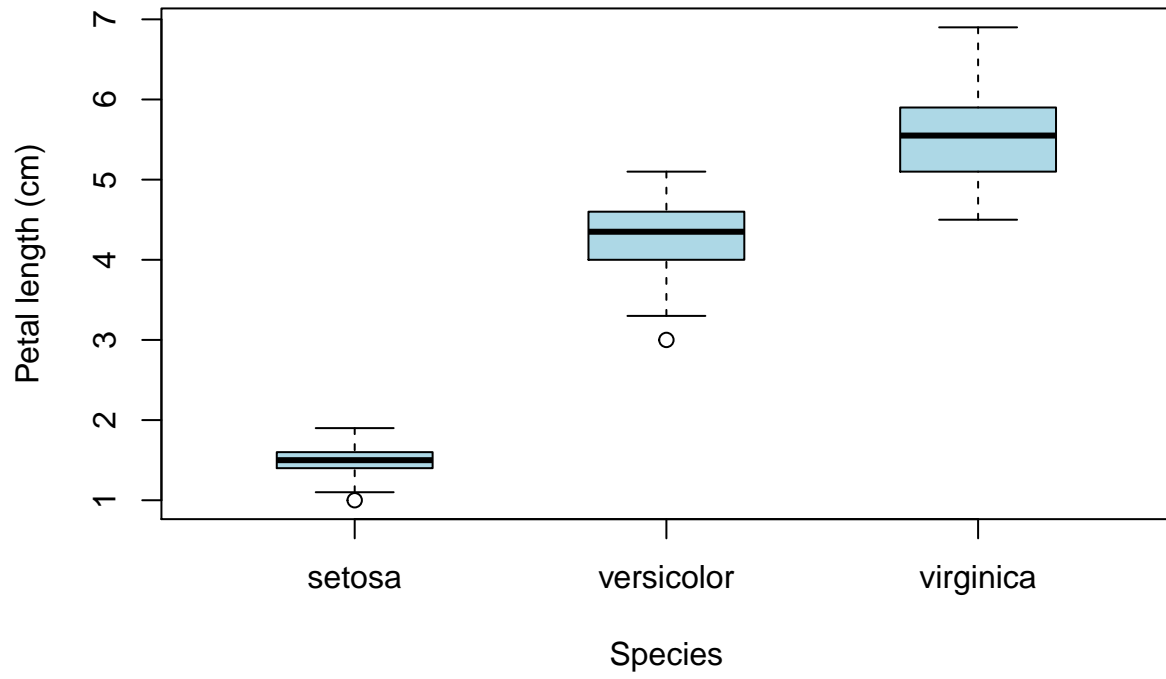
Los diagramas de cajas resultan útiles para representar gráficamente una serie de datos numéricos a través de sus cuartiles. Normalmente, se realizan diagramas de cajas de varios grupos para compararlos. Los diagramas de cajas nos pueden dar una idea del rango y la varianza de los distintos grupos, así como su distribución. Utilizamos la notación (\sim) para separar la variable numérica que se encuentra a la izquierda de la tilde y la variable x (normalmente un factor) a la derecha.

```
#Podemos usar el argumento data para evitar repetir todo el tiempo iris$  
#Diagrama de cajas de la variable numérica Petal.Length para los distintos niveles de Species  
boxplot(data=iris, Petal.Length ~ Species)
```

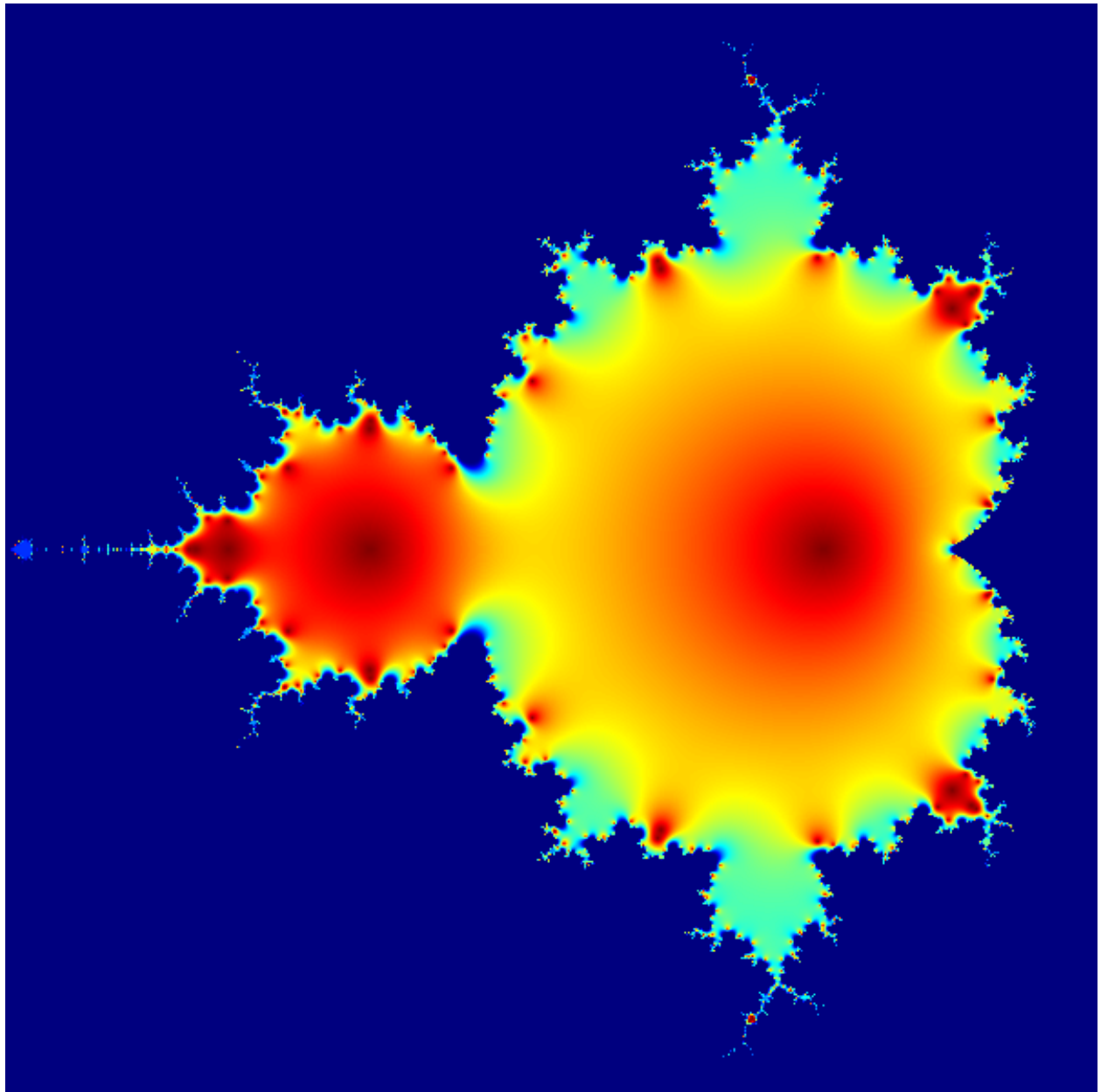


```
#Decora el gráfico  
boxplot(data=iris, Petal.Length ~ Species,  
col="lightblue", boxwex=.5,  
xlab="Species", ylab="Petal length (cm)",  
main="Box plot")
```

Box plot

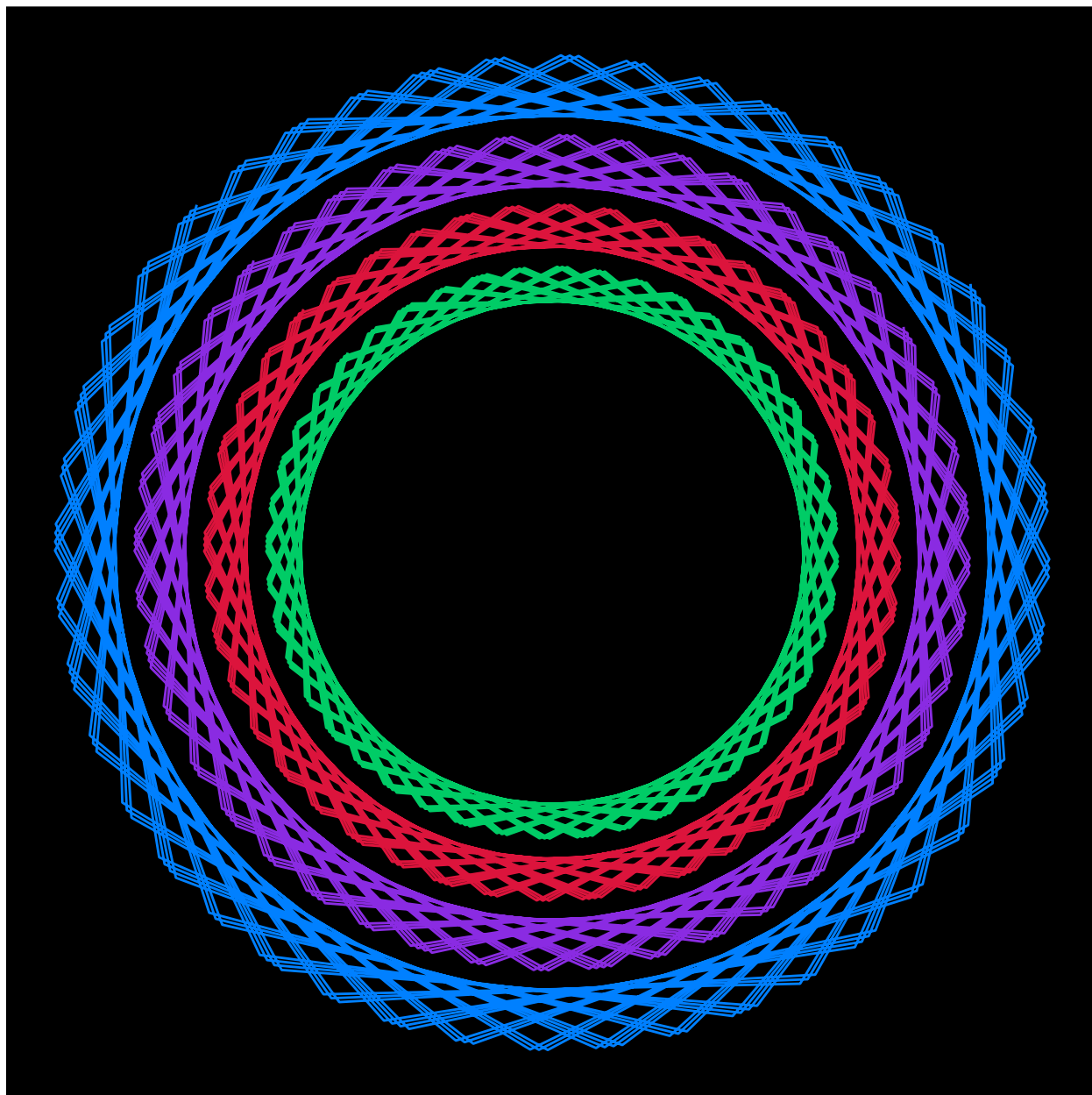


También hecho en R
El conjunto de Mandelbrot

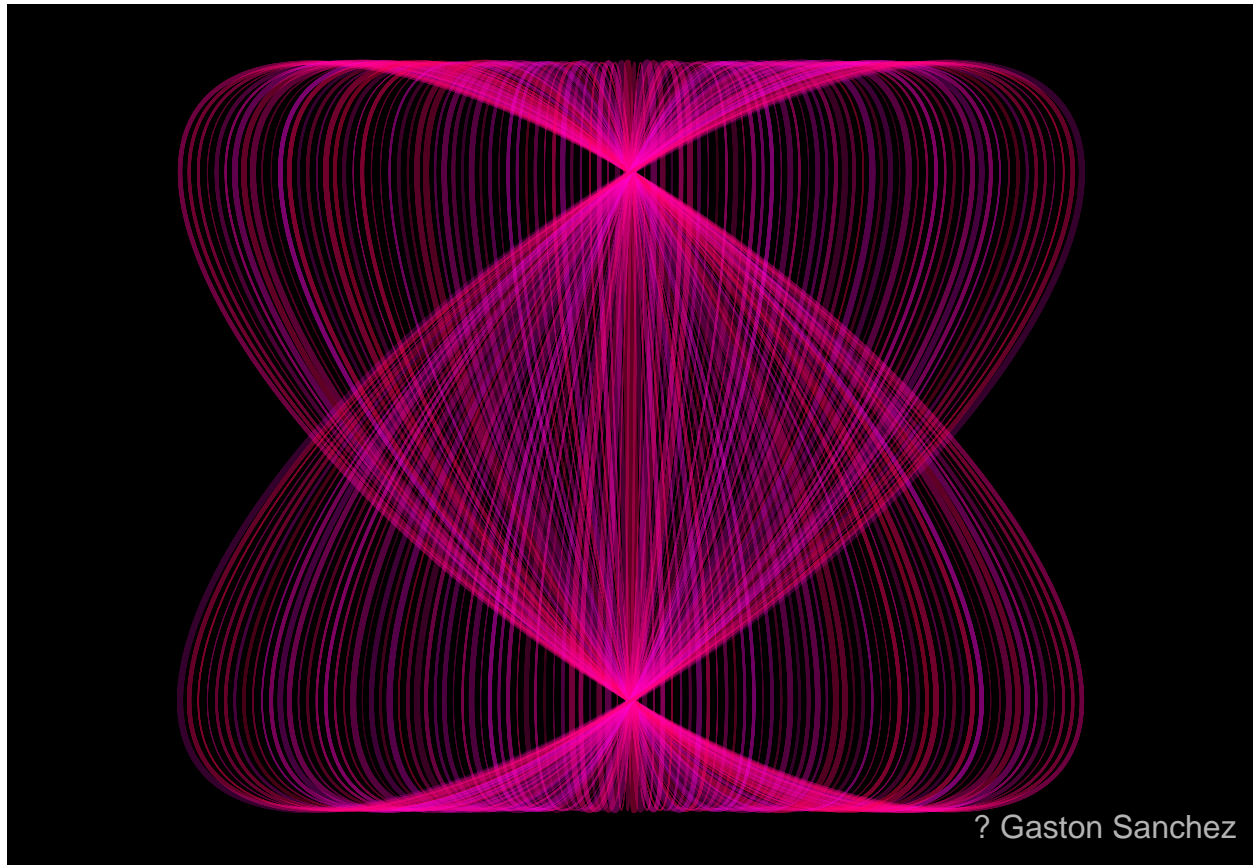


Este gráfico ha sido producido en R mostrando las 20 iteraciones de este conjunto.

Círculos concéntricos



Doble corazón



Práctica

Usando los datos “Energy Digestibility”:

- Produce un histograma de una de sus variables
 - Produce un diagrama de puntos de dos variables que estén altamente correlacionadas y observa la relación lineal
 - Produce un diagrama de cajas de una variable agrupada por factores. Añade un título apropiado y utiliza el color azul para las cajas.
 - Mismo diagrama de cajas coloreado por grupos de plantas
-

Más recursos

Recursos para el autoaprendizaje

Para continuar practicando en casa, te recomendamos alguno de estos recursos:

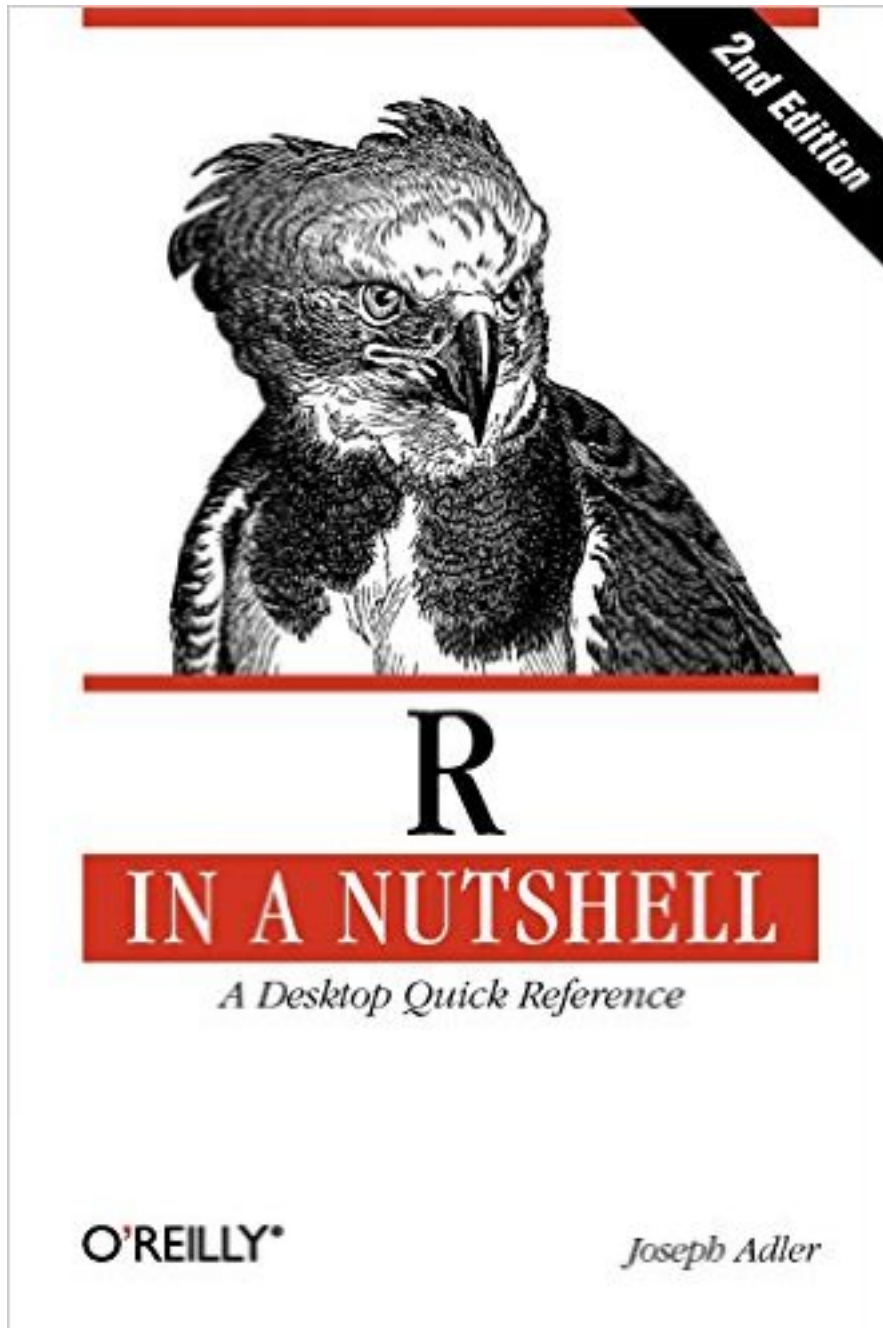
- Swirl - simplemente ejecuta `library(swirl)` y escribe `swirl()` en la consola y sigue las instrucciones para practicar más.
- Code School: <http://tryr.codeschool.com/>
- Datacamp - Introduction to R: <https://www.datacamp.com/courses/free-introduction-to-r>

- Quick-R Website: <http://www.statmethods.net/>. Para explicaciones simples de muchas de las características de R y sus funciones.
- ¡Google!

Libros

Te recomendamos el siguiente:

- *R in a Nutshell* escrito por Joseph Adler



Gestión y limpieza de datos

- Tidy data por Hadley Wickham: <https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html>
- Alguna de las guías simples para una gestión de datos efectiva: Borer, Elizabeth T., Eric W. Seabloom, Matthew B. Jones, and Mark Schildhauer. Bull. Ecol. Soc. Am. 90(2)205-214. 2009. <http://www.nceas.ucsb.edu/files/news/ESAdatamng09.pdf>
- Cornell University Research Data Management Service Group: <https://data.research.cornell.edu/content/tabular-data>

Otra información

- Página de R: <https://www.r-project.org/>
 - Página de RStudio: <https://www.rstudio.com/products/rstudio/>
 - R FAQ: <https://cran.r-project.org/doc/FAQ/R-FAQ.html>
 - Más sobre la instalación de paquetes en R y el manual de administración: https://cran.r-project.org/doc/manuals/R-admin.html#Add_002don-packages
 - Características de la programación en R: [https://en.wikipedia.org/wiki/R_\(programming_language\)#Programming_features](https://en.wikipedia.org/wiki/R_(programming_language)#Programming_features)
 - Galería de la gráfica de R Graph: <http://www.r-graph-gallery.com/>
-

Agradecimientos

El contenido de este curso fue originalmente desarrollado por Beverley Gogel y Sabela Muñoz Santa y ha sido adaptado en su formato actual por Sam Rogers. La traducción de las notas al español ha sido realizada por Sabela Muñoz Santa.

La sección de la gestión de datos fue adaptada de varias fuentes incluyendo:

- Limpieza de datos por Hadley Wickham: <https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html>
- Alguna de las guías simple de gestión de datos efectiva. Borer, Elizabeth T., Eric W. Seabloom, Matthew B. Jones, and Mark Schildhauer. Bull. Ecol. Soc. Am. 90(2)205-214. 2009. <http://www.nceas.ucsb.edu/files/news/ESAdatamng09.pdf>
- Cornell University Research Data Management Service Group: <https://data.research.cornell.edu/content/tabular-data>

Las gráficas del doble corazón y los círculos concéntricos fueron originalmente producidas por [Gastón Sánchez](#), y utilizados con su permiso.