Laboratorio: Introducción a la estadística usando R

Jose Ameijeiras Alonso
(basado en el material de Beatriz Pateiro López)

El entorno R es un conjunto integrado de programas para manejo y análisis de datos, cálculo y creación de gráficos. Se trata de un proyecto de software libre iniciado por Ross Ihaka y Robert Gentleman (R&R) en los años 90. En la actualidad, el *R Development Core Team* es el responsable de su desarrollo y mantenimiento. Además, la comunidad de usuarios que programan en R ha crecido considerablemente, proporcionando bibliotecas o paquetes con funcionalidades adicionales a las proporcionadas por la distribución básica. El objetivo de esta sesión es familiarizar al alumno con el entorno y sus principales utilidades. Al mismo tiempo, revisaremos algunas de las técnicas estadísticas más habituales y su implementación en R.

1 Introducción al lenguaje R

El sistema de R consta de un sistema base y de paquetes adicionales que extienden su funcionalidad (contributed packages). Todo lo necesario para instalar R se puede obtener a través de la página web del proyecto. Este es también el sitio principal para encontrar más información sobre R: documentación general, manuales, tutoriales, etc.

Al ejecutar R, verás que dentro de la ventana (R GUI) aparece una barra de menú, una barra de herramientas y la **consola de R**, en donde escribiremos los comandos de R y obtendremos los resultados del programa. Para comenzar, como R permite realizar cálculos aritméticos, podríamos utilizar el programa como una potente calculadora. Los operadores artiméticos básicos son `+`, `-`, `*`, `/`, `^` para elevar a una potencia, `%%` para el resto de la división entera, `%/%` para el cociente de la división entera.

```
> 7 * (3 + 2)/5
```

La mayor parte del trabajo en R se lleva a cabo mediante **funciones**. Por ejemplo, si queremos calcular $\sqrt{9}$ en R escribimos:

```
> sqrt(9)
```

Esto le dice a R que llame a la función `sqrt`. Entre paréntesis indicamos los argumentos de la función. R nos permite asignar valores a variables y referenciarlas por su nombre. Usaremos el símbolo `<-` para asignar valores a variables. Recuerda que R distingue entre minúsculas y mayúsculas (case sensitive). Por ejemplo,

```
> x <- 3
> x
```

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

1.1 Comandos y conceptos básicos

Describimos aquí algunos aspectos básicos de R.

- **Obteniendo ayuda:** Para obtener ayuda acerca de, por ejemplo, la función `sqrt`, se podrían utilizar los siguientes comandos `?sqrt`, `help(sqrt)`, `help.search("sqrt")`. Los dos primeros son equivalentes y el tercero permite realizar búsquedas más avanzadas.
- **Case sensitivity:** El lenguaje R es "case sensitive", es decir, distingue entre mayúsculas y minúsculas.
- **Funciones:** Para consultar el código de una función en R, ya sea una función propia de R o bien una del usuario, basta con teclear el nombre en la línea de comandos (sin paréntesis ni ningún tipo de argumentos).
- **Workspace:** El comando `getwd()` nos devuelve el directorio de trabajo y el comando `setwd` nos permite modificarlo. Al iniciar el programa R se abre automáticamente un espacio de trabajo, en él se almacenan todos los datos, funciones, etc. usadas durante esa sesión. En cualquier momento se pueden guardar todos los objetos del espacio de trabajo como un archivo con extensión .RData. Si en sesiones posteriores se necesitan se pueden cargar con la función `load`.
- **Librerías:** Al iniciar el programa R se cargan por defecto unas librerías básicas. A veces es necesario cargar otras librerías para realizar ciertos análisis. Esto se hace a través del comando `library`.
- **Listados:** Cuando introducimos datos en R (ya sea directamente a través del teclado o importándolos a partir de un fichero de texto), solemos guardar el resultado en un objeto. Para obtener un listado de los objetos disponibles en el directorio de trabajo, tecleamos `ls()`. Además, el comando `search()` nos da una lista de las librerías cargadas.
- **Borrado:** Podemos eliminar un objeto con la función `remove` o, equivalentemente, `rm`.

1.2 Objetos en R

R dispone de gran variedad de objetos, tales como escalares, vectores (numéricos, carácter, lógicos), factores, matrices y arrays, listas y data frames. El tipo de objeto que utilicemos dependerá de la estructura de datos que manejemos y del análisis que queramos realizar. Describiremos de forma breve algunas características de las estructuras de datos más importantes de R.

1.2.1 Vectores

El vector es la estructura de datos más simple en R. Un vector es un objeto que consiste en un número de elementos, todos ellos del mismo tipo. La forma más básica de definir un vector en R es introducir sus elementos uno a uno. Para ello se usa la función `c`, que concatena los valores introducidos en un mismo objeto. Por ejemplo:

```
> v <- c(4, 5, 23.8, 67)
> v
```

```
## [1]  4.0  5.0 23.8 67.0
```

```
> x <- c(v, v)
> x
```

```
## [1]  4.0  5.0 23.8 67.0  4.0  5.0 23.8 67.0
```

```
> x <- c(3, 5, 12)
> class(x)
```

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

```
> y <- c("Luis", "Pedro", "Laura")
> class(y)
```

```
## [1] "character"
```

Todos los elementos de un vector deben ser del mismo tipo. Si concatenamos elementos de diferentes tipos, el vector tendrá el tipo "menos restrictivo". Por ejemplo, si concatenamos números y caracteres, el vector resultante será de tipo carácter.

Además del comando `c`, existen otras funciones útiles para crear vectores. La función `seq` se utiliza para generar secuencias de números equidistantes. Por ejemplo, si queremos generar una sucesión de números que empiece en a, termine en b con incremento s, utilizamos la sintaxis `seq(from = a, to = b, by = s)`. Es decir, la opción `by` indica la distancia entre dos elementos consecutivos. Así, para generar la secuencia de los números pares de dos a cien basta escribir:

```
> seq(2, 100, by = 2)  # de 2 a 100 de 2 en 2
```

En caso de que el incremento sea 1, también podemos usar la notación `:`.

```
> 1:10
```

Si queremos generar una sucesión de números que empiece en a, termine en b y tenga longitud l, entonces utilizaremos la sintaxis `seq(from = a, to = b, length = l)`. R calcula la distancia para colocar el número de elementos pedido entre el extremos inferior y superior. Por ejemplo,

```
> seq(1, 10, length = 6)  # 6 números equidistantes entre 1 y 10
```

Cuando escribimos `seq(1, 10, length = 7)`, R asume que el primer valor que introducimos corresponde al valor inicial de la secuencia y que el segundo valor que introducimos corresponde al valor final de la secuencia (*positional matching*). Esto significa que no necesitamos especificar el nombre de los argumentos siempre que los introduzcamos en el mismo orden en el que aparecen en la lista de argumentos de la función. Si una función tiene muchos argumentos y no sabemos en que orden están definidos, lo que debemos hacer es especificar el nombre del argumento que estamos pasando. Suele ser suficiente con indicar la parte inicial del nombre del argumento siempre que no se genere confusión con otros argumentos (*partial matching of names*).

1.2.1.1 Indexado de vectores

Para conocer el valor de la componente i-ésima de un vector `x` tendremos que escribir `x[i]`. Por ejemplo:

```
> x[3]  # Tercer elemento de  x
```

Es posible recuperar más de una componente simultáneamente. Por ejemplo:

```
> y[c(1, 3)]  # Primer y tercer elemento de y
> y[1:3]  # Primeros 3 elementos de y
```

Incluso podemos indicar las componentes que no queremos recuperar:

```
> y[-2]  # Todos los elementos de y salvo el segundo
```

Podemos operar con vectores utilizando los operadores habituales (`+`,`-`, `*`, `/`, ...). Además, existen comandos en R que están especialmente pensados para trabajar con vectores. En el siguiente cuadro se muestran algunos de ellos.

Función R	Descripción
`sum(x)`	suma de los elementos de `x`
`cumsum(x)`	vector con la suma acumulada hasta cada componente `x`
`prodsum(x)`	producto de los elementos de `x`
`cumprod(x)`	vector con producto acumulado de `x`
`max(x)`	máximo de `x`
`min(x)`	mínimo de `x`
`length(x)`	longitud de `x`
`sort(x)`	ordena los elementos de `x` de menor a mayor
`mean(x)`	media aritmética de `x`
`union(x,y)`	union de los vectores `x` e `y` sin repetir
`intersect(x,y)`	comunes sin repetir en los vectores `x` e `y`
`setdiff(x,y)`	cuáles de `x` no están en `y`

1.2.2 Matrices

Una matriz es una colección de elementos dispuestos de forma rectangular (en filas y columnas). Como ocurría con los vectores, los elementos de una matriz deben ser todos del mismo tipo. Una matriz para R es esencialmente un vector que tiene dimensiones, donde las filas y las columnas juegan un papel bastante simétrico (cosa que no ocurre con los data frames). La forma habitual de crear matrices en R es con la función `matrix`.

```
> a <- matrix(10:15, nrow = 2, ncol = 3)
> a
```

```
##      [,1] [,2] [,3]
## [1,]   10   12   14
## [2,]   11   13   15
```

```
> class(a)
```

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

```
> dim(a)  # Dimensión de una matriz [nº de filas, nº de columnas]
```

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

Las matrices se van rellenado por columnas. Utiliza el argumento `byrow = TRUE` para rellenar una matriz por filas.

```
> b <- matrix(10:15, nrow = 2, ncol = 3, byrow = TRUE)
> b
```

```
##      [,1] [,2] [,3]
## [1,]   10   11   12
## [2,]   13   14   15
```

```
> class(b)
```

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

```
> dim(b)
```

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

Otras funciones útiles para definir matrices son `cbind` y `rbind`. La función `cbind` crea una matriz combinando 2 o más vectores por columnas. La función `rbind` crea una matriz combinando 2 o más vectores por filas.

```
> d <- cbind(x, 2 * x)
> d
```

```
##       x
## [1,]  3  6
## [2,]  5 10
## [3,] 12 24
```

1.2.2.1 Indexado de matrices. Como hacíamos con los vectores, haremos referencia a elementos de una matriz utilizando los corchetes. Hacemos referencia a las filas con el primer índice y a las columnas con el segundo índice (ambos separados por una coma).

```
> a[2, 3]  # Elemento de a en la segunda fila, tercera columna
> a[2, ]  # Segunda fila de a
> a[, 3]  # Tercera columna de a
```

Al igual que pasaba con los vectores podemos pedir un conjunto de filas y/o columnas. Por ejemplo,

```
> a[, c(1, 3)]  # Columnas 1 y 3
```

También podemos eliminar columnas utilizando el signo menos. Por ejemplo,

```
> a[, -1]  # Matriz a sin la primera columna
```

R nos permite realizar operaciones aritméticas con matrices utilizando los operadores habituales (`+`, `-`, `*`, `/`, ...). Las operaciones se realizan elemento a elemento. Por ejemplo:

```
> a * b  # Producto elemento a elemento a[i,j]*b[i,j]
```

El producto matricial en R se obtiene con el operador `%*%`. Es decir, dadas dos matrices `A` y `B`, el comando `A%*%B` multiplica las filas de la matriz `A` por las columnas de la matriz `B` (siempre que que el número de columnas de `A` sea igual al número de filas de `B`).

Existen comandos en R que están especialmente pensados para trabajar con matrices. Algunos de ellos se muestran en el siguiente cuadro.

Función R	Descripción
`dim(A)`	dimensión de `A`
`nrow(A)`	número de filas de `A`
`ncol(A)`	número de columnas de `A`
`rownames(A)`	nombre de las filas de `A`
`colnames(A)`	nombre de las columnas de `A`
`t(A)`	traspuesta de la matriz `A`
`crossprod(A,B)`	producto `t(A)%*%B`
`crossprod(A)`	producto `t(A)%*%A`
`solve(A)`	inversa de `A`
`rowMeans(A)`	medias por filas
`rowSums(A)`	sumas por filas
`colMeans(A)`	medias por columnas
`colSums(A)`	sumas por columnas

La función `apply` permite repetir una misma operación en cada una de las filas (o columnas) de una matriz. En muchos casos esto es más rápido que otras alternativas para repetir una operación. Su sintaxis es `apply(X, MARGIN, FUN)`. En esta expresión `X` es una matriz, `MARGIN` puede valer 1 o 2 y `FUN` es un nombre de comando que actúa sobre vectores. Si `MARGIN = 1` se genera un nuevo vector como resultado de aplicar `FUN` a cada fila de `X`. Si `MARGIN = 2`, se genera un nuevo vector como resultado de aplicar `FUN` a cada columna de `X`.

1.2.3 Data frames

Los data frame son la estructura de datos más importante de R. Un data frame es un objeto con filas y columnas. Es más general que una matriz ya que permite combinar en un mismo objeto columnas de distintos tipos (en un data frame podemos tener, por ejemplo, columnas numéricas y columnas de tipo carácter, mientras que en una matriz todos los elementos deben ser del mismo tipo). Trabajaremos como ejemplo con el conjunto de datos conocido como *Fisher's iris data*. Este conjunto nos da la medida en cm. de las variables longitud y anchura de sépalo y longitud y anchura de pétalo para un total de 150 flores de tres especies diferentes de iris (Iris setosa, versicolor y virginica) y están disponibles en R (R nos permite cargar en el directorio de trabajo conjuntos de datos que ya existen en formato R y que forman parte de alguna de las librerías instaladas). Para acceder a los datos en R usaremos:

```
> data(iris)
> head(iris)  # Muestra las primeras filas del conjunto 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
## 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
```

```
> class(iris)
```

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

Puesto que un data frame puede ser considerado como una generalización de las matrices, muchas de las funciones que operan sobre matrices también se pueden utilizar con data frames.

1.2.3.1 Indexado de data frames Obtenemos los nombres de las variables de un data frame con la función `names`.

```
> names(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
## [5] "Species"
```

Podemos acceder a los datos de las variables de un data frame de manera individual con la notación `$`. Por ejemplo,

```
> iris$Sepal.Length
```

Los métodos de indexado que utilizábamos con las matrices también funcionarán con data frames.

```
> iris[3, 5]  # Especie (5ª columna) de la tercera observación
```

También podemos usar el indexado para extraer subconjuntos de un data frame. Por ejemplo, si queremos obtener las anchuras de sépalo y especie de todas las observaciones, escribiremos:

```
> iris[, c("Sepal.Width", "Species")]
```

1.3 Uso de condiciones lógicas para seleccionar subconjuntos de datos

En la práctica, es habitual trabajar con subconjuntos de datos del conjunto original que satisfagan ciertos criterios. Por ejemplo, nos puede interesar conocer las longitudes de pétalo de aquellas flores cuya longitud de sépalo es superior a 5 cm o las anchuras de pétalo de las flores de la especie Virginica. Podemos seleccionar subconjuntos utilizando condiciones lógicas en R. El siguiente cuadro muestra los operadores lógicos en R. El uso de operadores lógicos con vectores es extremadamente útil.

`<`	menor que
`<=`	menor o igual que
`>`	mayor que
`>=`	mayor o igual que
`==`	igual a
`!=`	no igual a

Un operador lógico permite comprobar si se cumple o no una determinada condición. Si se usa en un vector (por ejemplo un conjunto grande de medidas) podremos conseguir el número de casos que cumplen la condición. Por ejemplo,

```
> iris$Sepal.Length >= 5
```

Este comando nos diría qué elementos del vector `iris$Sepal.Length` son mayores o iguales a 5. Para los valores que cumplen la condición devuelve el valor lógico `TRUE` y para el resto `FALSE`. El resultado es un vector de valores de tipo lógico. Si queremos contar cuántos son podemos usar el comando `sum`, que suma los elementos de un vector. Así,

```
> sum(iris$Sepal.Length >= 5)
```

sumaría los elementos que aparecen etiquetados como `TRUE`.

Los operadores lógicos se pueden utilizar para hacer comparaciones entre vectores lógicos.

`&`	"y" lógico
`|`	"o" lógico
`!`	"no" lógico

Por ejemplo, para comprobar si las longitudes de sépalo están comprendidas entre 5 y 6 tendríamos que escribir

```
> (iris$Sepal.Length >= 5) & (iris$Sepal.Length <= 6)
```

Las funciones `any` y `all` son muy útiles para trabajar con vectores lógicos. La función `any` comprueba si algún valor es verdadero mientas que `all` se usa para saber si todos son verdaderos. Por ejemplo

```
> any(iris$Sepal.Length >= 5)
```

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

```
> all(iris$Sepal.Length >= 5)
```

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

También podemos combinar la sintaxis de indexado que hemos visto con condiciones lógicas para definir las componentes que queremos recuperar a través de una condición. Por ejemplo

```
> iris$Petal.Length[iris$Sepal.Length >= 5]
```

nos devuelve las longitudes de pétalo de aquellas flores cuya longitud de sépalo es superior a 5 cm. Para obtener las anchuras de pétalo de las flores de la especie Virginica escribimos:

```
> iris$Petal.Width[iris$Species == "virginica"]
```

A estos nuevos vectores les podríamos aplicar cualquier función de las que hemos visto para vectores como, por ejemplo, la media. Obtendríamos así la anchura media de pétalo de las flores de la especie Virginica:

```
> mean(iris$Petal.Width[iris$Species == "virginica"])
```

```
## [1] 2.026
```

1.4 Funciones gráficas y argumentos de funciones gráficas en R

Existen dos tipos básicos de funciones gráficas en R: las conocidas como funciones de primer nivel y las de segundo nivel.

- De primer nivel: Crean las ventanas gráficas y establecen sus coordenadas.
- De segundo nivel: Añaden elementos gráficos en ventanas gráficas ya abiertas y con coordenadas establecidas. Las funciones gráficas de segundo nivel no se pueden ejecutar si antes no se ha usado una función de primer nivel. R dará un error si se intentan usar sin abrir antes una ventana gráfica

La función `plot` es la función genérica de dibujo de primer nivel. Por tanto abrirá un gráfica con un sistema de coordenadas. En su versión más simple dibuja un gráfico de puntos.

```
> plot(iris$Petal.Width, iris$Petal.Length)
```

La función `plot` y el resto de funciones de primer nivel admiten muchos argumentos que nos permiten cambiar la apariencia del gráfico. En el siguiente cuadro se muestran algunos de los que más se utilizan.

`main`	título principal
`xlab`	etiqueta para el eje x
`ylab`	etiqueta para el eje y
`pch`	tipo de punto
`lty`	tipo de línea
`lwd`	ancho de línea
`col`	color
`xlim`	límites del eje x
`ylim`	límites del eje y

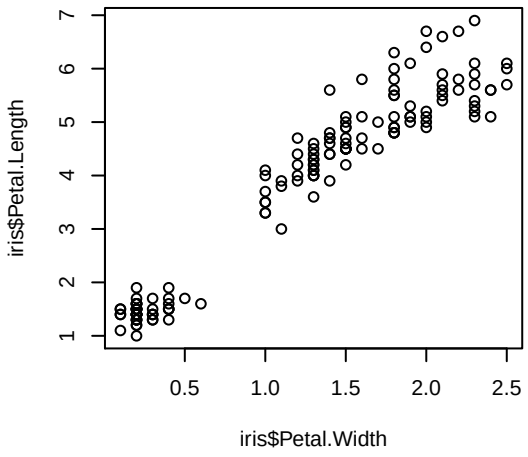


Figure 1: Diagrama de dispersión de la anchura de pétalo frente a logitud de pétalo en flores de iris

Así, podemos cambiar el color y tipo de punto para representar el diagrama de dispersión anterior.

```
> plot(iris$Petal.Width, iris$Petal.Length, col = 2, pch = 19)
```

Otra función de primer nivel es la función **pairs**, que representa matrices de diagramas de dispersión.

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

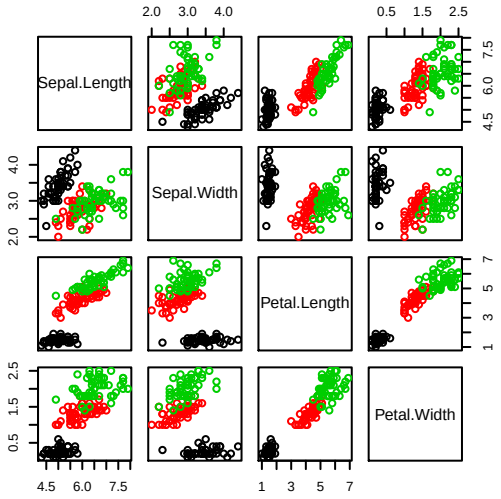


Figure 2: Diagramas de dispersión para datos de iris

Siempre que queramos añadir algún elemento nuevo a un gráfico ya creado (líneas, puntos, texto, etc.), debemos utilizar una función de segundo nivel. A continuación se muestran algunas de las principales funciones de segundo nivel en R.

`lines`	añade una línea a un gráfico
`points`	añade puntos a un gráfico
`abline`	añade una línea recta a un gráfico
`text`	añade texto a un gráfico
`legend`	añade una leyenda a un gráfico

1.5 Importar datos en R

Hasta ahora hemos visto una forma muy sencilla de introducir datos en R: directamente por teclado. Sin embargo esta no es la forma más habitual de introducir datos en R. Frecuentemente querremos que R lea datos de otros programas.

Son varias las funciones de R que nos permite leer datos dispuestos de forma rectangular en un fichero de texto (`read.table`, `read.csv`, ...). Todas ellas tienen como primer argumento:

- `file`: nombre del fichero que contiene los datos que queremos leer.

Además, estas funciones tienen otros argumentos. Algunos de los más importantes son:

- `header`: valor lógico (`TRUE` o `FALSE`) que indica si el fichero contiene en la primera fila los nombres de las variables.
- `sep`: carácter separador de columnas.
- `dec`: carácter utilizado para el punto decimal.

Las principales diferencias entre estos comandos están en qué caracteres se usan por defecto para separar los números de la tabla y cómo se representa la coma decimal de los números.

Función R	`header`	`sep`	`dec`
`read.table`	`FALSE`	`" "`	`"."`
`read.csv`	`TRUE`	`","`	`"."`
`read.csv2`	`TRUE`	`";"`	`","`
`read.delim`	`TRUE`	`"\t"`	`"."`
`read.delim2`	`TRUE`	`"\t"`	`","`

2 Análisis exploratorio de datos

> *It is important to understand what you can do before you learn to measure how well seem to have done it* (Tukey 1977)

El objetivo del análisis exploratorio de datos es identificar las principales características de un conjunto de datos por medio de resúmenes numéricos y gráficos. Estas herramientas nos permitirán conocer características básicas de la distribución de las variables, detectar relaciones entre ellas, reconocer la presencia de valores atípicos, etc.

Trabajaremos de nuevo con el conjunto de datos de iris. Comenzaremos con un análisis individual de cada variable del conjuntos de datos y posteriormente estudiaremos las relaciones entre variables. Debemos distinguir entre las variables cualitativas, que no tienen naturaleza numérica, y las cuantitativas que sí la tienen. Dentro de éstas últimas debemos además distinguir entre las variables de tipo discreto, donde es frecuente que se repitan los valores de la variable, y las continuas, donde no lo es (salvo que sus valores

estén redondeados numéricamente). A continuación se muestra cómo hacer en R los principales resúmenes numéricos y gráficos.

2.1 Tablas de frecuencias y resúmenes gráficos

Comenzaremos con la variable `Species` (variable cualitativa). El siguiente código muestra la tabla de frecuencias correspondiente.

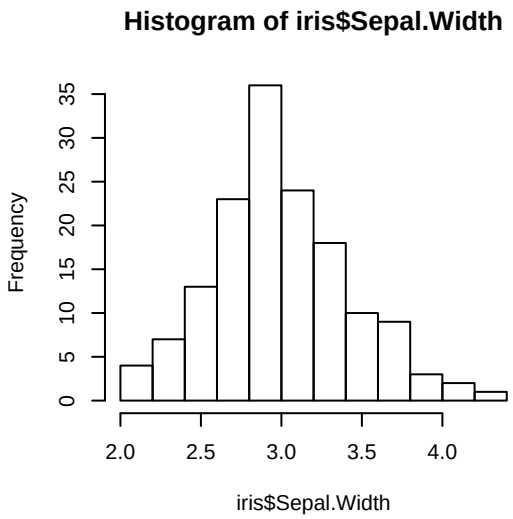
```
> fabs <- table(iris$Species)  # Frecuencia absoluta
> frel <- fabs/sum(fabs)  # Frecuencia relativa
```

Los datos de variables categóricas también se pueden resumir mediante gráficas representativas. Por ejemplo, para dibujar un diagrama de sectores utilizaremos la función `pie`. La función de R que permite crear un gráfico de barras es `barplot`.

```
> pie(fabs)  # Diagrama de sectores
> barplot(fabs)  # Diagrama de barras de frecuencias absolutas
```

El **histograma** en un método gráfico que nos permite resumir la información de una variable continua. Para obtener un histograma de, por ejemplo, la variable `Sepal.Width` se puede escribir:

```
> hist(iris$Sepal.Width)
```



Por defecto R selecciona los intervalos de agrupación con la misma longitud y en el eje vertical se representa la frecuencia absoluta de los datos que caen en cada intervalo. Si se desea representar un histograma de área uno, se debe seleccionar la opción `freq = FALSE`. También podemos introducir nosotros los puntos de corte. Para eso, se usa la opción `breaks`.

2.2 Medidas características

R nos permite calcular fácilmente medidas características. En esta sección veremos como calcular la media, mediana, varianza, desviación típica y otras medidas resumen para variables cuantitativas.

2.2.1 Medidas de posición

R ofrece funciones para calcular medidas de tendencia central. Para calcular la **media** utilizamos la función `mean`.

```
> mean(iris$Sepal.Width)
```

```
## [1] 3.057333
```

La **mediana** se calcula en R con la función `median`.

```
> median(iris$Sepal.Width)
```

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

Los **cuantiles** se calculan en R con la función `quantile`. Por ejemplo:

```
> quantile(iris$Sepal.Width)
```

```
##   0%  25%  50%  75% 100%
##  2.0  2.8  3.0  3.3  4.4
```

Por defecto obtenemos el mínimo, el máximo, y los tres cuartiles. También podemos especificar los cuantiles que deseamos calcular. Por ejemplo, si queremos los deciles:

```
> pdec <- seq(0, 1, by = 0.1)
> quantile(iris$Sepal.Width, pdec)
```

```
##   0%  10%  20%  30%  40%  50%  60%  70%  80%  90% 100%
## 2.00 2.50 2.70 2.80 3.00 3.00 3.10 3.20 3.40 3.61 4.40
```

2.2.2 Medidas de dispersión

Una de las formas más sencillas de medir dispersión es con el **rango**. La función `range` devuelve el menor y mayor valor de un conjunto de datos. La diferencia entre ellos se calcula con la función `diff` como se muestra a continuación.

```
> range(iris$Sepal.Width)  # Mínimo y máximo
```

```
## [1] 2.0 4.4
```

```
> diff(range(iris$Sepal.Width))  # Máximo - mínimo
```

```
## [1] 2.4
```

El **rango intercuartílico** (IQR) es la distancia entre el percentil 75 y el percentil 25. Se puede calcular con la función `IQR` o usando la función `quantile` como se explicó con anterioridad.

```
> IQR(iris$Sepal.Width)
```

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

La función `var` calcula la **varianza** de un conjunto de datos a partir de la fórmula:

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

```
> var(iris$Sepal.Width)
```

```
## [1] 0.1899794
```

La **desviación típica** se calcula como la raíz cuadrada de la varianza. Se puede calcular con la función `sd`.

2.2.3 Medidas de forma

Para obtener medidas de forma, necesitamos la librería moments:

```
> library(moments)
> skewness(iris$Sepal.Width)
```

```
## [1] 0.3157671
```

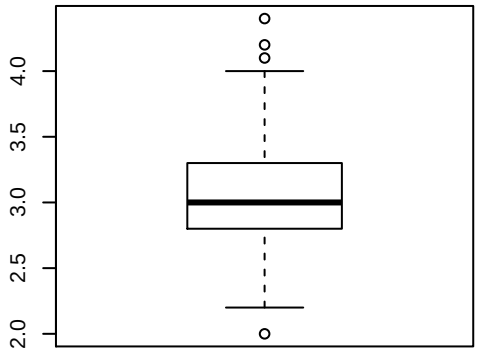
```
> kurtosis(iris$Sepal.Width)
```

```
## [1] 3.180976
```

2.3 El diagrama de cajas

El **diagrama de cajas** (boxplot) es una representación gráfica de variables continuas muy sencilla y útil en la que se visualizan, entre otros, los cuartiles y los valores atípicos. Representamos un boxplot en R con la función `boxplot`.

```
> boxplot(iris$Sepal.Width)
```



El diagrama de caja consta de una caja central que está delimitada por la posición de los cuartiles Q_3 y Q_1. Dentro de esa caja se dibuja la línea que representa la mediana. De los extremos de la caja salen unas líneas que se extienden hasta los puntos $LI = \max\{\min(x_i), Q_1 - 1.5\,(RI)\}$ y $LS = \min\{\max(x_i), Q_3 + 1.5\,(RI)\}$ que representarían el rango razonable hasta el cual se pueden encontrar datos. Los datos que caen fuera del intervalo (LI, LS) se consideran datos atípicos y se representan individualmente.

Cuando trabajamos con datos que se pueden agrupar por grupos, resulta interesante obtener información resumida y medidas características para los distintos grupos. Por ejemplo, supongamos que queremos calcular la longitud media de sépalo para cada especie. Debemos entonces considerar la variable cuantitativa `Sepal.Length` agrupada por `Species`. Para ello, podemos utilizar la función `tapply`. Por ejemplo,

```
> tapply(iris$Sepal.Length, iris$Species, mean)
```

```
##     setosa versicolor  virginica
##      5.006      5.936      6.588
```

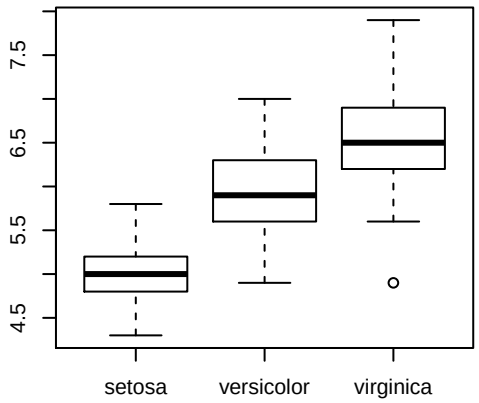
La función `tapply` toma los valores de `Sepal.Length`, los separa en función de `Species` y calcula la media (`mean`) de cada grupo. De la misma manera podemos calcular, por ejemplo, los cuantiles:

```
> tapply(iris$Sepal.Length, iris$Species, quantile)
```

```
## $setosa
##   0%  25%  50%  75% 100%
##  4.3  4.8  5.0  5.2  5.8
##
## $versicolor
##   0%  25%  50%  75% 100%
##  4.9  5.6  5.9  6.3  7.0
##
## $virginica
##    0%   25%   50%   75%  100%
## 4.900 6.225 6.500 6.900 7.900
```

Los boxplot también son útiles para entender diferencias entre grupos. Para obtener un boxplot de la longitud de sépalo para cada especie escribiremos:

```
> boxplot(iris$Sepal.Length ~ iris$Species)
```



3 Ejercicios de repaso

El fichero titanic_es.csv contiene información sobre aproximadamente el 80% de los pasajeros del Titanic[1]. Las variables incluidas en el fichero son:

- clase: clase en la que viajaba el pasajero (primera = Primera clase; segunda = Segunda clase; tercera = Tercera clase).
- sobreviviente: indica si cada pasajero sobrevivió o no al naufragio (0 = No; 1 = Sí).
- sexo: (hombre o mujer).
- edad: edad en años (los menores de un año se representan con un número menor uno).
- tarifa: precio del billete en libras.
- embarque: puerto en el que se realizó el embarque (Cherbourg, Queenstown o Southampton).

Importa los datos en R y almacena el contenido en un objeto de nombre `titanic`. Contesta a las siguientes preguntas utilizando R:

1. ¿Cuántos pasajeros aparecen registrados en el conjunto de datos?
2. ¿Cuántos pasajeros sobrevivieron al naufragio?
3. Calcula el porcentaje de pasajeros que sobrevivió al naufragio.
4. ¿Cuántos pasajeros viajaban en primera clase?
5. ¿Cuántos niños (menores de 12 años) aparecen registrados?
6. Calcula las frecuencias relativas para las distintas clases de pasajeros.

[1] http://www.encyclopedia-titanica.org/

7. Representa con un gráfico adecuado la distribución de las distintas clases de pasajeros.
8. Representa con un gráfico adecuado la distribución de la edad de los pasajeros registrados.
9. ¿Cuál era la edad media de los pasajeros? ¿Y la mediana?
10. ¿Cuál era el precio medio de un pasaje en el Titanic?
11. La frase "Mujeres y niños primero" hace referencia a un protocolo histórico por el que las mujeres y los niños debían ser los primeros en ser salvados en una emergencia. Según los datos, ¿qué porcentaje de mujeres sobrevivió al naufragio? ¿qué porcentaje de hombres sobrevivió al naufragio? ¿qué porcentaje de niños (menores de 12 años) sobrevivió al naufragio?
12. ¿Es la tasa de supervivencia de los hombres que viajaban en primera clase superior a la de los hombres que viajaban en segunda? ¿Es superior a las de los hombres que viajaban en tercera clase?
13. Representa mediante un diagrama de cajas las edades de los pasajeros agrupadas por clase. ¿Qué observas?
14. Calcula la media y la varianza de las edades en cada clase.
15. En el barco se proporcionaba asistencia especial al 5% de los pasajeros de mayor edad. ¿Qué edad debía tener un pasajero para recibir asistencia especial? ¿Qué edad tenía el pasajero más mayor?
16. Dibuja un histograma para el precio del pasaje en primera clase.
17. ¿Cuál es la moda para el puerto de embarque?
18. Los pasajeros que pagaban menos por su billete debían dormir en los camarotes más modestos del barco. En concreto, el 5% de los pasajeros que menos pagaban viajaban en dichos camarotes. ¿Cuál era el precio máximo que se pagaba por uno de estos camarotes?
19. Construye una tabla de contingencia para la distribución de la clase del pasajero por sexo.
20. Representa en un mismo gráfico la distribución de la clase del pasajero por sexo.
21. Calcula los cuartiles para la edad.
22. ¿Pagaban billete los menores de un año?

Referencias

Tukey, J.W. 1977. *Exploratory Data Analysis*. Addison-Wesley Series in Behavioral Science. Addison-Wesley Publishing Company. https://books.google.es/books?id=UT9dAAAAIAAJ.