# Tutorial 04: Variables aleatorias.

Atención:

- Este documento pdf lleva adjuntos algunos de los ficheros de datos necesarios. Y está pensado para trabajar con él directamente en tu ordenador. Al usarlo en la pantalla, si es necesario, puedes aumentar alguna de las figuras para ver los detalles. Antes de imprimirlo, piensa si es necesario. Los árboles y nosotros te lo agradeceremos.
- Fecha: 19 de abril de 2017. Si este fichero tiene más de un año, puede resultar obsoleto. Busca si existe una versión más reciente.

# Índice



# 1. Variables aleatorias discretas con R.

## 1.1. Tabla (función) de densidad de una variable aleatoria discreta en R

Una variable aleatoria discreta $X$ que toma los valores

$$x_1, x_2, \ldots, x_k$$

se define mediante su tabla de densidad de probabilidad:

| *Valor:* | $x_1$ | $x_2$ | $x_3$ | $\cdots$ | $x_k$ |
|---|---|---|---|---|---|
| *Probabilidad:* | $p_1$ | $p_2$ | $p_3$ | $\cdots$ | $p_k$ |

Como ya hemos dicho, las probabilidades se pueden entender, en muchos casos, como una versión teórica de las frecuencias relativas. Así que esta tabla se parece mucho a una tabla de frecuencias relativas, y parte de las operaciones que vamos a hacer nos recordarán mucho a las que hicimos en la primera parte del curso usando tablas de frecuencias.

La primera variable aleatoria que vamos a estudiar va a ser, como en el Ejemplo 4.1.1 del libro (pág. 97), la variable $X$ cuyo valor es el resultado de sumar los puntos obtenidos al lanzar dos dados. Para estuidarla, vamos a recordar algunas de las técnicas de simulación que aprendimos en el Tutorial03. Usaremos matrices de R para reproducir los resultados de ese Ejemplo 4.1.1. Concretamente, empezamos usando una matriz para representar, por columnas, los resultados posibles al tirar el primer dado:

```
(dado1 = matrix(rep(1:6,6), nrow = 6))
```

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

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

Ahora hacemos lo mismo para el segundo dado, pero en este caso por filas:

```
(dado2 = matrix(rep(1:6,6), nrow = 6, byrow = TRUE))

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

La matriz con la tabla de las sumas posibles se obtiene entonces así:

```
dado1 + dado2

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

Y podemos hacer una tabla de frecuencias de los resultados fácilmente (la primera fila es el valor de la suma y el segundo la frecuencia de ese valor):

```
table(dado1 + dado2)

##
##  2  3  4  5  6  7  8  9 10 11 12
##  1  2  3  4  5  6  5  4  3  2  1
```

Si queremos convertir estos valores en probabilidades tenemos que dividirlos por el número de resultados posibles, que son 36. Recuerda que R siempre devuelve respuestas numéricas (no *simbólicas*):

```
table(dado1 + dado2) / 36

##
##      2      3      4      5      6      7      8      9     10     11
## 0.0278 0.0556 0.0833 0.1111 0.1389 0.1667 0.1389 0.1111 0.0833 0.0556
##     12
## 0.0278
```

Estos resultados son (las versiones numéricas de) los mismos que aparecen en la Tabla 4.1 del libro (pág. 97). Pero lo que queremos ahora es una simulación de esos valores, para compararlo con lo que predice la teoría (la variable $X$).

**Ejercicio 1.**
*Este ejercicio debería resultar sencillo, después del trabajo del Tutorial03. Lo que queremos es simular* $n = 1000000$ *tiradas de dos dados, y calcular la tabla de frecuencias relativas de la variable*

$$X = \{\textit{suma de los dos dados}\}.$$

*Solución en la página 25.* □

## 1.2. Media, varianza y desviación típica.

En este apartado vamos a ocuparnos de los cálculos necesarios para trabajar con una variable aleatoria discreta $X$, dada mediante una tabla de valores y probabilidades (la tabla de densidad de probabilidad) como esta:

| Valor | $x_1$ | $x_2$ | $\cdots$ | $x_k$ |
|---|---|---|---|---|
| Probabilidad | $p_1$ | $p_2$ | $\cdots$ | $p_k$ |

La teoría correspondiente se encuentra en el Capitulo 4 del libro. A partir de la información de esta tabla, queremos calcular la media $\mu_X$ de $X$ y la varianza $\sigma_X^2$ de $X$. Vamos a aprender a utilizar R para calcularlos.

Para fijar ideas vamos a pensar en un ejemplo concreto. Supongamos que la densidad de probabilidad de la variable $X$ es esta:

| Valor | 2 | 4 | 7 | 8 | 11 |
|---|---|---|---|---|---|
| Probabilidad | 1/5 | 1/10 | 1/10 | 2/5 | 1/5 |

De momento, y puesto que en ejemplos como este estamos trabajando con variables aleatorias con muy pocos valores, nos vamos a conformar con almacenar los valores y las probabilidades, por separado, en sendos vectores de R.

```
valores = c(2,4,7,8,11)
probabilidades = c(1/5,1/10,1/10,2/5,1/5)
```

Y ahora, para calcular en R la media, haremos:

```
(media = sum(valores * probabilidades) )

## [1] 6.9
```

mientras que la varianza y desviación típica se obtienen con:

```
(varianza = sum((valores - media)^2 * probabilidades) )

## [1] 9.49

(sigma = sqrt(varianza) )

## [1] 3.08
```

**Ejercicio 2.**

1. *Comprueba, usando R, los resultados de los Ejemplos 4.2.2 y 4.2.6 del libro (págs. 105 y 108, respectivamente), en lo que se refiere a la variable $X$, suma de dos dados.*
2. *Usando R habrás obtenido un valor numérico (aproximado) de la varianza de $X$. Usa un programa simbólico (Wiris o Wolfram Alpha, por ejemplo) para calcular el valor exacto de la varianza.*
3. *Repite los apartados anteriores para la variable $Y$, la diferencia (en valor absoluto) de los dos dados.*

*Solución en la página 26.* □

## 1.3. Operaciones con variables aleatorias.

En el Ejercicio 2 acabamos de calcular la media y varianza de las variables $X$ e $Y$, que representan la suma y diferencia de los resultados al lanzar dos dados, respectivamente. Vamos a usar estas

dos variables para experimentar con los resultados teóricos que aparecen en la Sección 4.3 del libro (pág. 109).

Es importante recordar siempre que las variables aleatorias son modelos *teóricos* de las asignaciones de probabilidad. La media $\mu_X$ de la variable aleatoria $X$ representa el valor medio esperado en una serie muy larga de repeticiones del suceso aleatorio que representa la variable $X$. Por tanto, la media sirve para hacer predicciones *teóricas* y, en cada casos concreto, los valores que obtendremos serán *parecidos, pero no idénticos* al valor que predice la media.

Para ilustrar esto, vamos a tomar como punto de partida la variable $X$ (suma al lanzar dos dados), y definiremos una nueva variable:

$$W = 3 \cdot X - 4.$$

La teoría predice que ha de ser:

$$E(W) = E(3 \cdot X - 4) = 3 \cdot E(X) - 4$$

y, usando los resultados del Ejercicio 2 de este tutorial, tenemos

$$E(W) = 3 \cdot E(X) - 4 = 3 \cdot 7 - 4 = 17.$$

Para *"comprobar experimentalmente"* esta predicción teórica vamos a fabricar una serie muy larga ($n = 10000$) de valores aleatorios de $W$, y calcularemos su media. Los valores de $W$ se obtienen de los de $X$ con este código en R (la primera parte es muy parecida al comienzo de la solución del Ejercicio 1):

```
set.seed(2014)
n = 10000
dado1 = sample(1:6, n, replace=TRUE)
dado2 = sample(1:6, n, replace=TRUE)
X = dado1 + dado2
W = 3 * X - 4
```

La novedad, naturalmente, es esa última línea, en la que calculamos los valores de $W$ a partir de los de $X$. La media de esos 10000 valores de $W$ es:

```
mean(W)

## [1] 16.9
```

Y, como puedes ver, el resultado del experimento se parece mucho a nuestra predicción teórica.

Vamos a aprovechar, también, para comprobar que las cosas no siempre son tan sencillas. En particular, vamos a usar la variable

$$V = X^2$$

para comprobar que:

$$E(V) = E(X^2) \neq (E(X))^2 = 7^2 = 49.$$

Aquí, de nuevo, $X$ es la variable suma al lanzar dos dados. Para comprobar *experimentalmente* esto procedemos de forma muy parecida a lo que acabamos de hacer con $W$. Generamos una lista de valores de $V$, y calculamos su media:

```
V = X^2
mean(V)

## [1] 54.6
```

¿Cuál es el cálculo teórico correspondiente? Para calcular la media de $V = X^2$ empezamos por hacer la tabla de densidad de esta variable. Esa tabla se obtiene fácilmente de la Tabla 4.2.2 del libro (pág. 105), elevando los valores al cuadrado (las probabilidades se mantienen):

| Valor | 4 | 9 | 16 | 25 | 36 | 49 | 64 | 81 | 100 | 121 | 144 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Probabilidad | $\frac{1}{36}$ | $\frac{2}{36}$ | $\frac{3}{36}$ | $\frac{4}{36}$ | $\frac{5}{36}$ | $\frac{6}{36}$ | $\frac{5}{36}$ | $\frac{4}{36}$ | $\frac{3}{36}$ | $\frac{2}{36}$ | $\frac{1}{36}$ |

Y ahora puedes usar cualquier programa (Wolfram Alpha, o el propio R) para comprobar que:

$$\mu_V = \frac{1974}{36} \approx 54.83.$$

Fíjate en que este valor se parece mucho más al que hemos obtenido en la versión experimental del cálculo, y que era 54.6.

Los resultados que acabamos de obtener confirman que la media no se lleva bien con el cuadrado: la media del cuadrado no es el *"cuadrado de la media"*. De hecho, la diferencia entre esas dos cantidades es, precisamente, la varianza:

$$\mathrm{Var}(X) = E(X^2) - (E(X))^2 .$$

Sin entrar a dar una demostración teórica de este hecho, vamos a comprobarlo usando la variable $X$. Empezamos repitiendo los mismos cálculos que aparecen en la solución del Ejercicio 2 (ver página 26).

```
valoresX = 2:12
probabilidadesX = c(1:6,5:1) / 36
(muX = sum(valoresX * probabilidadesX))

## [1] 7

(varianzaX = sum((valoresX - muX)^2 * probabilidadesX) )

## [1] 5.83
```

Recuerda que el cálculo que estamos haciendo aquí es teórico (no es un "experimento"). Ahora vamos a calcular la media de $V = X^2$:

```
(valoresV = valoresX^2)

##  [1]   4   9  16  25  36  49  64  81 100 121 144

(probabilidadesV = probabilidadesX)

##  [1] 0.0278 0.0556 0.0833 0.1111 0.1389 0.1667 0.1389 0.1111 0.0833 0.0556
## [11] 0.0278

(muV = sum(valoresV * probabilidadesV))

## [1] 54.8
```

Y ahora podemos comprobar, en este ejemplo, la identidad $\mathrm{Var}(X) = E(X^2) - (E(X))^2$. Se tiene:

```
varianzaX

## [1] 5.83

muV - (muX)^2

## [1] 5.83
```

como esperábamos. Naturalmente, este resultado teórico también se puede comprobar experimentalmente. Y es interesante hacerlo, así que lo exploraremos en los ejercicios adicionales.

## 1.4. Función de distribución (probabilidad acumulada)

La función de distribución de la variable aleatoria $X$ es, recordémoslo:

$$F(k) = P(X \leq k)$$

Es decir, que dado el valor de $k$, debemos sumar todos los valores de la densidad de probabilidad para valores menores o iguales que $k$. En el ejemplo de la variable $X$ que aparece al comienzo de la Sección 1.2 (pág. 3), si queremos calcular $F(7)$, debemos hacer: $F(7) = P(X = 2) + P(X = 4) + P(X = 7) = \frac{1}{5} + \frac{1}{10} + \frac{1}{10}$. Se trata de sumas acumuladas, como las que vimos en el caso de las tabla de frecuencias acumuladas. Así que usaremos la función `cumsum`, que ya encontramos en el Tutorial02 (Sección 4.2). Es decir,

```
cumsum(probabilidades)

## [1] 0.2 0.3 0.4 0.8 1.0
```

que, como ves, produce un vector con los valores de $F(k)$ para cada $k$. Seguramente preferiremos ver estos resultados en forma de tabla, para poder localizar fácilmente el valor de $F$ que corresponde a cada valor de $k$. Con lo que hemos aprendido de matrices, es fácil conseguirlo. Pondríamos:

```
rbind(valores,cumsum(probabilidades))

##         [,1] [,2] [,3] [,4] [,5]
## valores  2.0  4.0  7.0  8.0   11
##          0.2  0.3  0.4  0.8    1
```

Aunque en esta tabla sólo aparecen los valores $2, 4, 7, 8$ y $11$, es bueno recordar que la función de distribución $F(x)$ está definida *sea cual sea el valor de* $x$. Es decir, que tiene prefecto sentido preguntar, por ejemplo, cuánto vale $F(\frac{8}{3})$. Hay una forma de conseguir que R conteste esta pregunta automáticamente, pero todavía tenemos que aprender algunas cosas más antes de ver cómo podemos conseguir eso.

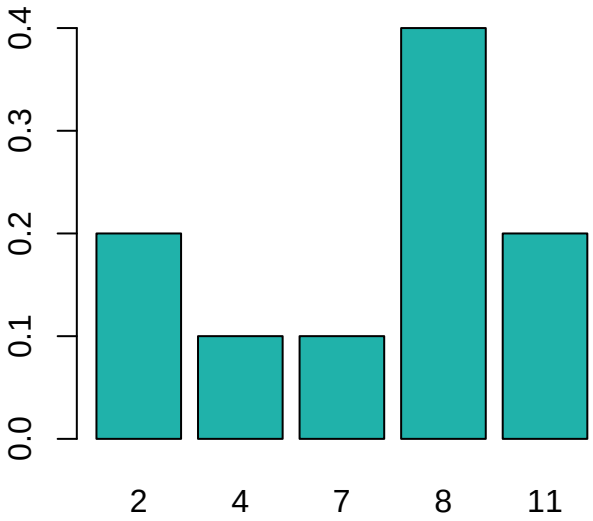
**Ejercicio 3.**
*¿Cuánto vale $F(\frac{8}{3})$? Aunque no forme parte del ejercicio, trata de ir pensando en un procedimiento que te permita, dado un valor $x$ cualquiera, obtener el valor $F(x)$.* *Solución en la página 27.* □

Todavía no disponemos de todas las herramientas necesarias para definir en R la función de distribución. Pero pronto aprenderemos lo necesario. Así que, como aperitivo, en el fichero de código que se incluye en la Sección hemos incorporado lo necesario para definir la función de distribución.

## 1.5. Representación gráfica de las variables aleatorias.

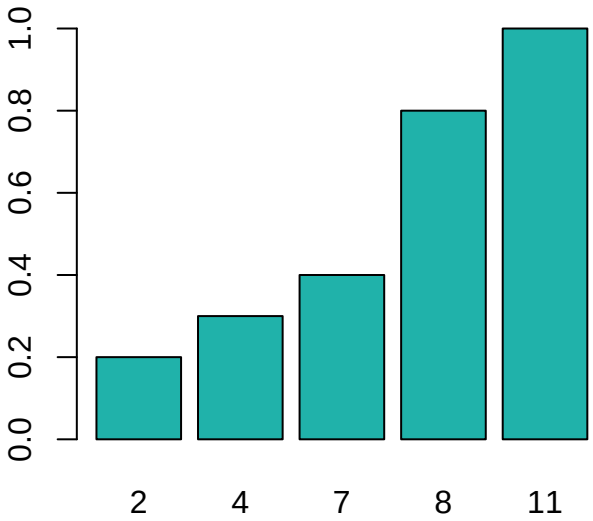
Dada una variable aleatoria $X$, por ejemplo la que venimos usando desde el comienzo de la Sección 1.2, podemos representar gráficamente su tabla de densidad de probabilidad, en un diagrama de columnas, mediante este comando:

```
barplot(probabilidades, names.arg=valores, col="lightseagreen")
```




Si lo que queremos es ver su función de distribución, podríamos pensar en acumular esas probabilidades:

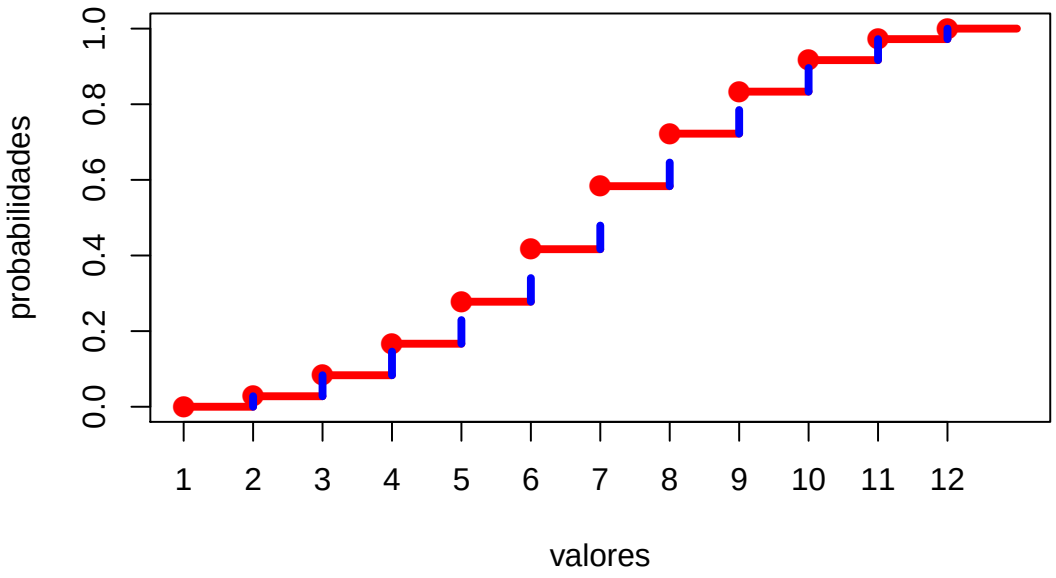
```
barplot(cumsum(probabilidades), names.arg=valores, col="lightseagreen")
```




Pero este gráfico, como ves, tiene muchas limitaciones y no es especialmente informativo. Para representar la función de distribución es más común utilizar *gráficos de escalera* como el que aparece en la Figura 4.3 del libro (página 113). En R hay varias maneras de hacerlo, más o menos complicadas. Aquí vamos a limitarnos a incluir, sin demasiadas explicaciones, un fragmento de código que produce ese tipo de gráficos a partir de los vectores de valores y probabilidades. Para mostrar su funcionamiento usamos la misma variable $X$ que venimos utilizando para los anteriores gráficos. Si deseas usar el código con otra variable sólo tienes que descomentar y cambiar las dos

primeras líneas, en las que se definen los valores y probabilidades. El resto no necesita modificación. A medida que aprendamos más sobre R, este código quedará más claro.

```
# valoresX = 2:12
# probabilidadesX = c(1:6,5:1) / 36

##  NO MODIFIQUES EL RESTO DEL CODIGO ##

graficoEscalera = function(valores, probabilidades ){
  probabilidades = probabilidades/sum(probabilidades)
  y0 = c(0,cumsum(probabilidades), 1)
  x0 = c(min(valores)-1,valores, max(valores)+1)
  xA = x0[-length(x0)]
  xB = x0[-1]
  yA = y0[-length(y0)]
  plot(x0, y0, type="n", xlab="valores", ylab="probabilidades", las=3, xaxt="n")
  segments(xA,yA,xB,yA, lwd=4, col="red")
  points(xA, yA, pch=16, col="red", cex=1.5)
  axis(1, at=xA,   labels=xA)
  segments(valores, yA[-length(yA)] , valores, yA[-1], lty=2, col="blue", lwd=4)
}
graficoEscalera(valoresX, probabilidadesX)
```




El resultado, como ves, es bastante más satisfactorio.

## 1.6. Un fichero de comandos R para estudiar una variable discreta.

Al igual que hicimos en el Tutorial02, en el que incluimos un fichero que resumía muchos comandos de Estadística Descriptiva, vamos a incluir aquí el fichero

Tut04-AnalisisVariableAleatoriaDiscreta.R

que reúne los comandos que hemos ido viendo en este Tutorial para trabajar con una variable aleatoria discreta (con un número finito de valores) definida mediante su tabla de densidad.

## 2. Ficheros de datos en R: objetos de tipo `data.frame`.

Vamos a aprovechar las próximas secciones de este tutorial para mejorar nuestras habilidades como usuarios de R. Uno de los objetivos de este curso, en esta vertiente más aplicada de los Tutoriales, es poner al lector en contacto con algunos de los problemas más prácticos que se va a encontrar al trabajar con datos. En particular, en algunas ocasiones,

- vamos a trabajar con algunos ficheros de datos *muy grandes*,
- que contendrán muchos datos que no vamos a utilizar directamente, de manera que habrá que *seleccionar* una parte de los datos,
- esos datos no estarán siempre en el formato más cómodo, o más conveniente, así que habrá que *transformarlos*.

En el anterior Tutorial03 hemos aprendido el manejo básico de las matrices en R. Las matrices, en comparación con los vectores, mejoran nuestra capacidad de trabajar con conjuntos de datos más complicados. Pero para poder afrontar las situaciones a las que nos referíamos más arriba, todavía debemos aprender a superar algunas dificultades. En primer lugar, no hemos aprendido a leer y escribir las matrices usando ficheros (de tipo `csv`, por ejemplo). En el Tutorial02 hemos usado las funciones `scan` y `cat` para leer y escribir, respectivamente, vectores usando ficheros `csv`. En este tutorial vamos a aprender a usar las dos funciones de R que se emplean más a menudo para leer ficheros de datos, y que son `write.table` y `read.table`.

Aparte de este problema de lectura/escritura de datos, tenemos que aprender a trabajar en R con conjuntos de datos heterogéneos. Para entender a qué nos referimos con esto, vamos a volver a pensar en el fichero

Tut01-PracticaConCalc.csv,

que utilizamos en el Tutorial01, y que incluía un conjunto de datos con tres columnas con 1300 valores de cada una de las variables `var1`, `var2` y `var3`. En la siguiente figura se muestra el comienzo de aquel fichero, abierto con un editor de texto. Lo más importante es observar que las variables

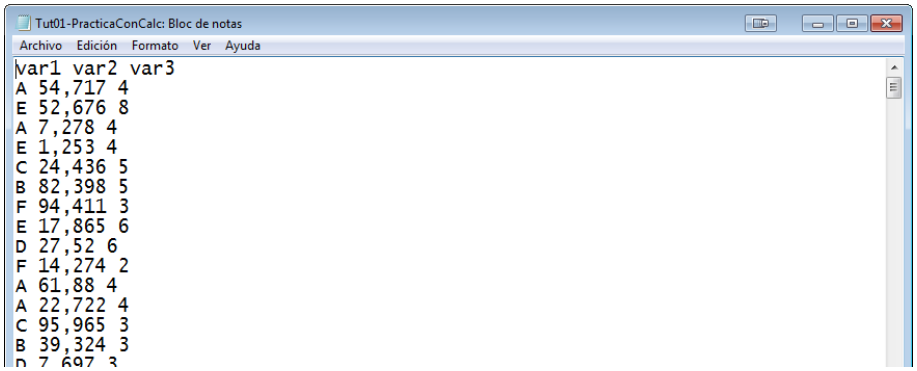



Figura 1: Contenido del fichero `Tut01-PracticaConCalc.csv`

que ocupan las columnas de esa tabla son cada una de un tipo distinto: `var1` es una variable de tipo `character` (en el lenguaje de tipos de datos de R; nosotros diríamos que es un factor), `var2` es de tipo `numeric` (una variable cuantitativa continua) y, finalmente, `var3` es de tipo `integer` (una variable cuantitativa discreta). Y además, **esto es esencial**, no queremos separar las columnas para trabajar con ellas por separado, porque los valores de cada fila están relacionados entre sí. Muy a menudo, por ejemplo, cada fila corresponde a una misma observación en la que, para un individuo dado de la población, se han observado los valores de varias variables en ese individuo (por ejemplo, para una misma persona hemos medido su presión arterial, su edad, su peso, etc.)

Para trabajar con este tipo de conjuntos de datos R nos ofrece un tipo de objetos, llamado `data.frame`. A riesgo de ser pesados, la diferencia básica entre una matriz y un `data.frame` es esta: una matriz (tipo `matrix`) contiene datos en filas y columnas, pero *todos del mismo tipo*: todos `numeric` o todos `character`, pero no se admiten mezclas. Y en cambio el `data.frame` permite mezclar datos de distintos tipos.

La "limitación", en el caso del `data.frame`, es que los datos de una misma *columna* tienen que ser todos del mismo tipo. Pero eso sólo es una limitación a medias, porque, de hecho, vamos a insistir varias veces a lo largo del curso en que esa es la forma adecuada de organizar nuestros datos: cada columna corresponde a una variable, y por lo tanto sus valores son todos de un mismo tipo, el de esa variable (cada fila, por su parte, corresponde a una observación de todas esas variables en un individuo de la población).

## 2.1. Operaciones con `data.frames`.

### 2.1.1. Creando un `data.frame` a partir de vectores.

Podemos crear un `data.frame` a partir de unos cuantos vectores que, eso sí, deben ser de la misma longitud. Cada uno de los vectores corresponderá a una columna del `data.frame`. Para crear nuestro primer ejemplo de `data.frame`, empieza por ejecutar estos comandos:

```
    nombres = c("Io", "Europa", "Ganimedes", "Calisto" )
    class(nombres)
```

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

```
    diametrosKm = c(3643, 3122, 5262, 4821)
    class(diametrosKm)
```

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

Para más información, ver el enlace http://es.wikipedia.org/wiki/Satélite_galileano.

Hemos usado la función `class` para enfatizar de nuevo el hecho de que los vectores corresponden a tipos distintos de datos. Ahora podemos crear el `data.frame` a partir de esos dos vectores (con los paréntesis exteriores mostraremos la respuesta; R normalmente no lo haría, como sabes):

```
(satelitesGalileanos = data.frame(nombres, diametrosKm) )
```

```
##     nombres diametrosKm
## 1        Io        3643
## 2    Europa        3122
## 3 Ganimedes        5262
## 4   Calisto        4821
```

La respuesta que muestra R nos recuerda (y es, a la vez, distinta) a la que obtenemos al trabajar con una matriz. Los dos tipos de objetos son tabulares, y buena parte de los trucos que hemos aprendido para matrices se aplican también a los `data.frame`. Por ejemplo, vamos a añadir a nuestros datos una columna adicional con el periodo orbital (en días) de cada uno de los satélites galileanos de Jupiter. Esos datos están almacenados en el vector:

```
periodoOrbital = c(1.77, 3.55, 7.16, 16.69)
```

Y para añadirlos como columna al `data.frame` podemos recurrir a la función `cbind` que ya conocemos de las matrices:

```
(satelitesGalileanos = cbind(satelitesGalileanos, periodoOrbital) )
```

```
##     nombres diametrosKm periodoOrbital
## 1        Io        3643          1.77
## 2    Europa        3122          3.55
## 3 Ganimedes        5262          7.16
## 4   Calisto        4821         16.69
```

La notación de corchetes, que usamos con vectores y matrices, también sirve en este caso. Algunos ejemplos:

```
satelitesGalileanos[3,2]

## [1] 5262

satelitesGalileanos[1, ]

##   nombres diametrosKm periodoOrbital
## 1      Io        3643           1.77

satelitesGalileanos[c(1,4), c(1,3)]

##   nombres periodoOrbital
## 1      Io           1.77
## 4 Calisto          16.69
```

Asegúrate antes de seguir de que has entendido por qué se obtiene esa respuesta en cada uno de los ejemplos. Fíjate, además, en que hemos usado el mismo nombre para el `data.frame` modificado al añadir esa columna. También es posible seleccionar elementos del `data.frame` mediante condiciones. Por ejemplo, imagínate que queremos seleccionar los satélites galileanos cuyo diámetro supera los 4000 kilómetros. Podmeos hacerlo así:

```
satelitesGalileanos[ (satelitesGalileanos[ ,2] > 4000) , ]

##     nombres diametrosKm periodoOrbital
## 3 Ganimedes        5262           7.16
## 4   Calisto        4821          16.69
```

Hemos rodeado con paréntesis la condición para ayudarte a ver la estructura de este comando. Lo que estamos haciendo se puede traducir a palabras así: "muéstrame las filas de `satelitesGalileanos` tales que la segunda columna sea mayor que 4000". Y para asegurarnos de que entiendes esa condición entre paréntesis vamos a analizarla con un poco más de detalle.

**Ejercicio 4.**
*Ejecuta la parte del anterior comando que define la condición,*

```
(satelitesGalileanos[ ,2] > 4000)
```

*e interpreta el resultado. Solución en la página 27.* □

### 2.1.2. Funciones `read.table` y `write.table`.

Esta forma de crear un `data.frame`, a partir de vectores, no resuelve nuestro problema inicial de esta sección, el de leer los datos del fichero `Tut01-PracticaConCalc.csv`. Ahora ya sabemos que, una vez leídos, los almacenaremos en un `data.frame`, pero ¿cómo conseguimos que pasen del fichero a ese `data.frame`? Pues usando (entre otras posibilidades) la función `read.table`. Para verla en acción, asegúrate de que has seleccionado como directorio de trabajo (recuerda, menú `Session` de RStudio) la carpeta que contiene el fichero `Tut01-PracticaConCalc.csv`, y ejecuta estos comandos, que explicaremos detenidamente a continuación:

```
datosTutorial01 =
    read.table(file="../datos/Tut01-PracticaConCalc.csv", header=TRUE, sep=" ", dec=",")
class(datosTutorial01)

## [1] "data.frame"

head(datosTutorial01)

##   var1  var2 var3
```

```
## 1    A 54.72    4
## 2    E 52.68    8
## 3    A  7.28    4
## 4    E  1.25    4
## 5    C 24.44    5
## 6    B 82.40    5
```

La función `read.table` se encarga de leer el fichero `csv` y guardar su contenido en un `data.frame`. Enseguida volvemos sobre los argumentos de `read.table`.Pero antes, fíjate en que el resultado de `class` muestra que `datosTutorial01` es, de hecho, un data.frame. Hemos visto anteriormente que el comando `head` se puede usar para mostrar el comienzo de un vector, mientras que `tail` muestra el final. Ambos comandos se pueden usar igualmente con `data.frames`, para obtener, respectivamente, las primeras o últimas filas del `data.frame` (que tiene 1300 filas).

Veamos ahora los argumentos de `read.table`:

- El primero, `file`, apenas necesita explicación. Asegúrate, eso sí, de que el fichero que vas a usar está en tu directorio de trabajo. Si es así, tras escribir `file=` en RStudio basta con que pulses el tabulador para ahorrarte errores y trabajo.
- La opción `header` nos permite decirle a R si el fichero `csv` incluye una primera fila en la que aparecen los nombres de las variables (usamos `TRUE` en este caso) o no (y usamos `FALSE`). Si el fichero no incluye esa línea, R pondrá nombres a las variables por nosotros, de forma automática, y no siempre nos gustará el resultado (aunque siempre podemos ponerlos sobre la marcha con la opción `col.names`).
- La opción `sep` le dice a R como están separados (con comas, espacios, etc.) los valores de las distintas variables, dentro de cada fila del fichero `csv`.
- Finalmente (para este ejemplo), la opción `dec` nos permite indicarle a R si se usan puntos o comas para separar los decimales. El programa R siempre trabajará, en sus operaciones internas, con el punto como separador, pero gracias a esta opción resulta fácil leer ficheros de datos en el formato (desgraciadamente, todavía) habitual en España y otros países.

En resumen, ya hemos leído el fichero `csv` llamado `Tut01-PracticaConCalc.csv`, lo hemos guardado en un `data.frame` llamado `datosTutorial01`, y seguramente, podemos ponernos a hacer cosas con las variables `var1, var2, var3` que corresponden a las columnas de ese fichero. Vamos a tratar de calcular, por ejemplo, la media de `var3`:

```
mean(var3)
```

```
## Error in mean(var3): objeto 'var3' no encontrado
```

La respuesta de R, en color rojo, indica que se ha enfadado con nosotros. ¿Por qué? Pues porque las variables de un `data.frame` no viven vidas propias. El nombre correcto de esta variable no es `var3` sino `datosTutorial01$var3`.

Y lo mismo sucede con `var1` y `var2`, claro. Si pruebas con ese nombre, el cálculo de la media funcionará sin problemas:

```
mean(datosTutorial01$var3)
```

```
## [1] 5.04
```

"Entonces, cada vez que quiera referirme a esta variable, ¿tengo que escribir `datosTutorial01$var3`?" La respuesta corta es: sí. La respuesta larga es que,

(a) por un lado, así evitamos confusiones entre variables con el mismo nombre, pero procedentes de distintos `data.frames`,

(b) usando el tabulador en RStudio, este problema se alivia mucho. Tras escribir sólo las tres letras `dat`, si pulso el tabulador aparece el nombre del `data.frame`. Y si acepto ese nombre,

y a continuación escribo `$` (de manera que tengo escrito `datosTutorial01$`), entonces una nueva pulsación del tabulador me permite acceder a la variable que deseo, fácilmente y sin errores.

(c) existen funciones (como `attach` o `with`) que nos permite aliviar este problema, *cuando sabemos bien lo que hacemos*. Más adelante veremos algunos ejemplos.

Antes de seguir adelante, vamos a pensar un momento en el proceso contrario, en el que tenemos un `data.frame` y queremos escribirlo en un fichero `csv`. La función correspondiente se llama, como era de esperar, `write.table`, y vamos a explorar su funcionamiento mediante algunos ejercicios.

**Ejercicio 5.**

1. *Vamos a fabricar un* `data.frame`*, con* 300 *filas y* 3 *columnas, usando tres vectores. El primero será un vector con las letras* `A`*,* `B` *y* `C`*, repetidas cada una* 100 *veces. Concretamente, las posiciones de la* 1 *a la* 100 *serán* `A`*, las* 100 *siguientes* `B` *y las* 100 *últimas* `C`*. El segundo vector estará formado por* 300 *números enteros, elegidos al azar entre* $-100$ *y* 100 *(usa* `set.seed(2014)` *para poder comparar tus soluciones con las nuestras). Para fabricar el tercer vector usa el comando* `rnorm(300)`*. Pronto aprenderemos su significado, pero te podemos adelantar que genera números reales al azar (pero no todos los valores son igual de probables). Cuando tengas los tres vectores úsalos para crear un* `data.frame` *llamado* `Tut04WriteTable`*, y comprueba su contenido con las funciones* `head` *y* `tail`*.*

```
##   vector1 vector2 vector3
## 1       A     -43 -0.0557
## 2       A     -67  0.7253
## 3       A      25  1.0051
## 4       A     -38  0.1155
## 5       A      10  0.2637
## 6       A     -83  0.9814
##     vector1 vector2 vector3
## 295       C      81   1.126
## 296       C     -67   0.383
## 297       C      38  -0.645
## 298       C     -71  -0.805
## 299       C      -1  -0.703
## 300       C      89   1.841
```

2. *Para guardar el* `data.frame` *en un fichero, llamado* `Tut04WriteTable.csv`*, asegúrate de que sabes cual es el directorio de trabajo (ejecuta* `getwd()` *si dudas), y usa la función* `write.table` *así:*

```
write.table(Tut04WriteTable, file="../datos/Tut04WriteTable.csv")
```

   *Después comprueba, usando un editor de texto (como el Bloc de Notas), que el contenido del fichero es correcto.*

3. *Abre el fichero* `csv` *que has creado con Calc, como aprendimos a hacer en el Tutorial00 (usa un espacio como separador), y calcula, con Calc, la media de la tercera columna de la tabla (el* `vector3` *que hemos creado en R). ¿Qué dificultades te encuentras?*

4. *Para soslayar esas dificultades vamos a ejecutar otra versión de la función* `write.table`*, que dará ocmo resultado un nuevo fichero* `csv` *(hemos añadido un* `2` *al nombre del fichero, para distinguirlos).*

```
write.table(Tut04WriteTable, file="../datos/Tut04WriteTable2.csv", dec = ",", row.names = FALSE)
```

   *Abre este nuevo fichero con Calc, y completa la tarea pendiente del apartado anterior. Comprueba con R el valor de la media.*

*Solución en la página 27.* □

## 2.2. Funciones para trabajar con `data.frames`.

El punto de partida de casi todo el trabajo que se hace en R es, a menudo, un `data.frame` (o algún tipo de objeto similar). Ese `data.frame` puede ser el resultado de leer un fichero con `read.table`. Otra posibilidad, que no vamos a explorar por el momento, es la de usar funciones de R para descargar los datos directamente desde internet, y guardar esos datos descargados en un `data.frame`. En una sesión típica de trabajo, una vez que tenemos un conjunto (o conjuntos) de datos almacenados en un `data.frame` (o en varios), a continuación realizamos algún tipo de análisis, más o menos complicado con esos datos, y guardamos el resultado en un nuevo `data.frame`, que a su vez puede exportarse a algún fichero (por ejemplo, guardando el resultado en un fichero `csv`). Conviene tener en cuenta, no obstante, que si guardamos los datos de partida y el fichero de comandos de R que hemos usado (¡bien comentado!), nuestro trabajo será, en gran medida, reproducible. Ya veremos algún ejemplo más detallado, más adelante en el curso.

Por el momento, lo que queremos transmitirle al lector es que aprender a manejar los `data.frame` de R, al menos de forma básica, es un requisito imprescindible para utilizar el programa de forma eficaz. Y por esa razón vamos a aprender algunas funciones adicionales, que hacen más cómodo nuestro trabajo con los `data.frames`.

Para empezar, las funciones `nrow` y `ncol` nos permiten obtener el número de filas y columnas de un `data.frame`.

```
nrow(satelitesGalileanos)

## [1] 4

ncol(satelitesGalileanos)

## [1] 3
```

En un `data.frame` tan pequeño esto puede parecer trivial, pero cuando trabajemos con miles de filas, y cientos de columnas, se hará inevitable.

Otra función útil es la función `colnames` que nos permite obtener, pero también modificar, los nombres de las columnas.

```
colnames(satelitesGalileanos)

## [1] "nombres"        "diametrosKm"    "periodoOrbital"
```

Fijate en el resultado de este comando, que mostramos usando `head`:

```
colnames(satelitesGalileanos) = c("varUno", "varDos", "varTres")
head(satelitesGalileanos)

##      varUno varDos varTres
## 1        Io   3643    1.77
## 2    Europa   3122    3.55
## 3 Ganimedes   5262    7.16
## 4   Calisto   4821   16.69
```

**Ejercicio 6.**

1. *Devuelve el* `data.frame` *a su estado anterior, y comprueba el resultado con* `head`.
2. *¿Qué resultado se obtiene al aplicar la función* `dim` *al* `data.frame`*?*

*Soluciones en la página 30.* □

La función `str` (puedes pensar que es una abreviatura de *structure*) es una función cuyo uso no se limita a los `data.frame`, y que nos proporciona información útil sobre los componentes del objeto de interés. Un par de ejemplos:

```
str(satelitesGalileanos)

## 'data.frame': 4 obs. of  3 variables:
##  $ varUno : Factor w/ 4 levels "Calisto","Europa",..: 4 2 3 1
##  $ varDos : num  3643 3122 5262 4821
##  $ varTres: num  1.77 3.55 7.16 16.69

str(Tut04WriteTable)

## 'data.frame': 300 obs. of  3 variables:
##  $ vector1: Factor w/ 3 levels "A","B","C": 1 1 1 1 1 1 1 1 1 1 ...
##  $ vector2: int  -43 -67 25 -38 10 -83 83 20 -81 -69 ...
##  $ vector3: num  -0.0557 0.7253 1.0051 0.1155 0.2637 ...
```

Como ves, el resultado es una descripción del conjunto de datos: qué variables lo forman y de qué tipo son, además del número de observaciones (filas) del conjunto de datos, y algunos valores iniciales de cada variable.

A lo largo del curso iremos conociendo más funciones que nos facilitarán el trabajo con `data.frames`, con el objetivo de ser capaces de seleccionar y transformar los datos, como decíamos al principio de esta sección.

## 2.3. Conversión entre tipos de datos

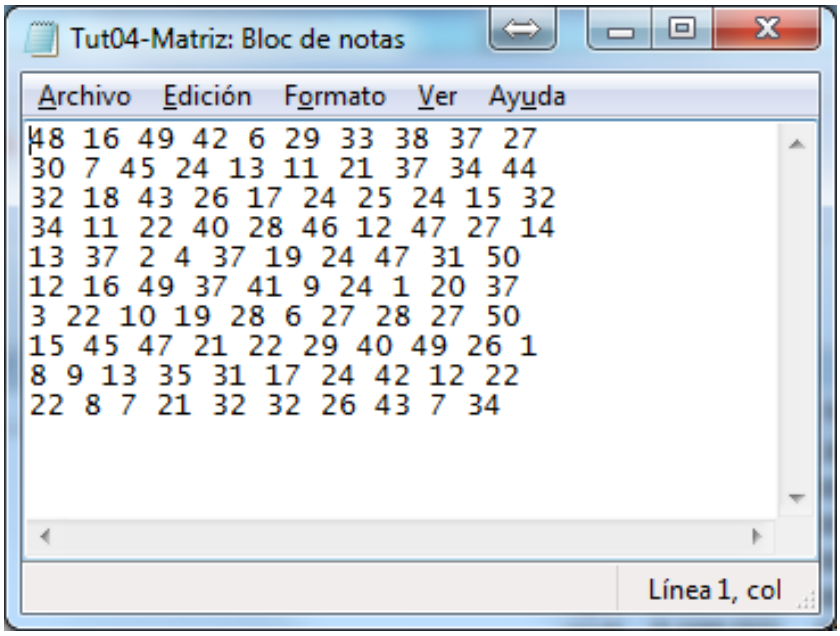
La función `read.table` siempre produce como resultado un `data.frame`. Y ya sabemos que la razón por la que usamos un `data.frame` es para poder trabajar con tablas, cuyas columnas contienen variables de distintos tipos. Las matrices, en cambio, tienen todas sus columnas del mismo tipo. Pero, puesto que en cualquier caso son también tablas, muchas veces usaremos ficheros `csv` para almacenar matrices, y leeremos esos ficheros usando la función `read.table`.

Por ejemplo, el fichero

Tut04-Matriz.csv

contiene una matriz $10 \times 10$ de números enteros. La siguiente figura muestra el contenido del fichero, tal y como se ve usando el *Bloc de Notas*:

Las distintas estructuras de datos, y la capacidad de R para modificarlas, nos pueden ser de utilidad a la hora de manipular ficheros de datos. Imagínate que tenemos un fichero de datos en el que los datos aparecen en forma de tabla, como el fichero Tut05-DatosEnTabla.csv, que en la siguiente figura mostramos abierto en el Bloc de Notas de Windows. Los datos, como puede verse, están separados por espacios, pero cada fila contiene 10 datos.




Para abrir este fichero con R (fijamos el directorio de trabajo y) usamos el comando

```
(datos = read.table(file="../datos/Tut04-Matriz.csv", sep=" "))
##    V1 V2 V3 V4 V5 V6 V7 V8 V9 V10
## 1  48 16 49 42  6 29 33 38 37  27
## 2  30  7 45 24 13 11 21 37 34  44
## 3  32 18 43 26 17 24 25 24 15  32
## 4  34 11 22 40 28 46 12 47 27  14
## 5  13 37  2  4 37 19 24 47 31  50
## 6  12 16 49 37 41  9 24  1 20  37
## 7   3 22 10 19 28  6 27 28 27  50
## 8  15 45 47 21 22 29 40 49 26   1
## 9   8  9 13 35 31 17 24 42 12  22
## 10 22  8  7 21 32 32 26 43  7  34
class(datos)

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

El resultado de la función `class` demuestra que la variable `datos` es de tipo `data.frame`, porque ese es el resultado que produce siempre `read.table`. De hecho, R piensa que la interpretación natural de este fichero es pensar que se trata de 10 observaciones (una por fila) de 10 variables (una por columna), y por eso ha añadido, de su cosecha, nombres para esas variables, llamándolas `V1, V2,...,V10`.

No es eso lo que queremos, claro. Pero, afortunadamente, R nos ofrece varias funciones que permiten convertir un `data.frame` en una matriz o un vector, si esas conversiones tienen sentido. En este ejemplo usaremos la función `as.matrix`, de este modo:

```
(matrizDatos = as.matrix(datos))

##       V1 V2 V3 V4 V5 V6 V7 V8 V9 V10
##  [1,] 48 16 49 42  6 29 33 38 37  27
##  [2,] 30  7 45 24 13 11 21 37 34  44
##  [3,] 32 18 43 26 17 24 25 24 15  32
##  [4,] 34 11 22 40 28 46 12 47 27  14
##  [5,] 13 37  2  4 37 19 24 47 31  50
##  [6,] 12 16 49 37 41  9 24  1 20  37
##  [7,]  3 22 10 19 28  6 27 28 27  50
##  [8,] 15 45 47 21 22 29 40 49 26   1
##  [9,]  8  9 13 35 31 17 24 42 12  22
## [10,] 22  8  7 21 32 32 26 43  7  34
```

```
class(matrizDatos)

## [1] "matrix"
```

Así que ya tenemos una matriz de R. Fíjate en que el formato de la respuesta (los nombres de filas) para esta matriz es diferente del formato del `data.frame` anterior. Para evitar una posible pérdida de información R ha conservado los nombres de las columnas, pero podemos librarnos de ellos fácilmente.

```
colnames(matrizDatos) = NULL
matrizDatos

##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]   48   16   49   42    6   29   33   38   37    27
##  [2,]   30    7   45   24   13   11   21   37   34    44
##  [3,]   32   18   43   26   17   24   25   24   15    32
##  [4,]   34   11   22   40   28   46   12   47   27    14
##  [5,]   13   37    2    4   37   19   24   47   31    50
##  [6,]   12   16   49   37   41    9   24    1   20    37
##  [7,]    3   22   10   19   28    6   27   28   27    50
##  [8,]   15   45   47   21   22   29   40   49   26     1
##  [9,]    8    9   13   35   31   17   24   42   12    22
## [10,]   22    8    7   21   32   32   26   43    7    34
```

Ya sabemos que la función `write.table` nos permite guardar un `data.frame` en un fichero `csv` (en realidad, en un fichero de texto; la extensión puede ser `txt`, `.dat`, o la que queramos). Pero también podemos usarla para escribir otros objetos de R, como matrices o vectores (y en ese caso, sustituye a la función `cat` que vimos en el Tutorial02).

Para que puedas practicar esto, haremos algunos ejercicios.

**Ejercicio 7.**

1. *Construye la matriz traspuesta de* `matrizDatos` *y guárdala en un fichero llamado* `traspuesta.csv`. *No queremos que el fichero contenga nada, excepto los números de la matriz, separados por espacios. Piensa, además, en lo que habrías tenido que hacer para hacer este trabajo a mano, sin usar R.*

2. *El fichero*

   *Tut04-3000datos.csv*

   *contiene una lista de 3000 números, escritos en forma de tabla de 5 columnas y 600 filas (de nuevo, para evitarte cualquier tentación de hacer el trabajo a mano). El objetivo de este ejercicio es convertir esos datos a un fichero* `csv` *con una única columna en la que aparezcan esos mismos 3000 números (leyendo por filas la tabla, de manera que la columna resultante empezará con los elementos de la primera fila de la tabla).*

*Solución en la página 30.* □

## 3. Condicionales `if-else` y bucles `for` en R.

**Opcional: esta sección puede omitirse en una primera lectura.**

Si R ha llegado a la posición que ahora ocupa, y si su radio de acción no deja de crecer, es sobre todo porque no se limita a ser un programa de Estadística más al uso, con cada vez más opciones en los menús. Ante todo, R es un lenguaje de programación, con un fuerte sesgo -desde luego-

hacia el Análisis de Datos y la Estadística. Para sacarle todo el partido a R hay que aprender algo de programación.

Y un programa, en código R, es un conjunto de instrucciones, como los que ya hemos aprendido a escribir, pero que toma decisiones por nosotros de forma automática, y que en función de esas decisiones puede repetir un conjunto de instrucciones una cierta cantidad de veces. Esos dos ingredientes, las decisiones, mediante condicionales, y la repetición, mediante bucles, son los dos pilares básicos de la programación en R, y en casi cualquier otro lenguaje de programación. En este tutorial vamos a aprender a usar los condicionales, y el tipo de bucle más sencillo, usándolos para ilustrar problemas sobre el Teorema de la Probabilidad Total.

Para empezar, veamos un ejemplo muy sencillo de toma de decisiones. La decisión más básica consiste siempre en elegir entre dos caminos posibles (en general, dos opciones; pero la imagen del camino ayuda a pensar). Y el esquema de una decisión básica siempre es este:

> *Si se cumple* **cierta condición**, *entonces*
> *vamos por* **el camino A**.
> *Y si no se cumple,*
> *vamos por* **el camino B**.

Las frases que hemos destacado, en negrita, son las que cambiarán en cada caso concreto, porque en cada ejemplo la condición será distinta, y distintos serán también los caminos A y B. Pero la forma de la frase es la misma siempre. Para traducir una frase de esta forma al lenguaje de R disponemos de un estructura especial que, esquemáticamente, dejando todavía sin traducir las partes destacadas de la frase, funciona así:

```
if(se cumple cierta condicion){
  vamos por el camino A
} else {
  vamos por el camino B
}
```

Veamos un ejemplo de decisión más concreto. Vamos a lanzar un dado (usando la función `sample` de R). Si el resultado es mayor o igual que tres, entonces gano un euro. Y si eso no se cumple (es decir, si es menor que tres), pierdo un euro. Vamos a escribir un fragmento de código R que calcule mis ganancias o pérdidas después de este juego tan sencillo. En R esto se convierte en:

```
    (dado=sample(1:6,1))

    if(dado >= 3){
        ganancia = 1
    } else {
        ganancia = -1
    }

    ganancia
```

Antes de seguir adelante es una buena idea que ejecutes varias veces este código para que veas que, en efecto, se obtienen las respuestas esperadas según sea el resultado del dado.

El primer paso de una estructura condicional `if/else` es, naturalmente, una condición. Las condiciones en R son expresiones booleanas (o lógicas), que ya hemos visto en la Sección 6.3 (pág. 42) del Tutorial02. Es decir, una fórmula que sólo puede tomar dos valores, verdadero o falso. La fórmula `dado >= 3` es una de estas fórmulas lógicas, porque al sustituir cada valor concreto de `dado`, el resultado será verdadero o falso, dos valores que que en R se representan mediante dos expresiones que ya conocemos, `TRUE` y `FALSE`. Para ver esto con más detalle, vamos a ver un par de ejemplos de ejecución del código de la condición (cuando tú lo ejecutes obtendrás otros resultados, claro):

```
    (dado = sample(1:6,1))

## [1] 5
```

```
    dado >= 3
```

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

```
    (dado = sample(1:6,1))
```

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

```
    dado >= 3
```

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

En este fragmento de código no usamos el resultado de la condición (o fórmula lógica) `dado >= 3` para nada. Nos limitamos a *calcular* esa fórmula y mostrar el resultado. Ejecuta estos comandos varias veces. Obtendrás `TRUE` o `FALSE` como resultado según cual haya sido el resultado de `dado`, claro. Si es necesario, vuelve ahora al primer ejemplo de `if else` que hemos visto, y asegúrate de que entiendes su mecanismo.

Las fórmulas booleanas, o lógicas, pueden ser muy sencillas, como ese `dado >= 3` que hemos visto, o pueden ser bastante más complicadas. Iremos aprendiendo más sobre ellas a lo largo del curso, pero podemos adelantarte que están muy relacionadas con las operaciones de unión e intersección que hacemos con los sucesos en Probabilidad. Al fin y al cabo, un suceso $A$ es algo que puede ocurrir o no ocurrir, y eso es similar a decir que $A$ es `TRUE` cuando ocurre, y `FALSE` cuando no ocurre. Y de la misma forma que combinamos los sucesos con:

- uniones (A o B),
- intersecciones ( A y B)
- y complementarios (no A, o lo contrario de A),

también aprenderemos a combinar las fórmulas booleanas con operaciones análogas. Pero antes, vamos a ver un ejemplo más elaborado de estructura `if-else`, relacionado con el Teorema de las Probabilidades Totales. Esta es la descripción del problema:

> *Tiramos un dado. Si el resultado es menor que 5, usamos una urna con 1 bola blanca y 9 negras. Si es 5 o 6, usamos una urna con 4 bolas blancas y 3 bolas negras. En cualquiera de los dos casos, extraemos una bola al azar. ¿Cuál es la probabilidad de que esa bola sea negra?*

El Teorema de las Probabilidades Totales proporciona este valor de la probabilidad:

$$P(\text{bola negra}) = P(\text{bola negra}\,|\,\text{urna 1})P(\text{urna 1}) + P(\text{bola negra}\,|\,\text{urna 2})P(\text{urna 2}) =$$

$$\frac{4}{6}\cdot\frac{9}{10} + \frac{2}{6}\cdot\frac{3}{7} = \frac{26}{35} \approx 0.743$$

Y lo que vamos a hacer es usar R para comprobar "experimentalmente" este resultado, mediante una simulación, como hemos hecho en otras ocasiones. Para hacer esto necesitamos un fragmento de código R que tire un dado y, en función del resultado del dado, elija una urna y extraiga una bola de esa urna. Para empezar escribimos este esquema del código, que todavía no funciona. Es un *borrador* del código definitivo, que de momento sólo contiene la estructura `if/else` básica, acompañada de comentarios que nos dicen lo que queremos hacer al tomar cada uno de los dos caminos (o ramas, o brazos, que de todas esas formas se llaman):

```
    # Tiramos el dado.
    (dado = sample(1:6,1))
    # y, según el resultado, elegimos urna.
    if(dado < 5){
        # Usamos la urna 1 para elegir la bola.
    } else {
```

```
        # Usamos la urna 2 para elegir la bola.
    }
    # Y, al final, mostraremos la bola que ha salido.
```

Para escribir el código que falta, supongamos por un momento que el dado ha resultado menor que 5. Entonces tenemos que sacar una bola blanca o negra de la urna 1. Como se trata de un sorteo, vamos a usar la función `sample`. Podríamos usar números para codificar los colores blanco y negro, diciendo por ejemplo que 1 es blanco y 2 es negro. Pero vamos a hacer algo mejor, y vamos a aplicar `sample` a un vector de caracteres, así (añadimos paréntesis para que veas el tipo de resultado que se obtiene):

```
(bolaUrna1 = sample( c("blanca","negra"), 1, prob=c(1,9) ))

## [1] "negra"
```

El vector `prob=c(1,9)` es el que contiene la información sobre la composición de esta urna. Si quieres estar seguro de que lo entiendes, ejecuta esta línea de código varias veces.

**Ejercicio 8.**
*Escribe la línea que sortea una bola para la urna 2.* *Solución en la página 31.* □

¡No sigas, si no has hecho este ejercicio!

Juntando las piezas obtenemos el código completo de la simulación (se muestra el código sin ejecutar).

```
# Tiramos el dado.
(dado = sample(1:6,1))
# y según el resultado, elegimos urna.
if(dado < 5){
  # usamos la urna 1 para elegir la bola.
  bola = sample( c("blanca","negra"), 1, prob=c(1,9) )
} else {
  # usamos la urna 2 para elegir la bola.
  bola = sample( c("blanca","negra"), 1, prob=c(4,3) )
}
# Y, al final, mostraremos la bola que ha salido:
bola
```

Fíjate en que, al añadir el código, desde luego, no hemos quitado los comentarios. Siguen cumpliendo una de esas funciones básicas, que es la de explicarnos (a nosotros mismos, o a otros usuarios del código) cuál es la lógica que hemos utilizado, y para qué sirve cada cosa. Copia ese código en RStudio, ejecútalo varias veces y mira cómo funciona. Obtendrás, como era de esperar, más bolas negras que blancas.

Claro, el problema es que para comprobar experimentalmente lo que predice el Teorema de las Probabilidades Totales, tendríamos que tirar el dado muchas veces, varios miles de veces probablemente. Y no tiene sentido ejecutar el código anterior a mano miles de veces. Lo que necesitamos es, como habíamos adelantado, aprender a pedirle a R que repita algo un cierto número de veces.

## 3.1. El bucle `for` de R

El bucle `for` es una estructura de programación de R que sirve para repetir cierta tarea un número fijo de veces. Al decir que el número de repeticiones es fijo lo que queremos decir es que el número de repeticiones se decide antes de empezar a repetir la tarea. Este tipo de bucle, el bucle `for`, es, por esa razón, muy sencillo. Porque primero decidimos el número $n$ de veces que queremos repetir una acción, y luego le decimos a R, esencialmente "repite esto $n$ veces". Para llevar la cuenta de cuantas veces hemos pasado ya por el bucle, se utiliza una variable de control, que aparece al principio del bucle, y recorre un cierto rango de números.

Veamos un ejemplo muy sencillo. Vamos a escribir un bucle que repite la misma tarea sencilla 10 veces. La tarea consiste en aumentar la variable `cuenta` en 3 unidades, cada vez que pasamos por el bucle. El valor inicial de `cuenta` es 0. Y además de esa variable cuenta, tenemos la variable de control del bucle, llamada `k`, que recorre los valores del rango `1:10`. El valor de `k` nos dice cuántas veces hemos repetido el bucle. Naturalmente, si empezamos en 0, aumentamos `cuenta` de 3 en 3, y repetimos esa acción 10 veces, el valor final debería de `cuenta` debería ser 30. El código del correspondiente bucle `for` es este:

```
    cuenta=0
    for(k in 1:10){
        # Cuerpo del bucle (entre las dos llaves)
        cuenta = cuenta + 3
    }
    cuenta

## [1] 30
```

El resultado es el valor 30 esperado. Como hemos indicado, el bucle consta de una primera línea, la cabecera del bucle, que en este caso es:

```
for(k in 1:10){
```

La cabecera termina con una llave `{` abierta, que indica el comienzo del `cuerpo de bucle`, que consta de un comentario y de la línea que realmente se repite para modificar el valor de `cuenta`

```
        # Cuerpo del bucle (entre las dos llaves)
        cuenta = cuenta + 3
```

Estas líneas, las que forman el cuerpo, son las que se repiten cada vez que pasamos el bucle. Y cada una de esas repeticiones es una iteración del bucle. Al ejecutar esas líneas de código (las del cuerpo) dentro de un bucle, no vemos los valores intermedios de la variable `cuenta`, ni los de la variable de control `k`. Podrías pensar en rodear con paréntesis la línea del cuerpo del bucle, así:

```
(cuenta = cuenta + 3)
```

Pero esto no funcionará, al estar dentro de un bucle. Y si encierras todo el bucle entre paréntesis, R te mostrará sólo el resultado final del bucle. Para conseguir esto, vamos a usar un nueva función de R, llamada `print`. Añadimos una línea al cuerpo del bucle, para que el código completo quede así:

```
    cuenta=0
    for(k in 1:10){
        # Cuerpo del bucle (entre las dos llaves)
        cuenta = cuenta + 3
        print(cuenta)
    }
```

```
## [1] 3
## [1] 6
## [1] 9
## [1] 12
## [1] 15
## [1] 18
## [1] 21
## [1] 24
## [1] 27
## [1] 30
```

```
    cuenta
```

```
## [1] 30
```

Y ahora sí funciona, como ves. El resultado de la ejecución muestra los valores intermedios de la variable `cuenta` en cada iteración del bucle.

**Ejercicio 9.**
*Modifica el código para que, además, se muestre el valor de la variable de control* `k`. *Solución en la página 31.* □

Con esto ya estamos listos para escribir un bucle `for` que repita el experimento del Teorema de las Probabilidades anteriores del apartado anterior un número arbitrario de veces. El código para 10000 tiradas del dado es así (lo analizaremos a continuación):

```
# Empezamos lanzando un dado n veces.
n = 10000
dado = sample(1:6, n, replace=TRUE)

# El proceso ahora depende de si el dado es o no menor que 5
bola=c()
for(k in 1:n){
  if(dado[k] < 5){
      #Se ha elegido la urna 1.
      #Estas instrucciones (hasta la linea con else) se usan si dado<5
      #print("Usamos una urna con 3 bolas blancas y 5 negras")
```

```
      (bola = c(bola, sample(c("b","n"), 1, prob=c(1,9)) ) )
  }else{
    #Se ha elegido la urna 2.
    #Estas instrucciones (hasta la llave) se usan si dado>=5
    #print("Usamos una urna con 7 bolas blancas y 3 negras")
      (bola = c(bola, sample(c("b","n"), 1, prob=c(4,3)) ) )
  }
}
# Y al final, miramos la tabla de frecuencias relativas.
table(bola) / length(bola)

## bola
##     b     n
## 0.258 0.743
```

Como ves, el resultado de esa simulación en particular se parece mucho a lo que predice el Teorema de las Probabilidades Totales (no siempre es tan preciso). El aspecto más novedoso de este código es que usamos un vector, el vector `bola`, de tipo `character`, que almacena los resultados, representando con `"b"` la bola blanca y con `"n"` la bola negra. En el cuerpo del bucle tenemos un condicional `if/else` como el que ya hemos visto antes. Pero ahora, después de extraer la bola debemos recordar el resultado, guardarlo en algún sitio, para que, al llegar al final del bucle, tengamos la lista completa de resultados. De eso se encarga el vector `bola`. Las dos líneas que empiezan así:

```
(bola=c(bola, sample
```

dicen esto: "toma el vector `bola`, añádele al final el resultado de `sample`, y guarda el resultado otra vez con el nombre `bola`". De esa manera, en cada iteración del bucle vamos concatenando los resultados de la extracción de una nueva bola. Al final, en la última línea de código, calculamos la tabla de frecuencias de los dos colores, para ver si se corresponden con lo que predice el teorema.

**Ejercicio 10.**

1. *Copia el código del programa y ejecútalo unas cuantas veces (sin usar* `set.seed`*, para que cada simulación represente* 10000 *nuevas tiradas), para ver lo que sucede.*
2. *Para ver más detalladamente como se va formando el vector* `bola`*, puedes añadir líneas con la función* `print`*. Debes reducir mucho el número de iteraciones (por ejemplo a 10), para que la salida del programa no sea excesivamente larga.*

*Solución en la página 32.* □

Podemos detenernos un momento a mirar el código de la simulación, y sentirnos orgullosos: hemos escrito nuestro primer programa en R. Es verdad que es un programa modesto, y por eso estamos modestamente orgullosos. Pero no es menos cierto que hemos escrito un fragmento de código que, ante un valor aleatorio como es `dado`, toma decisiones de forma autónoma, sin consultarnos, y produce un resultado interesante: la tabla de frecuencias de esta simulación.

# 4. Ejercicios adicionales y soluciones.

## Ejercicios adicionales

**Función de densidad de una variable aleatoria discreta.**

1. Una compañía de seguros agrarios ha recopilado la siguiente tabla sobre seguros para pérdidas en cosechas.

| Porcentaje de Ha perdidas | 0 | 25 | 50 | 100 |
|---|---|---|---|---|
| Probabilidad | 0.9 | 0.05 | 0.02 | ?? |

Si ofrecen una poliza que paga 150 euros por cada hectárea perdida, ¿cuál es la indemnización media (en euros/hectárea) que esperan pagar?

2. Sea $X$ una variable aleatoria discreta cuya distribución viene dada por esta tabla:

| Valor | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Probabilidad | 1/6 | 1/12 | 1/4 | 1/4 | 1/12 | 1/6 |

Calcular la media de las variables $5X + 1$ y $X^2$ ($X$ al cuadrado).

3. En el *chuck-a-luck*, un juego típico de los casinos de Las Vegas, el croupier lanza tres dados. Cada jugador apuesta por un número entre 1 y 6 y recibe una cantidad igual a su apuesta si el número aparece una vez, el doble si aparece dos veces y el triple si aparece tres veces. Si su número no figura entre los resultados, pierde su apuesta. Calcular el beneficio esperado del jugador.

**Varianza de una variable aleatoria discreta.**

4. Calcula la varianza de las variables que aparecen en los ejercicios previos de esta hoja. Es decir, busca la forma de, usando el ordenador, automatizar la tarea de calcular la varianza a partir de la tabla de densidad de probabilidad de una variable aleatoria discreta. Indicación: no debería suponer mucho trabajo, ya hemos hecho algo parecido antes en el curso.

5. En la Sección 1.3 hemos visto la identidad

$$\mathrm{Var}(X) = E(X^2) - (E(X))^2\,,$$

que es válida en el contexto de las variables aleatorias. En este ejercicio queremos que el lector conozca una identidad estrechamente relacionada con esta, que se aplica de forma general a cualquier conjunto de números. Dados $n$ números cualesquiera,

$$x_1, x_2 \ldots, x_n$$

la diferencia entre la media de sus cuadrados y el cuadrado de su media es la varianza poblacional de ese conjunto de números. Es decir:

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

*a*) El primer objetivo concreto de este ejercicio es usar R para elegir 50 números al azar, por ejemplo entre $-100$ y 100 y con reemplazamiento, y usarlos para comprobar esta identidad.

*b*) Un segundo objetivo, más teórico, consiste en entender por qué siempre sucede esto, y por qué es cierta la identidad en el caso de las variables aleatorias.

**Función de distribución.**

6. Sea $X$ una variable aleatoria discreta cuya densidad de probabilidad viene dada por esta tabla:

| Valor | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|
| Probabilidad | 0.05 | 0.1 | 0.6 | 0.15 | 0.1 |

*a*) Hallar la función de distribución acumulada $F$.

*b*) Usar $F$ para calcular $P(X \leq 8)$.

*c*) Usar $F$ para calcular $P(X < 8)$. Usar $F$ para calcular $P(X > 7)$. Usar $F$ para calcular $P(7 \leq X \leq 9)$.

7. Construye la (tabla de la) función de distribución de las variables que aparecen en los ejercicios previos de esta hoja. Indicación: sirve la misma indicación que para el ejercicio 7.

**Trabajo con `data.frames` de R.**

8. Sólo si has leído la Sección 4.5 (pág. 115) del libro.

   *a*) Usar R para construir la Tabla 4.12(a) del libro (pág. 121), que corresponde al Ejemplo 4.5.1. Concretamente, se trata de construir un `data.frame` cuyas cuatro columnas contengan las columnas de esa tabla.

   *b*) Construye también (como matriz de R) la tabla de densidad conjunta de $X$ e $Y$ (Tabla 4.12(b) del libro). Usa una representación numérica de los valores para simplificar las operaciones (en lugar de las fracciones que hemos usado nosotros). Comprueba que la suma de todos los elementos de esa matriz es 1.

   *c*) Construye, a partir de esa tabla, las densidades marginales de $X$ e $Y$.

   *d*) Usa las marginales para comprobar la posible independencia de $X$ e $Y$.

   *e*) Calcula también la tabla de densidades condicionadas con respecto a $X$ (ver Ejemplo 4.5.5 del libro, pág. 125) y con respecto a $Y$.

9. Sólo si has leído la Sección 4.5 (pág. 115) del libro.

   La construcción, usando R, de la Tabla 4.11 del libro excede los objetivos de este curso, porque eso nos obligaría a aprender bastantes detalles sobre la forma en la que R gestiona la información sobre fechas [1]. En lugar de eso, el fichero adjunto:

   Tut04-dias2014.csv

   contiene los datos ya procesados, a partir de los cuales es sencillo construir esa tabla. Una vez construida, piensa cuánto deben sumar todos sus elementos, y luego comprueba que, en efecto, es así.

**Densidades marginales, condicionadas e independencia.**

10. Sólo si has leído la Sección 4.5 (pág. 115) del libro.
    Sea $X$ la variable suma de dos dados, y sea $Z$ la variable

    $$Z = \text{mín}(\text{dado1}, \text{dado2}).$$

    Calcula la función de densidad condicionada:

    $$P(Z|X = 7).$$

## Soluciones de algunos ejercicios

**• Ejercicio 1, pág. 2**

Ya sabes que para experimentar con otros valores basta con comentar (usa `#`) la línea con `set.seed`.

```
set.seed(2014)
n = 1000000
dado1 = sample(1:6, n, replace=TRUE)
dado2 = sample(1:6, n, replace=TRUE)
suma = dado1 + dado2
table(suma)/n
```

```
## suma
##      2      3      4      5      6      7      8      9     10     11
## 0.0279 0.0558 0.0829 0.1107 0.1389 0.1668 0.1396 0.1109 0.0833 0.0554
##     12
## 0.0278
```

[1] El lector interesado (y es un tema útil e interesante) encontrará más información, por ejemplo, en el libro *R in a Nutshell*, que aparece en la Bibliografía del libro.

Puedes comparar estas frecuencias relativas (experimentales) con las probabilidades (teóricas):

```
c(1:6,5:1)/36

##  [1] 0.0278 0.0556 0.0833 0.1111 0.1389 0.1667 0.1389 0.1111 0.0833 0.0556
## [11] 0.0278
```

- **Ejercicio 2, pág. 3**

1. Los valores y probabilidades son:

   ```
   valores = 2:12
   probabilidades = c(1:6,5:1)/36
   ```

   Así que la media, varianza y desviación típica son:

   ```
   (mu=sum(valores*probabilidades) )

   ## [1] 7

   (varianza=sum((valores-mu)^2*probabilidades) )

   ## [1] 5.83

   (sigma=sqrt(varianza) )

   ## [1] 2.42
   ```

2. Para obtener un valor simbólico en Wolfram Alpha evalúa este comando (corta y pega):

   ```
   (1/36)*(2-7)^2 + (2/36)*(3-7)^2 + (3/36)*(4-7)^2 + (4/36)*(5-7)^2 + (5/36)*(6-7)^2 +
   (6/36)*(7-7)^2 + (5/36)*(8-7)^2 + (4/36)*(9-7)^2 + (3/36)*(10-7)^2 + (2/36)*(11-7)^2
                                  + (1/36)*(12-7)^2
   ```

   ¿Cómo se puede obtener una expresión como esta sin que nos asalten el tedio y/o la melancolía? Pues aprendiendo a usar la función `paste` de R, de la que hablaremos próximamente en otro tutorial.

   Para explicar cómo hacer esta cuenta con Wiris (de manera no manual) tendríamos que adentrarnos más de lo que queremos en la sintaxis de ese programa. Una posibilidad, sin salir de R, es usar la función fractions de la librería `MASS`, de este modo:

   ```
   library(MASS)
   valores = 2:12
   (probabilidades= fractions(c(1:6,5:1)/36))

   ##  [1] 1/36 1/18 1/12  1/9 5/36  1/6 5/36  1/9 1/12 1/18 1/36

   sum((valores-7)^2*probabilidades)

   ## [1] 35/6
   ```

3. El código en R es:

   ```
   library(MASS)
   valores = 0:5
   probabilidades = fractions(c(6,10,8,6,4,2)/36)
   (mu = sum(valores * probabilidades))
   ```

```
## [1] 35/18

(varianza=sum((valores-mu)^2*probabilidades) )

## [1] 665/324

(sigma=sqrt(varianza))

## [1] 2563/1789
```

Para convertirlos en valores numéricos podemos usar `as.numeric`:

```
as.numeric(mu)

## [1] 1.94

as.numeric(varianza)

## [1] 2.05

as.numeric(sigma)

## [1] 1.43
```

- **Ejercicio 3, pág. 6**

El valor es $F(\frac{8}{3}) = 0.2$, porque se tiene $2 < \frac{8}{3} < 4$, así que

$$F(\frac{8}{3}) = P(X \leq \frac{8}{3}) = P(X \leq 2) = F(2).$$

- **Ejercicio 4, pág. 11**

```
(satelitesGalileanos[ ,2] > 4000)

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

El resultado es un vector de cuatro valores lógicos (`TRUE/FALSE`)que nos dicen, para cada fila del `data.frame` si la condición se cumple. Es decir si el valor en la segunda columna de esa fila es, o no, mayor que 4000.

- **Ejercicio 5, pág. 13**

1. Código para la primera parte:

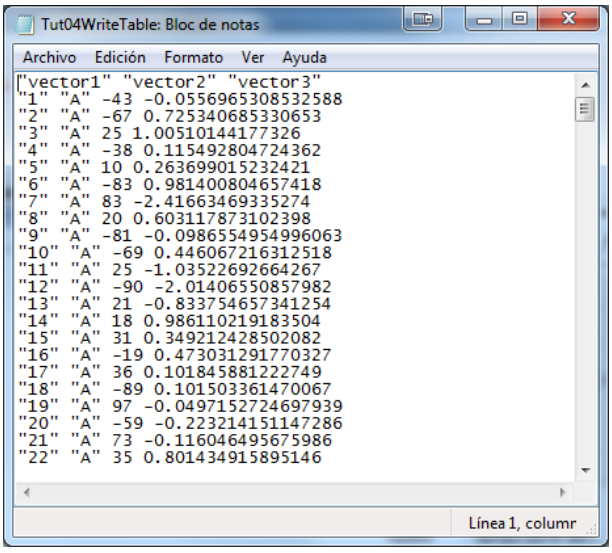
```
set.seed(2014)
vector1 = rep(c("A", "B", "C"), rep(100, 3))
vector2 = sample(-100:100, 300, replace=TRUE)
vector3 = rnorm(300)

Tut04WriteTable = data.frame(vector1, vector2, vector3)
head(Tut04WriteTable)
```

```
##   vector1 vector2 vector3
## 1       A     -43 -0.0557
## 2       A     -67  0.7253
## 3       A      25  1.0051
## 4       A     -38  0.1155
## 5       A      10  0.2637
## 6       A     -83  0.9814

tail(Tut04WriteTable)

##     vector1 vector2 vector3
## 295       C      81   1.126
## 296       C     -67   0.383
## 297       C      38  -0.645
## 298       C     -71  -0.805
## 299       C      -1  -0.703
## 300       C      89   1.841
```

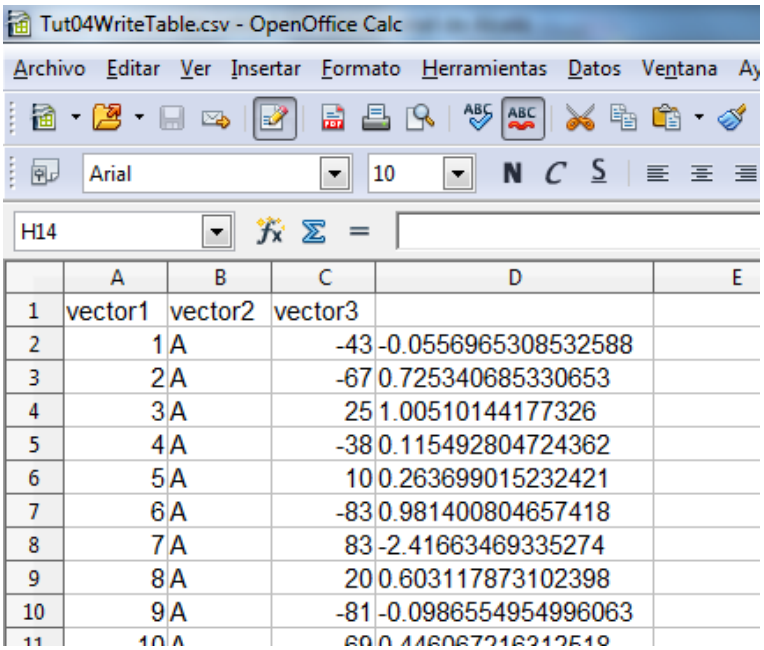
2. Tras ejecutar

```
write.table(Tut04WriteTable, file="../datos/Tut04WriteTable.csv")
```

el *Bloc de Notas* muestra que el contenido del fichero `Tut04WriteTable.csv` tiene este aspecto:



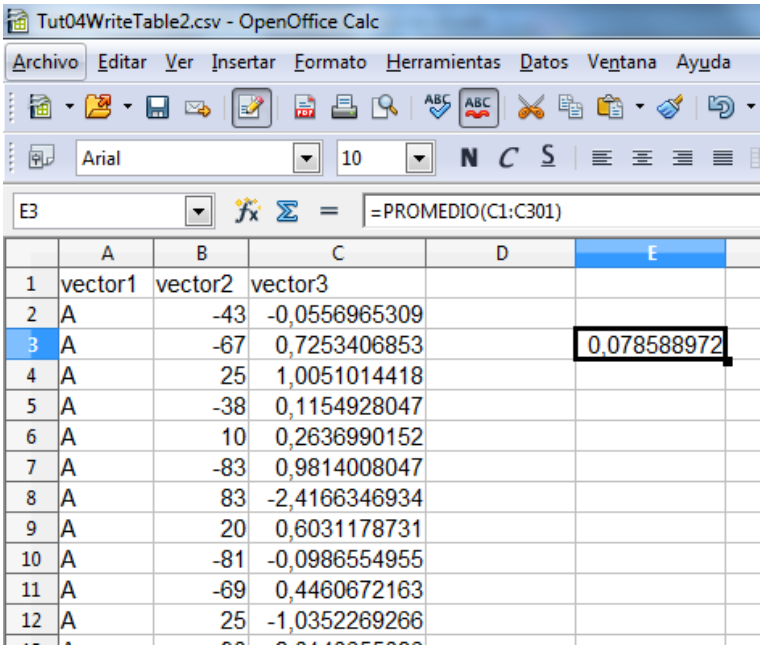

3. La primera dificultad es que R ha añadido una primera columna adicional con los números de las filas, que en Calc aparecen en la columna `A` (eso, además, hace que los nombres de las variables queden descolocados), como se ve en esta figura.




| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | vector1 | vector2 | vector3 | | |
| 2 | 1 | A | -43 | -0.0556965308532588 | |
| 3 | 2 | A | -67 | 0.725340685330653 | |
| 4 | 3 | A | 25 | 1.00510144177326 | |
| 5 | 4 | A | -38 | 0.115492804724362 | |
| 6 | 5 | A | 10 | 0.263699015232421 | |
| 7 | 6 | A | -83 | 0.981400804657418 | |
| 8 | 7 | A | 83 | -2.41663469335274 | |
| 9 | 8 | A | 20 | 0.603117873102398 | |
| 10 | 9 | A | -81 | -0.0986554954996063 | |

Pero, además, los elementos de la tercera columna usan puntos (en lugar de comas) como separadores decimales. A Calc, en su versión en español, eso le hace pensar que no se trata de números. Y si intentas calcular la media (recuerda usar la función `PROMEDIO`) aparecerá un mensaje de error.

4. El fichero `Tut04WriteTable2.csv`, tras abrirlo con Calc y calcular la media de la tercera columna (en la celda `E3`), muestra este aspecto:




| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | vector1 | vector2 | vector3 | | |
| 2 | A | -43 | -0,0556965309 | | |
| 3 | A | -67 | 0,7253406853 | | 0,078588972 |
| 4 | A | 25 | 1,0051014418 | | |
| 5 | A | -38 | 0,1154928047 | | |
| 6 | A | 10 | 0,2636990152 | | |
| 7 | A | -83 | 0,9814008047 | | |
| 8 | A | 83 | -2,4166346934 | | |
| 9 | A | 20 | 0,6031178731 | | |
| 10 | A | -81 | -0,0986554955 | | |
| 11 | A | -69 | 0,4460672163 | | |
| 12 | A | 25 | -1,0352269266 | | |

La media, calculada por Calc, se puede comprobar con R de cualquiera de estas dos formas (una usa el vector `vector3` y la otra la tercera columna del `data.frame`):

```
mean(vector3)

## [1] 0.0786

mean(Tut04WriteTable[,3])

## [1] 0.0786
```

- **Ejercicio 6, pág. 14**

```
colnames(satelitesGalileanos) = c("nombres", "diametrosKm", "periodoOrbital")
head(satelitesGalileanos)

##     nombres diametrosKm periodoOrbital
## 1        Io        3643           1.77
## 2    Europa        3122           3.55
## 3 Ganimedes        5262           7.16
## 4   Calisto        4821          16.69
```

El resultado de `dim` es:

```
dim(satelitesGalileanos)

## [1] 4 3
```

Es decir, un vector con el número de filas y columnas del `data.frame`.

- **Ejercicio 7, pág. 17**

1. El código para la primera parte es:

   ```
   traspuesta = t(matrizDatos)
   write.table(traspuesta, file="../datos/Traspuesta.csv",
       row.names = FALSE, col.names=FALSE, sep=" ")
   ```

   Hemos dividido la función `write.table` en dos líneas, para ajustarla al ancho de página.

2. Para la segunda parte empezamos por leer el fichero en un data.frame, que vamos a llamar igual que el fichero (es una costumbre que a menudo resulta útil), salvo por el cambio de `-` a `_`, porque el guión alto `-` representa la resta en R, y no se puede usar en nombres de objetos:

   ```
   Tut04_3000datos = read.table(file="../datos/Tut04-3000datos.csv", header=FALSE, sep=" ")
   ```

   Una vez hecho esto, convertimos el `data.frame` en una matriz:

   ```
   matriz3000Datos = as.matrix(Tut04_3000datos)
   head(matriz3000Datos, 10)

   ##       V1 V2 V3 V4 V5
   ##  [1,] 81 21  6 40 43
   ##  [2,] 60 62 34 24 35
   ##  [3,] 60 35 37  7 63
   ##  [4,] 13 46 67 76 63
   ##  [5,] 69 72 94 83  9
   ##  [6,] 43 70 60 69 59
   ##  [7,] 40 81 17 73  2
   ##  [8,] 53 26 50 54 71
   ##  [9,] 13 33  9 22 82
   ## [10,] 12 44 60 55 45
   ```

   Para convertirlo en un vector, recorriendo la matriz por filas, vamos a usar lo que aprendimos en la Sección 2.5 (pág. 13) del Tutorial03. Mostramos sólo los primeros 20 elementos del vector resultante:

```
head(as.vector(t(matriz3000Datos)), 20)

##  [1] 81 21  6 40 43 60 62 34 24 35 60 35 37  7 63 13 46 67 76 63
```

Finalmente para guardarlo en un fichero `csv` puedes usar `cat` (que ya conoces de anteriores tutoriales) o puedes usar `write.table` así:

```
write.table(as.vector(t(matriz3000Datos)), file="../datos/Tut04-3000datos-solucion.csv",
        row.names=FALSE, col.names=FALSE)
```

- **Ejercicio 8, pág. 20**

```
(bolaUrna2 = sample( c("blanca","negra"), 1, prob=c(4,3) ))

## [1] "negra"
```

- **Ejercicio 9, pág. 22**

Los valores de `cuenta` y `k` se alternan en la salida. Ya aprenderemos a presentarlos un poco mejor.

```
    cuenta=0
    for(k in 1:10){
        # Cuerpo del bucle (entre las dos llaves)
        cuenta = cuenta + 3
        print(cuenta)
        print(k)
    }

## [1] 3
## [1] 1
## [1] 6
## [1] 2
## [1] 9
## [1] 3
## [1] 12
## [1] 4
## [1] 15
## [1] 5
## [1] 18
## [1] 6
## [1] 21
## [1] 7
## [1] 24
## [1] 8
## [1] 27
## [1] 9
## [1] 30
## [1] 10

    cuenta

## [1] 30
```

- **Ejercicio 10, pág. 23**

1. Aquí puedes ver el resultado de tres ejecuciones del código, todas con $n = 10000$:

```
## bola
##     b     n
## 0.258 0.743
```

```
## bola
##     b     n
## 0.251 0.749
```

```
## bola
##     b     n
## 0.251 0.750
```

2. El código modificado es este:

```
# Empezamos lanzando un dado n veces.
n = 10
dado = sample(1:6, n, replace=TRUE)

# El proceso ahora depende de si el dado es o no menor que 5
bola=c()
for(k in 1:n){
  if(dado[k] < 5){
      #Se ha elegido la urna 1.
      #Estas instrucciones (hasta la linea con else) se usan si dado<5
      #print("Usamos una urna con 3 bolas blancas y 5 negras")
      bola = c(bola, sample(c("b","n"), 1, prob=c(1,9)) )
  } else {
    #Se ha elegido la urna 2.
    #Estas instrucciones (hasta la llave) se usan si dado>=5
    #print("Usamos una urna con 7 bolas blancas y 3 negras")
      bola = c(bola, sample(c("b","n"), 1, prob=c(4,3)) )
  }
  print("----------------------------------------")
  print(c(" dado =", dado[k]) )
  print(c("k = ", k))
  print(bola)
}
# Y al final, miramos la tabla de frecuencias relativas.
table(bola) / length(bola)
```

Puedes ver que hemos cambiado $n = 10$, y que hemos añadido un grupo de sentencias con la función `print`, dentro del bucle `for` pero fuera del condicional `if/else`. El resultado de esas modificaciones es este:

```
## [1] "----------------------------------------"
## [1] " dado =" "2"
## [1] "k = " "1"
## [1] "b"
## [1] "----------------------------------------"
## [1] " dado =" "6"
## [1] "k = " "2"
## [1] "b" "n"
## [1] "----------------------------------------"
## [1] " dado =" "4"
```

```
## [1] "k = " "3"
## [1] "b" "n" "b"
## [1] "----------------------------------------"
## [1] " dado =" "4"
## [1] "k = " "4"
## [1] "b" "n" "b" "n"
## [1] "----------------------------------------"
## [1] " dado =" "3"
## [1] "k = " "5"
## [1] "b" "n" "b" "n" "n"
## [1] "----------------------------------------"
## [1] " dado =" "1"
## [1] "k = " "6"
## [1] "b" "n" "b" "n" "n" "n"
## [1] "----------------------------------------"
## [1] " dado =" "4"
## [1] "k = " "7"
## [1] "b" "n" "b" "n" "n" "n" "n"
## [1] "----------------------------------------"
## [1] " dado =" "6"
## [1] "k = " "8"
## [1] "b" "n" "b" "n" "n" "n" "n" "b"
## [1] "----------------------------------------"
## [1] " dado =" "3"
## [1] "k = " "9"
## [1] "b" "n" "b" "n" "n" "n" "n" "b" "n"
## [1] "----------------------------------------"
## [1] " dado =" "5"
## [1] "k = " "10"
##  [1] "b" "n" "b" "n" "n" "n" "n" "b" "n" "n"
## bola
##   b   n
## 0.3 0.7
```

Hemos usado un par de veces un "truco" para indicar cual es la variable que se muestra. Por ejemplo, en la línea de código:

```
print(c("k = ", k))
```

Al usar `c("k = ", k)` estamos construyendo un vector que contiene una mezcla de números y texto. Es muy posible que no lo recuerdes, pero en la Sección **??** del Tutorial02 hemos comentado brevemente que en estos casos, R convierte todos los elementos del vector en cadenas alfanuméricas. El resultado dista todavía mucho de lo que se puede conseguir (¡todas esas comillas!), pero es un primer paso.

Naturalmente, en este caso, con $n$ tan pequeño, las frecuencias se parecen bastante menos a las que predice la teoría.

---

Fin del Tutorial-04. ¡Gracias por la atención!