El entorno estadístico R

Estructura, lenguaje y sintaxis

Daniel Liviano Solís
Maria Pujol Jover

PID_00208271

UOC
Universitat Oberta de Catalunya

Índice

Introducción

Este módulo tiene como principal objetivo introducir al estudiante en el entorno estadístico R. Se podría definir R como un lenguaje y un entorno para computación y gráficos estadísticos. Es un proyecto libre, es decir, no requiere el pago de ninguna licencia. R proporciona una amplia variedad de técnicas estadísticas (modelos lineales y no lineales, pruebas estadísticas clásicas, análisis de series temporales, análisis multivariante, etc.). Además, también ofrece una alta potencialidad a la hora de elaborar gráficos compejos y de calidad.

El uso de R en las asignaturas cuantitativas ofrecidas por la UOC permite al estudiante la realización de muchas y variadas posibilidades de análisis numérico y estadístico. Algunas de estas se detallan a continuación. R proporciona:

1) Facilidades para almacenar y manejar datos numéricos en diferentes formatos, como vectores, matrices y bases de datos.

2) Una amplia colección de operadores y funciones matemáticas.

3) Una larga lista de paquetes estadísticos con todo tipo de herramientas estadísticas y matemáticas.

4) Facilidades gráficas para el análisis y la visualización de datos.

5) Un lenguaje de programación sencillo y eficaz bien desarrollado, que incluye condicionales, bucles y funciones recursivas.

6) La posibilidad de añadir funcionalidades adicionales mediante la definición de nuevas funciones creadas por el usuario.

7) La posibilidad de ser extendido a través de paquetes. Hay alrededor de ocho paquetes básicos suministrados con la distribución de R, pero existen muchos más disponibles en el CRAN (*Comprehensive R Archive Network*).

Hay que reconocer que empezar a trabajar con R no es fácil, sobre todo para aquellos estudiantes no acostumbrados a utilizar paquetes informáticos de análisis numérico. Es decir, existe una cierta curva de aprendizaje que hay que superar mediante la práctica. Este módulo pretende, pues, servir de apoyo en esos siempre complicados primeros pasos, de manera que el estudiante se familiarice con este paquete estadístico y pueda aprovechar toda su potencialidad.

Por último, este módulo introduce la extensión R-Commander, que usaremos durante los diferentes módulos que forman estos materiales. R-Commander es actualmente una de las alternativas más viables a paquetes estadísticos comerciales, como Minitab y SPSS, de hecho es la más utilizada en docencia en la mayoría de las universidades del planeta. Permite ejecutar una parte de las posibilidades de R mediante un entorno con menús desplegables, ideal para estudiantes de estadística de grado. Además, el hecho de que permanentemente se muestre en pantalla el código subyacente hace de R-Commander una buena alternativa para la primera fase de aprendizaje del lenguaje de R.

Objetivos

1. Instalar R en el ordenador correctamente.

2. Aprender a instalar correctamente la aplicación R-Commander.

3. Conocer los principales componentes de R-Commander.

4. Instalar y cargar paquetes adicionales en función de las necesidades del análisis tanto con R como con R-Commander.

5. Utilizar las herramientas de ayuda que ofrecen R y R-Commander para resolver dudas.

6. Localizar en la web oficial del programa los manuales disponibles.

7. Identificar los principales tipos de datos y objetos admitidos en R, así como poder manejarlos eficientemente.

8. Crear un *script* para escribir un análisis de forma coherente, asignando nombres a nuevas variables.

9. Realizar operaciones matemáticas y estadísticas con vectores y matrices.

10. Dominar los operadores lógicos y las funciones recursivas, ciclos y condicionales.

11. En general, realizar todo tipo de operaciones matriciales.

1. Primeros pasos con R

1.1. ¿Qué es R?

Esta es una pregunta simple que no tiene una respuesta fácil. En su definición más amplia, R es un lenguaje de programación que permite al usuario programar algoritmos y usar herramientas programadas por otros. Si nos limitamos al análisis estadístico y cuantitativo, podemos afirmar sin miedo a exagerar que R no tiene límites, es decir, es capaz de hacer cualquier cosa que el usuario se pueda imaginar. Con R se pueden escribir funciones, hacer cálculos, aplicar técnicas estadísticas complejas e implementar técnicas *ad-hoc* (la manera más sencilla de aplicar técnicas complicadas es aprovechar las que ya están disponibles en la red y, en caso necesario, optar por desarrollarlas nosotros mismos), crear gráficos simples y complejos, e incluso escribir funciones propias. El hecho de que muchos centros de investigación y universidades lo estén usando genera que miles de contribuciones estén disponibles a todos los usuarios. Además, hay abundante material (libros, manuales, ejemplos, etc.) de alta calidad disponibles gratuitamente. Una ventaja nada despreciable es que, a diferencia de muchos programas estadísticos, R no tiene coste alguno, ya que es un proyecto GNU (software libre).

The R Project for Statistical Computing

En la página web oficial *http://www.r-project.org* encontraremos toda la información que necesitamos sobre este programa en inglés: archivos de instalación, características, paquetes disponibles, manuales, etc. Para encontrar manuales en castellano, deberemos ir a la página *http://cran.es.r-project.org/* e ir a la sección *Contributed*.

En este contexto, a modo de resumen las principales ventajas de R se pueden resumir en la siguiente lista:

- Es un programa distribuido libremente bajo una licencia GNU.
- Existe abundante material de apoyo: manuales, tutoriales y ejemplos.
- Además, existen librerías que cubren casi todas las metodologías de la estadística y las matemáticas (optimización, series temporales, programación lineal, ecuaciones diferenciales, inferencia, etc.).
- Permite sistematizar análisis largos y complejos en unas pocas instrucciones sintéticas, siendo así un programa muy eficiente.

¡Que no cunda el pánico! Es normal que al principio cueste un poco introducirse en el mundo de la programación, pero una vez que os acostumbréis a este nuevo entorno y lo dominéis, todo será mucho mas fácil.

No obstante, también podemos citar algunos inconvenientes:

- Existe una considerable barrera de entrada o curva de aprendizaje.
- No es el tipo de programa con ventanas y menús, como lo pueden ser Minitab, E-Views o SPSS. No obstante, existen extensiones de R, como R-Commander, que sí se basan en el uso de ventanas y menús y que, por tanto, minimizan este inconveniente.
- Aprender a usar R, al principio, requiere primero aprender a hacer programación.

1.2. Instalación

El primer paso obligado es la instalación del programa, y para ello tendremos que visitar la página web del proyecto:

http://www.r-project.org/

Una vez allí, vemos que hay varias secciones, una de las cuales es el centro de descargas *CRAN*. Si accedemos a él, tendremos en pantalla una lista de *mirrors*, es decir, servidores desde donde descargar los archivos. Seleccionaremos el servidor que esté más próximo geográficamente a nuestra ubicación y nos aparecerá una página con diversas opciones de instalación. La manera más rápida es seleccionar una de las siguientes opciones, que dependerá de nuestro sistema operativo:

Los vídeos son útiles

En internet podéis encontrar multitud de vídeos en inglés y en castellano sobre cómo instalar R. Muchos de estos están disponibles en *www.youtube.com*.

- Download R for Linux
- Download R for MacOS X
- Download R for Windows

Si tenéis un Mac o una distribución de Linux, deberéis tener en cuenta, además, la versión del sistema operativo que tenéis en vuestro ordenador.

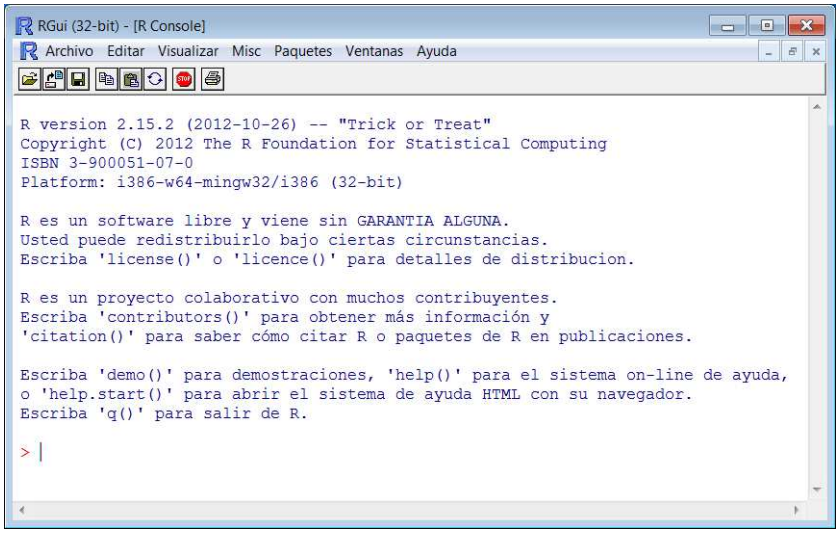
Si suponemos que tenemos un sistema operativo Windows, accedemos y descargamos la distribución base, a la cual se accede mediante la opción *install R for the first time*. Seguimos las instrucciones y descargamos el archivo ejecutable, tras lo que procedemos a la instalación sin más complicaciones.

1.3. Iniciar una sesión

Una vez instalado, ya podemos abrir el programa, ya sea mediante el acceso directo o mediante el archivo ejecutable incluido en la carpeta de instalación de R. De cualquier modo, al ejecutar R nos encontraremos con la siguiente ventana:

Es importante tener siempre instalada la última versión de R, ya que las nuevas versiones siempre incluyen mejoras. Además, nos ahorraremos problemas de compatibilidad con los últimos paquetes que instalemos.

Figura 1. Consola de R



Como podemos ver, las instrucciones se pueden introducir directamente en la consola, en el cursor que aparece tras el símbolo >. Una vez introducidas las instrucciones, pulsando *Enter* se ejecutarán los cálculos, igual que una calculadora:

```
> 2 * 3
[1] 6
```

Sin embargo, a la hora de realizar un análisis largo, lo más conveniente es escribir el conjunto de instrucciones con los cálculos que necesitamos en un archivo de texto, y esto se puede hacer de dos maneras:

- En un *script*, lo cual tiene la ventaja de que puede abrirse y guardarse como un archivo con formato *.R*, a través de la barra de menús del programa. Esta opción, que se muestra en la figura 2, permite ejecutar cada línea de código o toda una selección de líneas mediante la instrucción *CONTROL + R*, y aparecerá el resultado de color azul en la consola:

Figura 2. Consola de R con editor de texto

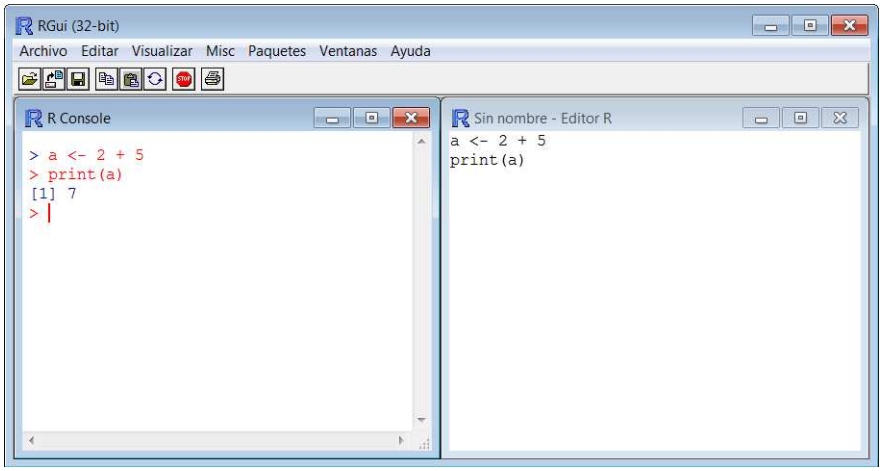


Figura 2

Fijaos en que R se basa en la asignación de objetos. Aquí vemos cómo el resultado de la operación 2 + 5 lo hemos asignado a un objeto al que hemos asignado el nombre de `a`.

- En un documento de texto aparte, donde una vez terminado se copia y se pega en la consola. El resultado será el mismo que el de la ventana izquierda de la figura 2, es decir, aparecerán las instrucciones y los resultados. Si optamos por esta opción, cualquier editor de texto servirá, aunque aquí recomendamos el editor *gedit*, ya que resalta la sintaxis de las instrucciones.

gedit

gedit es un editor de texto compatible con Linux, Mac OS X y Windows. Está diseñado como un editor de texto de propósito general y enfatiza la simplicidad y facilidad de uso. Además, incluye herramientas para la edición de código fuente y textos estructurados. Se puede descargar libremente desde la dirección *projects.gnome.org/gedit*

1.4. Gestión de paquetes

El programa R es modular, es decir, se basa en distintos módulos o paquetes (*packages*) que los usuarios han elaborado. Estos paquetes son unidades específicas que proporcionan funciones y datos para poner en práctica análisis específicos. Actualmente existen tantos paquetes que sería inviable cargarlos todos simultáneamente en

una sesión por motivos de memoria. La lista completa de paquetes se encuentra en la página del *CRAN*, en el apartado *Packages*. Allí podremos visualizar un listado de los más de 4.000 paquetes disponibles por orden alfabético con una breve descripción. Es muy útil saber que tenemos a nuestra disposición un listado por categorías: matemáticas, estadística, econometría, etc. Accediendo a cada paquete obtendremos más información acerca de este, así como distintos manuales de uso con las funcionalidades que incluyen.

En este sentido, cuando se abre una sesión de R se cargan automáticamente las funcionalidades básicas (denominadas *base*), y si el usuario está interesado en una metodología específica, deberá instalar el paquete correspondiente y cargarlo. Esto lo podemos hacer mediante el menú de la consola de R mostrado en la figura 1, en la opción *Paquetes*. Esta opción nos permite seleccionar un paquete de la lista que aparece, instalarlo y/o cargarlo. Otra opción es hacer esto manualmente utilizando la consola. Si conocemos el nombre del paquete, para instalarlo introduciremos:

Antes de intentar implementar nosotros mismos un complejo prodecimiento numérico o estadístico, vale la pena comprobar si alguien ya ha hecho el trabajo. Por eso mismo, es muy recomendable visitar el repositorio de paquetes de R (http://cran.r-project.org/), en el que podemos buscar paquetes por orden alfabético o por categorías.

```
>install.packages("paquete")
```

Una vez instalado, al principio de cada sesión en la que necesitemos este paquete bastará con introducir:

Una vez instalado un paquete, este ya queda almacenado en nuestro ordenador, de manera que no hay que volverlo a instalar. Solo tendremos que cargarlo en las sesiones que lo necesitemos.

```
>library(paquete)
```

Un análisis bien planificado incluirá al principio un listado de todos los paquetes que hay que cargar para llevar a cabo los análisis que deseemos hacer.

1.5. ¡Ayuda!

Para un usuario novel, la interfaz de R, la sintaxis y en general el modo de uso puede resultar complejo, especialmente en comparación con otros programas estadísticos. Sin embargo, R cuenta con muchos manuales y herramientas de apoyo y soporte al alcance de los usuarios. Para empezar, en la página del *CRAN* desde la que hemos descargado el archivo de instalación hay una sección llamada *Manuals*. Si accedemos a ella, tendremos acceso a los manuales oficiales de R, el primero de los cuales es *An Introduction to R*, un completísimo manual muy bien estructurado que cubre todos los aspectos elementales del programa. Si deseamos manuales en otros idiomas, en el enlace *contributed documentation* disponemos de una lista ordenada por idiomas, entre ellos el castellano.

Además, existe la posibilidad de obtener ayuda específica para cada paquete y función. Simplemente hay que introducir en la consola la instrucción `help()`, con el nombre del paquete o la función sobre la que necesitamos ayuda entre los símbolos de paréntesis, y se nos abrirá una ventana con una completa página de ayuda con descripciones y ejemplos.

¡Cuidado con las mayúsculas!

Hay que tener en cuenta que el lenguaje de R distingue entre mayúsculas y minúsculas. De esta manera, la instrucción `HELP()` resultaría en un mensaje de error.

2. Sintaxis y programación

2.1. Tipos de datos y objetos

R es un lenguaje orientado a objetos o estructuras de datos diferenciadas con características específicas. Los tres tipos básicos de datos son:

Es importante tener en cuenta que R interpreta de la misma manera las comillas simples 'x' que las dobles "x". Por tanto, podemos introducir las cadenas de texto de cualquiera de las dos maneras.

1) *Datos numéricos:* números reales y complejos $(-1, 2, 3)$.

2) *Caracteres:* cadenas de texto representadas entre comillas ("x", "y", "z").

3) *Datos lógicos:* aquellos que únicamente toman los valores verdadero o falso ($TRUE$, $FALSE$).

Esta orientación a objetos hace indispensable que conozcamos los diferentes tipos de objetos con los que podemos trabajar, ya que cada uno tiene características únicas. Sin embargo, como veremos en este módulo, R nos permite transformar algunos de ellos en otros (por ejemplo, convertir vectores en matrices y viceversa). Las principales clases de objetos están enumeradas y definidas en la tabla 1.

Tabla 1. Clases de objetos

Objeto	Descripción
Data frame	Estructuras de datos bidimensionales, donde se recogen diferentes variables por columnas.
Vector	Colección ordenada de datos con una longitud determinada.
Array	Conjunto de datos indexado por dos o más índices, caracterizado por tener dimensión.
Matrix	Caso particular de *array* cuando hay dos índices, es decir, dos dimensiones (filas y columnas).
List	Objeto que recoge varios tipos de elementos (componentes), que pueden ser de clases diferentes.
Function	Pieza de código que puede ser un algoritmo, una función matemática o una estructura lógica.
Factor	Vector que especifica una clasificación discreta de los elementos de otro vector de la misma longitud.

A continuación profundizaremos en cada una de estas estructuras.

2.2. Vectores

2.2.1. Almacenar datos en vectores

Hay dos maneras de introducir la información en el programa. Si introducimos información mediante expresiones sin asignarlas a ningún objeto, estamos utilizando el

programa como una simple calculadora. En cambio, si la almacenamos y le damos un nombre estamos guardando el resultado para utilizarlo posteriormente. Estas dos funcionalidades básicas quedan explicadas a continuación:

1) *Expresión:* la información se introduce en la consola, R la evalúa y la muestra en pantalla, pero no la almacena en la memoria.

```
> 4 + 3
[1] 7
```

2) *Asignación:* en este caso, la expresión se evalúa pero no se muestra, y se almacena con el nombre que escribamos antes del símbolo de asignación <−. Para ver su valor en pantalla, deberemos introducir el nombre que hemos dado al vector, que es la estructura de datos más simple[1]. La manera más inmediata de introducir los valores en un vector es mediante la instrucción `c`, que significa *concatenar*.

Cuestión de memoria

Cuando asignamos un nombre a un objeto, este quedará guardado en la memoria temporal. Esto es, podemos recuperar la información de dichos objetos siempre y cuando no cerremos la sesión de R.

```
> a <- c(1,2,3,4,5,6)
> a
[1] 1 2 3 4 5 6
```

Si introducimos una asignación entre paréntesis, además de quedar guardada en la memoria temporal, también se mostrará en pantalla.

```
> (b <- c(3,6,9))
[1] 3 6 9
```

Una vez tenemos varios vectores creados, los podemos combinar de una manera muy flexible. Veamos un ejemplo:

El espacio no existe

En el lenguaje de R los espacios no se interpretan. Es decir, las expresiones `a <- 6` y `a<-6` son análogas, no habiendo ninguna diferencia entre ellas.

```
> a <- c(1,2,3)
> b <- c(7,8,9)
> ab <- c(a,4,5,6,b)
> ab
[1] 1 2 3 4 5 6 7 8 9
```

2.2.2. Operaciones básicas

En la tabla 2 se recogen las operaciones aritméticas básicas tomando como ejemplo los vectores `v1` y `v2`. Estas operaciones están vectorizadas, es decir, se aplican elemento a elemento.

[1] La función `print` también muestra el resultado en pantalla.

Tabla 2. Operaciones aritméticas básicas (I)

```
v1 <- c(1,2,3)
v2 <- c(2,4,6)
```

Descripción	Instrucciones	Resultado
Concatenación	`c(v1,v2)`	`1,2,3,4,5,6`
Suma	`v1+v2`	`3,6,9`
Resta	`v1-v2`	`-1,-2,-3`
Multiplicación	`v1*v2`	`2,8,18`
División	`v1/v2`	`0.5,0.5,0.5`
Potencia	`v1^v2`	`1,16,729`

Cabe señalar que en R las comas quedan reservadas para la separación de los distintos valores o componentes de un mismo objeto o para los distintos parámetros de una función.

Cuando se intentan hacer operaciones matemáticas no definidas, como por ejemplo la raíz de un número negativo, aparecerá el valor `NaN`, que significa *Not a Number*. Además, los valores `± Inf.` aparecerán cuando el resultado tienda a $\pm\infty$. Veamos más operaciones matemáticas elementales aplicadas elemento a elemento en la tabla 3, tomando como ejemplo el vector `v3`:

Tabla 3. Operaciones aritméticas básicas (II)

```
v3 <- c(-1,0,1,2)
```

Descripción	Instrucciones	Resultado
Raíz cuadrada	`sqrt(v3)`	`NaN 0.00 1.00 1.41`
Valor absoluto	`abs(v3)`	`1 0 1 2`
Exponencial	`exp(v3)`	`0.36 1.00 2.71 7.38`
Log. base e	`log(v3)`	`NaN -Inf 0.00 0.69`
Log. base 10	`log10(v3)`	`NaN -Inf 0.00 0.30`
Factorial (!)	`factorial(v3)`	`NaN 1 1 2`

Tabla 3

Es fundamental tener en cuenta que, en R, los decimales se expresan mediante un punto, y no una coma como se hace en España. Confusiones en este sentido pueden llevar a errores.

Con R podemos realizar las operaciones estadísticas básicas. En la tabla 4 se recogen las más usuales, tomando como ejemplo los vectores `v4` y `v5`.

Tabla 4. Operaciones estadísticas básicas

```
v4 <- c(-2,3,5,0,0,3)
v5 <- c(20,14,51,76,21,30)
```

Descripción	Instrucciones	Resultado
Longitud	`length(v4)`	`6`
Máximo	`max(v4)`	`5`
Mínimo	`min(v4)`	`-2`
Suma	`sum(v4)`	`9`
Producto	`prod(v4)`	`0`
Media	`mean(v4)`	`1.5`
Mediana	`median(v4)`	`1.5`
Desviación estándar	`sd(v4)`	`2.58`
Varianza	`var(v4)`	`6.7`
Covarianza	`cov(v4,v5)`	`5.8`
Correlación	`cor(v4,v5)`	`0.09`
Producto escalar	`sum(v4,v5)`	`347`

2.2.3. Secuencias

Una herramienta muy útil es la creación de secuencias, lo cual se puede hacer de diferentes maneras, entre las cuales se encuentran las funciones `seq` (*sequence*), `rev` (*reverse*) y `rep` (*repeat*). La tabla 5 lo ilustra con algunos ejemplos, partiendo del vector `v6`:

Tabla 5

El dominio de estas instrucciones nos ahorrará muchas líneas de programación y nos permitirá programar complejos prodecimientos numéricos de una forma simple, compacta y elegante.

Tabla 5. Secuencias y repeticiones

```
v6 <- c(1,2,3)
```

Instrucciones	Resultado
`rep(v6,2)`	`1 2 3 1 2 3`
`rep(v6,each=2)`	`1 1 2 2 3 3`
`rep(v6,each=2,times=2)`	`1 1 2 2 3 3 1 1 2 2 3 3`
`rep(v6,c(2,3,4))`	`1 1 2 2 2 3 3 3 3`
`1:10`	`1 2 3 4 5 6 7 8 9 10`
`seq(1,10)`	`1 2 3 4 5 6 7 8 9 10`
`10:1`	`10 9 8 7 6 5 4 3 2 1`
`seq(10,1)`	`10 9 8 7 6 5 4 3 2 1`
`seq(4,length=8)`	`4 5 6 7 8 9 10 11`
`seq(0,40,by=5)`	`0 5 10 15 20 25 30 35 40`
`seq(0,1,length=4)`	`0.00 0.33 0.66 1.00`
`seq(from=2,to=8,by=3)`	`2 5 8`
`rev(1:5)`	`5 4 3 2 1`
`2*1:4`	`2 4 6 8`

2.2.4. Operadores lógicos

R también nos permite operar con vectores lógicos que pueden tomar los valores `TRUE` (verdadero) y `FALSE` (falso). Los principales operadores se muestran en la tabla 6:

Tabla 6

Podemos interpretar el operador lógico intersección como a y b, mientras que la unión sería interpretable como a o b.

Tabla 6. Operadores lógicos

Descripción	Operador
Menor	`<`
Mayor	`>`
Igual	`==`
No igual	`!=`
Menor o igual	`<=`
Mayor o igual	`>=`
Intersección	`a & b`
Unión	`a|b`
Negación	`!a`

Veamos algunos ejemplos del uso de estos operadores:

```
> a <- c(-2,-1,0,1,2)
> a >2
[1] FALSE FALSE FALSE FALSE FALSE
> a <= -1
[1] TRUE TRUE FALSE FALSE FALSE
> a >0 & abs(a) == 2
[1] FALSE FALSE FALSE FALSE TRUE
> a >0 | abs(a) == 2
[1] TRUE FALSE FALSE TRUE TRUE
```

Los símbolos = y ==

Cuando usamos el operador lógico igualdad, es importante tener en mente que se expresa mediante un doble símbolo de igualdad (==), diferente del símbolo de igualdad simple utilizado en asignaciones.

2.2.5. Indexación

Para acceder a los elementos de un vector, existe la posibilidad de *indexar*, es decir, acceder a los valores de un vector especificándolo en una expresión entre corchetes `[]`. Esto lo hacemos indicando la posición de los elementos o usando operadores lógicos. Un ejemplo considerando el vector `v7` se recoge en la tabla 7:

Tabla 7. Indexación de vectores

`v7 <- c(-3,-2,-1,0,1,2,3)` o de forma más compacta `v7 <- c(-3:3)`

Expresión	Resultado	Descripción
`v7[1]`	`-3`	Primer elemento
`v7[2:4]`	`-2 -1 0`	Del segundo al cuarto elemento
`v7[-(2:length(v7))]`	`-3`	Todos menos aquellos desde el segundo hasta el último elemento
`v7[c(2,4,6)]`	`-2 0 2`	Elementos segundo, cuarto y sexto
`v7[v7>0]`	`1 2 3`	Elementos mayores que 0
`v7[v7!=0]`	`-3 -2 -1 1 2 3`	Elementos diferentes de 0
`v7[abs(v7)==3]`	`-3 3`	Elementos con valor absoluto 3
`v7[v7>0 & v7!=2]`	`1 3`	Elementos positivos **y** que no sean 3
`v7[v7<0 | v7==3]`	`-3 -2 -1 3`	Elementos negativos **o** que sean 3

Además, la función `which` nos permite extraer la posición ordinal de los elementos descritos en la condición introducida. En el siguiente ejemplo, en el primer caso los elementos del vector `a` iguales a cero son el primero, tercero y sexto. En el segundo caso, el único elemento mayor a cero (`a>0`) y menor o igual a dos (`a<=2`) es el cuarto.

```
> a <- c(0,3,0,1,3,0)
> which(a==0)
[1] 1 3 6
> which(a>0 & a<=2)
[1] 4
```

De igual modo, las funciones `which.min` y `which.max` nos dan la posición de los valores mínimos y máximos del vector, respectivamente.

2.3. Matrices

2.3.1. Los objetos `array` y `matrix`

A menudo tendremos la necesidad de almacenar los datos en estructuras más complejas que simples vectores. Esto es, podemos necesitar tener una estructura de filas y columnas, así como poder realizar operaciones matriciales. En R, hay dos estructuras (tipos de objetos) que satisfacen esta necesidad:

1) Variable indexada (`array`): estructura que organiza los datos en k dimensiones, y es muy útil para tratar con datos con diversas clasificaciones simultáneas. Por ejemplo, la variable v_{ijt} puede representar las ventas de una empresa donde i es el sector, j la provincia y t el año, y se almacenaría en una estructura `array` con dimensión $k = 3$.

2) Matriz (`matrix`): se trata de un caso particular de *array* cuando hay dos dimensiones ($k = 2$), siendo la primera dimensión las filas y la segunda dimensión las columnas.

En muchos casos no hay diferencia entre los dos tipos de estructuras. Sin embargo, algunas operaciones, como la transposición, y algunos tipos de operaciones algebraicas están reservadas solo a las matrices. Por simplicidad, en lo que sigue nos centraremos en la creación y manipulación de estructuras bidimensionales (matrices). La manera más inmediata de crear una matriz es asignándole a un vector una dimensión, ya que, como hemos visto, los vectores tienen longitud y no dimensión:

```
> a <- 1:8
> a
[1] 1 2 3 4 5 6 7 8
> length(a)
[1] 8
> dim(a)
NULL
```

El resultado `NULL`

Siempre que un objeto esté vacío aparecerá el resultado `NULL`. En este caso, al ser `a` un vector y no una matriz, su dimensión no existirá.

2.3.2. Creación de matrices

Supongamos que deseamos convertir el vector `a` en una matriz de dimensión 4×2, es decir, una estructura con 4 filas y 2 columnas. En este caso, podemos asignarle una dimensión:

```
> dim(a)<-c(4,2)
> a
     [,1] [,2]
[1,]    1    5
```

```
[2,]    2    6
[3,]    3    7
[4,]    4    8
> dim(a)
[1] 4 2
```

Es importante destacar que, por defecto, R siempre empieza a colocar los valores de arriba a abajo y de izquierda a derecha, como se puede observar en el ejemplo anterior.

Una manera más directa de crear matrices es mediante la función `matrix`. Esta función tiene tres argumentos: el conjunto de valores que compondrán la matriz, el número de filas y el número de columnas:

```
> matrix(1:8,2,4)
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
```

En el caso de necesitar una estructura de datos con más de dos dimensiones, necesitaremos la función `array`. La principal diferencia respecto a la función `matrix` es que el número de dimensiones se introduce mediante un vector. Crearemos a modo de ejemplo una estructura $2 \times 2 \times 2$:

```
> array(1:4,c(2,2,2))

, , 1
     [,1] [,2]
[1,]    1    3
[2,]    2    4

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

El reciclaje es importante

Fijaos en que cuando se acaban de colocar todos los elementos del vector `1:4` en la estructura resultante y esta no está completa todavía, se vuelve a empezar de nuevo hasta que la estructura está completa. Esta funcionalidad se denomina *reciclaje*.

Una característica interesante de la construcción de matrices es el **reciclaje**, que consiste en que si el número total de elementos de la matriz (filas por columnas) es superior al número de valores iniciales, estos se repetirán hasta completar la matriz. En el ejemplo anterior, hemos querido transformar un vector con 4 elementos en una estructura con $2 \times 2 \times 2 = 8$ elementos, esto es, el doble. Por eso, R ha tenido que repetir los elementos del vector inicial dos veces para completar la estructura final. Veamos otro ejemplo de reciclaje:

```
> matrix(1:3,3,3)
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    2    2
[3,]    3    3    3
```

Además existe la opción de que la matriz se construya invirtiendo el orden, es decir, de izquierda a derecha y de arriba a abajo. Se consigue añadiendo una opción al final de la función:

```
> matrix(1:8,2,4,byrow=TRUE)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
```

¿Por filas o columnas?

Mediante el uso de la función `byrow`, invertimos el orden de la construcción de matrices, empezando por filas en lugar de por columnas, que es la opción por defecto.

También es posible especificar únicamente el número de filas y/o columnas de la matriz que queremos construir:

```
> matrix(1:6,nrow=2)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> matrix(1:4,ncol=2)
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Es muy fácil combinar escalares, vectores y matrices. La función `rbind` une varias estructuras por filas, mientras que `cbind` hace lo propio por columnas. Veamos cómo se hace:

```
> a<-matrix(0,2,2)
> a
     [,1] [,2]
[1,]    0    0
[2,]    0    0
> b<-matrix(1,1,2)
> b
     [,1] [,2]
[1,]    1    1

> rbind(a,b)
     [,1] [,2]
```

```
[1,]    0    0
[2,]    0    0
[3,]    1    1
```

Como hemos visto, estas dos funciones nos ofrecen muchas posibilidades:

```
> cbind(1,1:4)
     [,1] [,2]
[1,]    1    1
[2,]    1    2
[3,]    1    3
[4,]    1    4
```

Otro ejemplo de reciclaje

En este caso, vemos que unir un escalar y un vector provoca que el valor del escalar se repita tantas veces como la longitud del vector. Esto hace que la estructura resultante tenga dos columnas y tantas filas como elementos del vector.

La matriz identidad se crea mediante la función `diag`, especificando el número de filas y columnas:

```
> diag(2)
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

La matriz identidad

Recordemos que la matriz identidad siempre es cuadrada, esto es, tiene el mismo número de filas y de columnas.

Además, la función `diag` también se puede usar para acceder a los elementos de la diagonal principal de una matriz:

```
> a<-matrix(5*1:4,2,2)
> a
     [,1] [,2]
[1,]    5   15
[2,]   10   20
> diag(a)
[1]  5 20
```

2.3.3. Submatrices e indexación

Análogamente al caso de los vectores, también es posible acceder a subconjuntos de matrices mediante corchetes `[]`. Estos se usan con tres propósitos:

1) Tener acceso a filas y columnas específicas.
2) Crear submatrices.
3) Modificar directamente partes de una matriz.

Esto nos ofrece una gran potencialidad para hacer todo tipo de operaciones con matrices, y hacer que estas sean fácilmente replicables, modificables y combinables. A partir de la matriz `a`, la tabla 8 muestra ejemplos de creación de submatrices.

```
> a <- matrix(1:9*3,3,3)
> a
     [,1] [,2] [,3]
[1,]    3   12   21
[2,]    6   15   24
[3,]    9   18   27
```

Tabla 8. Indexación de matrices

Expresión	Resultado	Descripción
`a[,3]`	`21 24 27`	Todas las filas, 3.ª columna
`a[2,]`	`6 15 24`	2.ª fila, todas las columnas
`a[-1,-3]`	`6 15` `9 18`	Todas las filas menos la 1.ª y todas las columnas menos la 3.ª
`a[2:3,1]`	`6 9`	De la 2.ª a la 3.ª fila, primera columna

Es interesante comprobar que las submatrices creadas, en caso de tener una sola fila o columna, se degradan a vectores, con lo que pierden la dimensión[2]. Para que estas submatrices sigan teniendo dimensión, debemos especificarlo desactivando la opción `drop`:

```
> a
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> b <- a[,1,drop=FALSE]
> b
     [,1]
[1,]    1
[2,]    2
> dim(b)
[1] 2 1
```

Veamos ahora cómo se haría para introducir elementos nuevos en una matriz. Supongamos, por ejemplo, que queremos sustituir la primera columna de valores por una columna de ceros:

```
> a<-matrix(1:8,2,4)
> a[,1]<-0
> a
     [,1] [,2] [,3] [,4]
[1,]    0    3    5    7
[2,]    0    4    6    8
```

Modificar una matriz

En este caso, hemos utilizado `a[,1]<-0` para transformar la primera columna, inicialmente compuesta por los elementos 1 y 2, en una columna con dos ceros.

[2] En inglés, este hecho se denomina *deprecation*.

2.3.4. Operaciones matriciales básicas

R ofrece muchas posibilidades para aplicar operaciones y funciones a vectores y matrices[3]. En la tabla 9 se incluyen las funciones básicas aplicables a matrices:

Tabla 9. Funciones con matrices

Descripción	Función
Dimensiones	`dim`
Transpuesta	`t`
Determinante	`det`
Inversa	`solve`
Valores y vectores propios	`eigen`
Valores singulares	`svd`
Factorización de Choleski	`chol`
Factorización QR	`qr`
Rango	`rank`
Número de filas	`nrow`
Número de columnas	`ncol`
Suma de las filas	`rowSums`
Suma de las columnas	`colSums`
Media de las filas	`rowMeans`
Media de las columnas	`colMeans`

Para acabar el apartado dedicado a matrices, vale la pena mencionar que hay algunas funciones que permiten, por una parte, identificar objetos y, por otra, transformarlos en objetos de otras clases. Este primer ejemplo es ilustrativo:

```
> a <- c(1,3,6)
> is.matrix(a)
[1] FALSE
> is.vector(a)
[1] TRUE
> class(a)
[1] 'numeric'
```

La familia de funciones `is.` responde a la pregunta de si se trata de un objeto determinado mediante un valor lógico, mientras que la función `class` devuelve la clase del objeto. Finalmente, la familia de funciones `as.` se utiliza para transformar un tipo de objeto en otro. El siguiente ejemplo lo muestra:

```
> (A <- matrix(1,2,2))
     [,1] [,2]
[1,]    1    1
[2,]    1    1
> (a <- as.vector(A))
[1] 1 1 1 1
```

Convertir matrices en vectores

En este caso, hemos conseguido transformar la matriz `A` en el vector `a`.

[3] El módulo 2 de este manual incluye un apartado de álgebra vectorial y matricial dedicado a operaciones algebraicas específicas.

2.4. Funciones

Una herramienta muy útil y poderosa es el comando `function`, que también es una clase de objeto. Podemos crear desde simples funciones matemáticas a complejos algoritmos con los que realizar multitud de cálculos de una manera estructurada y sintética. La forma más sencilla de función tiene la siguiente estructura:

```
nombre <- function (argumentos) expresión
```

Como ejemplo, supongamos que queremos estudiar la función matemática $y = x^3$ y evaluarla en el vector de valores iniciales $x_0 = (-4, -3, \ldots, 3, 4)$. Los haremos creando nuestra propia función `cubic` y evaluándola en el vector de valores inciales `x0`:

```
> cubic <- function(x) x^3
> x0 <- (-4:4)
> cubic(x0)
[1] -64 -27  -8  -1   0   1   8  27  64
```

Un ejemplo de una función con más de un argumento es el de la fórmula financiera del interés simple $C_f = C_i\,(1 + rt)$. Según esta fórmula, el capital final (C_f) que hay que retornar en una operación financiera con un importe inicial C_i depende del tiempo de duración de la operación en años (t) y del tipo de interés en tanto por uno (r). ¿Qué importe final resulta de una operación a nueve meses ($t = 0,75$) con un tipo nominal del 5 % ($r = 0,05$) y capital inicial $C_i = 1000$?

```
> op.fin <- function(Ci,r,t) Ci*(1+r*t)
> op.fin(1000,0.05,0.75)
[1] 1037.5
```

Para funciones más complejas a menudo es necesaria más de una línea para introducir las fórmulas. En este caso, suponiendo que necesitamos n líneas, la estructura será la siguiente:

```
nombre <- function(argumentos){
expresión 1
...
expresión n
return(resultado)
}
```

Con el fin de ilustrar este tipo de estructura crearemos una función para el cálculo la covarianza muestral. Esto es un tanto innecesario, ya que podríamos usar la función `cov` que nos proporciona R. No obstante, lo hacemos para que sirva de ejemplo de funciones más elaboradas:

$$Cov(X,Y) = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})$$

```
> mi.covar <- function(x,y){
+     N <- length(x)
+     demean.x <- x-mean(x)
+     demean.y <- y-mean(y)
+     sumat <- sum(demean.x*demean.y)
+     return(sumat/(N-1))
+ }
```

Funciones multiusos

Crear funciones propias nos será muy útil para realizar un conjunto de cálculos que tengamos que usar en más de una ocasión, de manera estructurada y ordenada.

2.5. Ciclos y condicionales

R ofrece una serie de herramientas para hacer asignaciones múltiples y condicionales. `ifelse` permite realizar operaciones elemento por elemento a un vector sujeto a una condición. Supongamos que queremos aplicar a un vector la siguiente transformación:

$$f(x) = \begin{cases} log(x) & \text{si} \quad x > 0 \\ 0 & \text{si} \quad x \leq 0 \end{cases}$$

```
> a <- c(-1,0,2,5)
> ifelse(a>0,log(a),0)
[1] 0.0000000 0.0000000 0.6931472 1.6094379
```

Ciclos y condicionales

Para usar eficientemente ciclos y condicionales es fundamental dominar los operadores lógicos, explicados en este módulo.

Uno de los componentes fundamentales de cualquier lenguaje de programación es el uso de estructuras iterativas, es decir, funciones que repiten una o más expresiones iterativamente en un ciclo. El principal comando que hace esta función es `for`. La estructura general de esta función es, para un ciclo de *n* iteraciones:

```
for (i in 1:n){
expresión(es)
...
}
```

Un sencillo ejercicio para ilustrar un ciclo es la suma acumulativa. Partiendo de un vector $\vec{v} = (v_1, \cdots, v_n)$, crearemos un vector $\vec{s} = (s_1, \cdots, s_n)$ donde $s_j = \sum_{i=1}^{j} v_i$. Antes de implementar el ciclo crearemos un vector s vacío que iremos completando iterativamente:

```
> v <- 1:10
> n <- length(v)
> s <- rep(0,n)

> for (i in 1:n){
+     p <- v[1:i]
+     s[i] <- sum(p)
+ }

> print(s)
 [1]  1  3  6 10 15 21 28 36 45 55
```

Ciclos iterativos

Utilizaremos los ciclos cuando queramos trabajar con bucles o hacer cálculos iterativos.

Obviamente, en un análisis real no usaremos este cálculo iterativo, ya que la función implementada `cumsum` ya nos ofrece esta funcionalidad de una manera más eficiente.

2.6. La familia `apply`

En la práctica, es importante no hacer un uso excesivo de los ciclos iterativos, ya que son ineficientes en cuanto al uso de la memoria y pueden ralentizar el tiempo requerido de análisis. Por ello, la norma es que se crearán estructuras iterativas siempre que no haya alternativas, y el grupo de funciones `apply` es una de ellas.

Para ilustrar este concepto, supongamos que tenemos una matriz A donde las filas corresponden a observaciones y las columnas son las variables X, Y y Z:

$$A = \begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_n & y_n & z_n \end{pmatrix}$$

La función `apply` nos permite aplicar cualquier operación a las filas y a las columnas, independientemente. Supongamos que deseamos aplicar la operación $\sqrt{x_i^2 + y_i^2 + z_i^2}$ a cada observación (es decir, a cada fila). El primer paso será definir esta operación (a la que llamaremos `hipot`) y el segundo, aplicarla a las filas de la matriz A:

```
> hipot <- function(x) sqrt(sum(x*x))
> apply(A,1,hipot)
```

La función `apply`

La instrucción `apply(A,1,hipot)` se lee como *aplica a la dimensión 1 (filas) de la matriz A la operación* `hipot`. El resultado será un vector con n elementos.

Si deseáramos un vector con la media aritmética de cada columna, es decir $(\bar{x}, \bar{y}, \bar{z})$, deberíamos introducir:

```
> apply(A,2,mean)
```

Para explicar la funcionalidad de la función de la misma familia `tapply`, es necesario introducir los **factores**, que son una clase de objeto que establece una clasificación discreta de una o más variables. Supongamos que estamos estudiando la estatura de los alumnos de una escuela, para lo que disponemos de dos vectores: uno con la estatura de cada estudiante y otro con el género (masculino *m* y femenino *f*):

```
> altura <- c(131,125,126,140,152,119)
> genero <- c('m','m','f','m','f','f')
```

Introducción de caracteres

A la hora de introducir un vector de caracteres, como `genero`, los valores del vector han de ir entre comillas, para lo cual son válidos los símbolos `"` y `'` indistintamente.

En este caso es conveniente convertir el vector `genero` en un factor con dos categorías: masculina y femenina. Para ello, empleamos la función `factor` para obtener un vector de factores con las categorías $f = 1$ y $m = 2$, al que llamaremos `f.genero`. Veamos cómo son objetos de diferente clase:

```
> f.genero <- factor(genero)
> class(genero)
[1] "character"
> class(f.genero)
[1] "factor"
```

La instrucción `levels` devuelve las categorías que el vector incluye:

```
> levels(f.genero)
[1] 'f' 'm'

> cbind(f.genero,altura)
     f.genero altura
[1,]        2    131
[2,]        2    125
[3,]        1    126
[4,]        2    140
[5,]        1    152
[6,]        1    119
```

Un análisis estadístico de la variable *altura* deberá diferenciar entre *f* y *m*, para lo cual se emplea la función `tapply`. Calcularemos la media aritmética y la desviación estándar de la altura diferenciando por género:

```
> tapply(altura,f.genero,mean)
       f        m
132.3333 132.0000

> tapply(altura,f.genero,sd)
        f         m
17.387735  7.549834
```

La función `tapply`

Utilizaremos la función `tapply` cuando queramos aplicar un cálculo a una variable diferenciando por grupos o segmentos, esto es, según las categorías de un factor.

2.7. Bases de datos

Una base de datos (*data frame*) es similar a una matriz, ya que es una estructura bidimensional donde las filas son las observaciones y las columnas son las variables, cada una con un nombre específico. Es muy recomendable tener las variables almacenadas en este formato por varios motivos:

- Es muy fácil extraer subgrupos (*subsets*) de las variables usando operadores lógicos.
- Muchas funciones estadísticas, como los estimadores econométricos, admiten datos solo en este formato.
- Es posible extraer variables de la base de datos mediante la función `attach`.

Hay varias maneras de crear una base de datos. La primera es importar datos externos de un archivo de texto plano[4]. La segunda es agrupando vectores existentes, como mostramos en el siguiente ejemplo (continuación del anterior):

```
> g <- c('m','m','f','m','f','f')   # Género
> v1 <- c(131,125,126,140,152,119)  # Altura
> v2 <- c(48,53,45,40,49,50)        # Peso
>
> datos <- data.frame(genero=g,altura=v1,peso=v2)
> datos
  genero altura peso
1      m    131   48
2      m    125   53
3      f    126   45
4      m    140   40
5      f    152   49
6      f    119   50
```

Incorporar comentarios mediante el símbolo #

Muy a menudo nos interesará introducir aclaraciones y comentarios en nuestras líneas de código para entenderlo mejor. Para que R no los interprete y nos dé un mensaje de error, los introduciremos después del símbolo #.

[4] En el módulo dedicado al análisis estadístico descriptivo se explica este procedimiento con detalle.

Como vemos, en la primera columna se identifica cada elemento de la muestra, mientras que cada columna está encabezada por el nombre de la variable. Es importante destacar que, alternativamente, la función `data.frame` también admite una matriz.

Supongamos que queremos hacer un análisis solo de aquellos chicos (*m*) por debajo de 50 kilos. Para ello, es muy práctico crear otra base de datos que sea un subgrupo de la base de datos original. Llamaremos a este subgrupo `datos2`, y que como vemos solo incluye dos observaciones, las que satisfacen las condiciones establecidas:

```
> datos2<-subset(datos,genero=="m" & peso<50)
> datos2
  sexo altura peso
1    m    131   48
4    m    140   40
```

La función `subset`

La función `subset` nos permite crear una nueva base de datos que contenga una selección condicional de observaciones de la base de datos original.

Si pretendemos operar con las variables incluidas en una base de datos, deberemos referenciarlas con el símbolo $, ya que las variables no están en el espacio de trabajo individualmente, solo como elementos de una base de datos. Por ejemplo, si necesitamos la correlación entre la altura y el peso, deberemos introducir la siguiente instrucción:

```
> cor(datos$altura,datos$peso)
[1] -0.3202351
```

Esto puede llegar a ser algo engorroso. Por ello, si la intención es hacer operaciones con las variables de la base de datos, es recomendable *volcarlas* en el espacio de trabajo previamente mediante la función `attach`, con lo cual ya podremos hacer referencia a las variables individualmente:

```
> attach(datos)
> cor(altura,peso)
[1] -0.3202351
```

La función `attach`

Esta función será aplicable cuando queramos realizar operaciones y cálculos individuales con las variables incluidas en una base de datos.

2.8. Listas

Las listas (*lists*) son comparables a un cajón de sastre, ya que permiten almacenar objetos de clases diferentes en una misma estructura. Esto es muy útil cuando un mismo análisis tiene como resultado diferentes objetos dispares, por ejemplo vectores o matrices de diferente longitud o dimensión. Supongamos que deseamos almacenar en una estructura la matriz *A*, su descripción, su inversa y el determinante:

```
> descr <- c('Matriz A')
> A <- matrix(c(3,5,6,1),2,2)
> inv <- solve(A)
> deter <- det(A)
```

Las listas

Esta clase de objeto nos permitirá agrupar en una sola estructura elementos que son, a su vez, objetos de clases diferentes.

La forma de almacenarlo es mediante la función `list`. Para acceder a los diferentes elementos de una lista mediante su nombre, usaremos el símbolo $ de la siguiente manera:

```
> MatA <- list(descripcion=descr,
+              matriz=A,
+              inversa=inv,
+              deter=deter)

> MatA$descripcion
[1] 'Matriz A'

> MatA$matriz
     [,1] [,2]
[1,]    3    6
[2,]    5    1

> MatA$inversa
            [,1]       [,2]
[1,] -0.03703704  0.2222222
[2,]  0.18518519 -0.1111111

> MatA$deter
[1] -27
```

Los símbolos > y + en la consola

Como hemos visto, en la consola hemos de introducir cada línea de código tras el símbolo `>`. Sin embargo, algunas instrucciones de código son tan largas que ocupan más de una línea, como en el caso de la lista `MatA`. En este caso, R nos mostrará en la siguiente línea el símbolo `+` para recordarnos que la instrucción de la línea previa está incompleta.

También se puede acceder a los elementos de una lista no por su nombre, sino ordinalmente. Para ello usaremos el símbolo [[]]. Veamos una comparación de ambas maneras de acceder a elementos de la lista:

```
> MatA$matriz
     [,1] [,2]
[1,]    3    6
[2,]    5    1

> MatA[[4]]
[1] -27
```

Diferentes maneras de acceder a los elementos de una lista

Fijaos en que, en este ejemplo, las instrucciones `MatA$matriz` y `MatA[[2]]` darían el mismo resultado.

3. La extensión R-Commander

3.1. Introducción

Uno de los principales problemas a los que se ha enfrentado R desde su nacimiento es la importante curva de aprendizaje que el usuario ha de afrontar para poder usarlo. De hecho, esto es un problema a la hora de implementar este programa en los primeros cursos de estadística de grado, ya que a la dificultad de aprender el contenido de la asignatura se suma la complicación del lenguaje de R. Por suerte, este programa es tan flexible y maleable que se han creado varias aplicaciones basadas en R pero que son más fáciles de utilizar. Una de las aplicaciones con más éxito ha sido R-Commander, que consiste en una interfaz gráfica de usuario (GUI, en sus siglas en inglés). Su creador, el prof. John Fox, lo desarrolló con el objetivo de crear un programa con *menús desplegables*, y que fuera fácil de usar para estudiantes de estadística. Se puede considerar a R-Commander como una alternativa viable a los paquetes estadísticos comerciales (como Minitab o SPSS).

RStudio

Para los usuarios de R avanzados, la extensión RStudio es una potente herramienta consistente en un entorno de desarrollo integrado (IDE), en el que se visualizan el documento con el código, la ventana de resultados, una lista con los objetos creados y un espacio para los gráficos. En este sentido, es similar al entorno de trabajo que ofrece el programa de análisis matemático Matlab.

Otro aspecto destacable de R-Commander es que resulta muy útil para aquellos estudiantes que aprenden a utilizar R. Esto es así porque, al efectuar el análisis a través del menú desplegable, el código en R subyacente también aparece en pantalla junto al resultado, de manera que, al final, el estudiante acabará introduciendo el código directamente.

3.2. Instalación

Para instalar R-Commander hemos de tener instalada una versión de R. En general, como para todos los paquetes hay dos opciones de instalación:

1) Ir al menú desplegable de R, seleccionar *Paquetes* e *Instalar paquetes*. Una vez allí, seleccionamos un servidor (por proximidad podríamos elegir Spain (Madrid), France (Toulouse) o Portugal), y pulsamos *Ok*. A continuación nos aparecerá una lista de todas las librerías (paquetes o *"packages"* en inglés) que podemos instalar, y seleccionamos el paquete *Rcmdr*.

2) Como haríamos con cualquier otro paquete que deseáramos instalar, podríamos ir directamente a la consola de R e introducir `install.packages`(”`Rcmdr`”).

Ejecutar el programa más de una vez

Si cerramos el programa R-Commander, volveremos a la consola de R. Si entonces deseamos volver a cargar R-Commander sin tener que cerrar la sesión de R y abrirla de nuevo, deberemos introducir la instrucción `Commander()` en la consola de R.

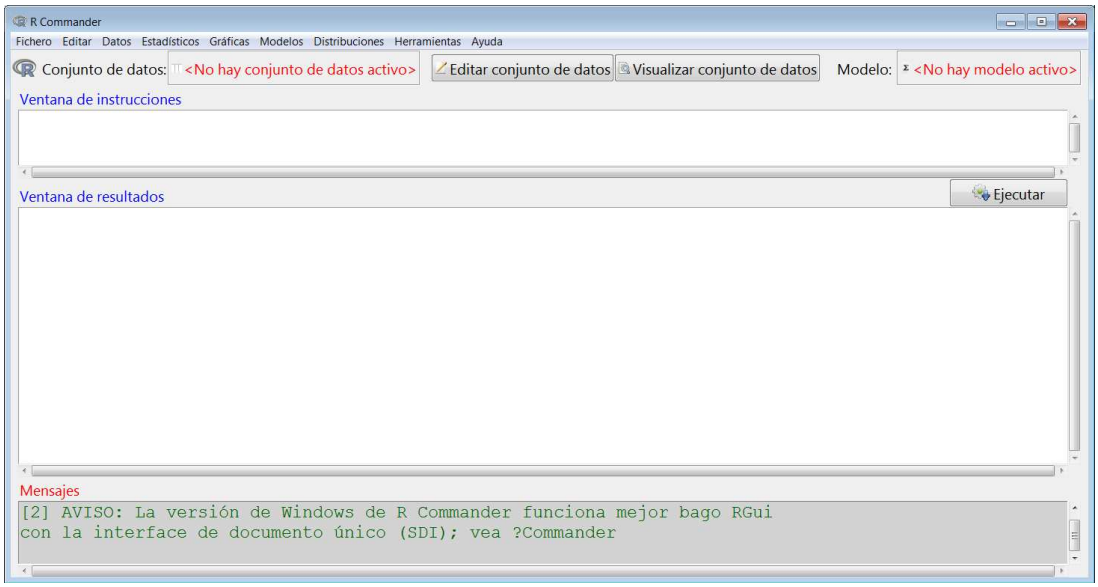
Una vez acabado el proceso de descarga, la instalación se completa introduciendo en la consola la instrucción `library`(`Rcmdr`). Entonces aparecerá una ventana de aviso indicándonos que hay que instalar algunos paquetes adicionales. Pulsamos en *Aceptar*

y esperamos a que se termine el proceso. Si todo ha ido bien, al acabar nos aparecerá en pantalla la ventana de R-Commander, lista para poder usarla. En general, siempre que queramos usar R-Commander deberemos introducir la instrucción `library(Rcmdr)` en la consola.

3.3. Componentes

La ventana principal de R-Commander se muestra en la figura 3 a continuación:

Figura 3. La interfaz gráfica R-Commander



A lo largo de los módulos de este material iremos analizando cada uno de los elementos de este programa. Sin embargo, en esta sección veremos una visión general de todos los componentes.

3.3.1. Barra de menús

Esta barra se encuentra en la línea superior e incorpora las principales funcionalidades del programa. A continuación se describen sus elementos:

1) Fichero. Instrucciones para crear, cargar y guardar documentos con código (*scripts*), resultados y espacios de trabajo (*workspaces*), así como para salir del programa.

2) Editar. Opciones de cortar, copiar y pegar, y en general para editar el contenido de las ventanas de instrucciones y de resultados.

3) Datos. Submenús que contienen elementos de menú para importar, leer y manipular datos y variables.

4) Estadísticos. Submenús que contienen elementos de menú para una variedad de análisis estadísticos básicos.

5) Gráficas. Elementos de menú para crear gráficos estadísticos simples.

6) Modelos. Elementos para obtener todo tipo de información sobre la estimación de modelos estadísticos: resúmenes numéricos, intervalos de confianza y contrastes de hipótesis, diagnósticos, análisis de residuos y gráficos derivados de un modelo estadístico.

7) Distribuciones. Contiene las principales distribuciones de probabilidad discretas y continuas, incluyendo probabilidades, cuantiles, gráficos y generación de datos aleatorios.

8) Herramientas. Permite cargar paquetes adicionales que no vienen en la distribución básica de R-Commander, y se pueden cargar a modo de extensiones si se necesita realizar un análisis específico. Además, en esta parte del menú también están las opciones de visualización (tamaño de letra, etc.).

9) Ayuda. Permite acceder a varios documentos de ayuda de R-Commander muy completos, de lectura recomendada a los no iniciados en el programa.

Un elemento fundamental que se debe tener en cuenta es que R-Commander solo incluye una pequeñísima parte de las potencialidades de R. Es decir, R-Commander ha sido diseñado para satisfacer las necesidades de estudiantes de cursos de estadística introductorios e intermedios. Aquellos usuarios que necesiten realizar análisis estadísticos y cuantitativos más avanzados necesitarán más funciones de las que se incluyen en la barra de menús.

3.3.2. Barra de herramientas

Esta barra incluye dos elementos básicos: el conjunto de datos y el modelo.

1) Conjunto de datos. El conjunto incluye una serie de datos, distribuidos en variables por columnas. Hay dos botones que permiten *editar* y *visualizar* el conjunto de datos. Un aspecto que se debe tener en cuenta es que se puede trabajar con varios conjuntos de datos simultáneamente, pero solo puede haber *un conjunto de datos activo* en cada momento. En el espacio de la figura 3 donde se lee *<No hay conjunto de datos activo>* aparecerá el menú desplegable con los diferentes conjuntos de datos existentes, de entre los que elegiremos, en cada momento, el conjunto de datos activo que deseemos analizar.

2) Modelo. El funcionamiento de este espacio es análogo al anterior. Esto es, podemos estimar diferentes modelos en cada sesión de trabajo, pero solo puede haber un modelo activo en cada momento, ya que la funcionalidad del menú desplegable se aplica siempre sobre el modelo activo.

3.3.3. Ventanas de instrucciones, resultados y mensajes

La **ventana de instrucciones** tiene como objetivo desplegar las instrucciones en código de R. Es importante explicar su funcionamiento: cuando realizamos un análisis mediante las opciones disponibles en el menú desplegable, el código correspondiente aparecerá igualmente en la ventana de instrucciones, y el resultado aparecerá en la **ventana de resultados**. Ahora bien, también es posible introducir las instrucciones *directamente* en la ventana de instrucciones sin necesidad de acudir al menú desplegable, seleccionar las líneas introducidas y pulsar en el botón **Ejecutar** situado a la derecha, con lo que el resultado aparecerá igualmente en la ventana de resultados. Esto lo haremos si conocemos el código y preferimos escribirlo en lugar de acudir al menú desplegable, pero sobre todo si queremos implementar un cálculo o técnica que no está disponible en el menú desplegable.

Por último, veremos una ventana inferior de **Mensajes**, en la que aparecerán básicamente mensajes de error que nos informarán de qué hemos hecho de manera incorrecta.

Bibliografía

Gibernans Bàguena, J.; Gil Estallo, À. J.; Rovira Escofet, C. (2009). *Estadística*. Barcelona: Material didáctico UOC.