Índice general

Tema 2

El entorno de programación y análisis estadístico R. Introducción

2.1. Introducción

2.1.1. Qué es R

Es posible que este sea tu primer acercamiento a R y con ello te cuestiones sobre la ventaja y la utilidad de R sobre otros programas de Estadística. En lo que sigue intentaremos responder a algunas de tus posibles cuestiones y te animaremos a que explores las distintas posibilidades que ofrece.

En primer lugar podemos presentar R como un *software* libre y de código abierto que se distribuye bajo los términos GNU (General Public Licence, `www.gnu.org`): *absolutely no warranty*. Se trata de un software multiplataforma disponible para los sistemas operativos más populares Linux, Windows y MacOS X, entre otros. En segundo lugar decir que R es un lenguaje de programación y un programa de análisis estadístico, en este sentido es usual referirse a él como un entorno (o ambiente) de programación y análisis estadístico. El término "entorno" pretende caracterizarlo como un sistema totalmente planificado y coherente, en lugar de una acumulación gradual de herramientas específicas, como suele ser el caso de algunos programas de estadística. Esta idea forma parte de la filosofía de creación donde se buscaba crear un entorno interactivo en el que el usuario no se vea conscientemente como un programador sino que, conforme sus necesidades sean más claras y más complejas, pueda gradualmente profundizar en los aspectos del lenguaje y la programación. Esta idea es la que a su vez ha guiado la planificación de los primeros temas de este curso de Estadística Computacional.

2.1.2. Cómo empezó R

El *proyecto* R comenzó en 1992 por un grupo de estadísticos de la Universidad de Auckland, dirigidos por Ross Ihaka y Robert Gentleman. Su motivación inicial era crear un lenguaje didáctico que pudiera ser utilizado en cursos introductorios de Estadística.

Para ello decidieron adoptar la sintaxis del lenguaje S desarrollado por Bell Laboratories en los años 80 (de ahí la similitud en la sintaxis que existe entre ambos lenguajes).

El nombre de R hace referencia a las iniciales de Ross y Robert que empezaron a denominar así al lenguaje que implementaron. En 1992 se da un primer anuncio al público del software R pero no fue hasta hasta febrero del 2000 cuando se considera al software completo y lo suficientemente estable para publicar la versión 1.0.[1] Desde entonces ha existido un grupo expertos, el *R Core Team*, que contribuye al desarrollo de R, manteniéndolo y guiando su evolución. Los miembros de este grupo se pueden consultar en `https://www.r-project.org/contributors.html`.

2.1.3. Características del software R

Como lenguaje de programación R consiste en un *lenguaje orientado a objetos*. R está basado en el lenguaje de programación S[2]. Por otro lado R es un *lenguaje funcional* lo que implica un estilo particular para resolver problemas centrado en la creación de funciones. Así un problema complejo se descompone en varias partes, las cuales se resuelven creando una función o combinación de funciones sencillas y fáciles de entender, que operan independientemente. Frente a otros lenguajes de programación, R es sencillo e intuitivo, ya que se trata de un lenguaje interpretado (similar a Matlab o Phyton) y está orientado al análisis estadístico (fórmulas, modelos, etc.).

Como *programa de análisis estadístico* R está dividido en dos partes: el sistema base de R, y todo lo demás, en forma de paquetes[3] específicos adicionales (*contributed packages*[4]) El sistema base de R contiene el paquete básico que se requiere para su ejecución y la mayoría de las funciones fundamentales. Además hay otros paquetes contenidos en el sistema base que incluyen funciones básicas para el análisis estadístico y gráfico de datos, así como otras utilidades (utils, stats, datasets, graphics, tools, parallel, etc.). A través de multitud de paquetes adicionales el sistema base de R permite extensiones que implementan técnicas estadísticas avanzadas así como herramientas sofisticadas para la visualización, la lectura y la manipulación de datos. De este modo R se presenta como una herramienta que cubre las necesidades de cualquier analista, tanto en el ámbito de la estadística profesional como en el de la investigación estadística.

El que R sea un software libre lo hace sin duda muy atractivo ya que nos podemos despreocupar por licencias. Además cuenta con la libertad que garantiza GNU. Entre otras el usuario tiene libertad para usar R con cualquier propósito, para estudiar cómo trabaja el programa y adaptarlo a sus necesidades (dado que se tiene acceso al código fuente), así como libertad para mejorar el programa y liberar tales mejoras al público en general.

---

[1] Puedes leer sobre los orígenes del lenguaje por ejemplo en `https://www.stat.auckland.ac.nz/~ihaka/downloads/Interface98.pdf`.

[2] S se considera un lenguaje de programación estadística orientado a objetos de alto nivel. Fue desarrollado por John Chambers y colaboradores en los Laboratorios Bell (AT&T) en los años 80, y diseñado específicamente para la programación de tareas estadísticas.

[3] Los paquetes de R (*R Packages*) son colecciones de funciones de R, datos y código compilado.

[4] Ver listado actual en `http://cran.es.r-project.org/`.

Otra ventaja de R es que tiene una comunidad de usuarios muy activa y motivada, que contribuye con la creación de diversas extensiones, así como respondiendo a cuestiones de otros usuarios[5], tanto en el ámbito más aplicado del análisis de datos como en el de la programación más avanzada y la investigación.

Finalmente hay que mencionar que R tiende a consumir muchos recursos de memoria. Esto hace que si se utilizan datos de gran tamaño el programa se ralentiza y, en el peor de los casos, no podría procesarlos. Esto motiva una programación cuidadosa en la que se exploten algunas facilidades de R como la vectorización (que veremos más adelante), así como, en casos más complejos, opciones relativas al particionamiento aprovechando por ejemplo el procesamiento en paralelo.

2.1.4. Descarga y documentación

El sitio CRAN, del inglés *Comprehensive R Archive Network*, en la dirección `http://cran.es.r-project.org/`, es una red de servidores en todo el mundo desde la cual podrás encontrar todo lo que necesites de R. Desde CRAN es posible descargar R en sus versiones para Windows, Mac y Linux, además de extensiones a través de paquetes específicos.

En CRAN también podemos acceder de forma gratuita a diversos manuales y documentos de ayuda en varios formatos. Pero también encontrar respuestas a preguntas habituales (FAQs[6]) y seguir los últimos desarrollos del proyecto. El denominado *The R Journal* es la revista oficial de proyecto R. Esta revista publica artículos que cubren temas de interés para usuarios y programadores (nuevos paquetes, herramientas para la programación y detalles útiles, así como aplicaciones interesantes).

2.1.5. Instalación de R

El primer paso para empezar a usar R es descargarlo de Internet e instalarlo en tu ordenador. Como comentábamos antes la descarga se puede hacer desde el sitio CRAN en la dirección `cran.r-project.org/`. En esta dirección se encuentra el proyecto R completo, lo que permite entre otras cosas ver cómo está construido y descargar las herramientas necesarias para hacerlo. Para nuestros objetivos en este curso nos bastará con descargar una de las versiones binarias desde el cuadro *Download and Install R*, eligiendo la apropiada para tu sistema operativo (Windows, Linux o Mac OS).

[5]Habitualmente encontraremos respuestas a muchas de nuestras preguntas desde la experiencia de otros usuarios, buscando por ejemplo en Google, restringiendo la búsqueda a páginas relacionadas con R en `https://rseek.org/`, y StackOverflow incluyendo `[R]` en la búsqueda.

[6]R tienes tres colecciones de FAQs: las generales para todas las plataformas (R FAQ) y complementos específicos para Mac (R Mac OS X FAQ) y Windows (R Windows FAQ).

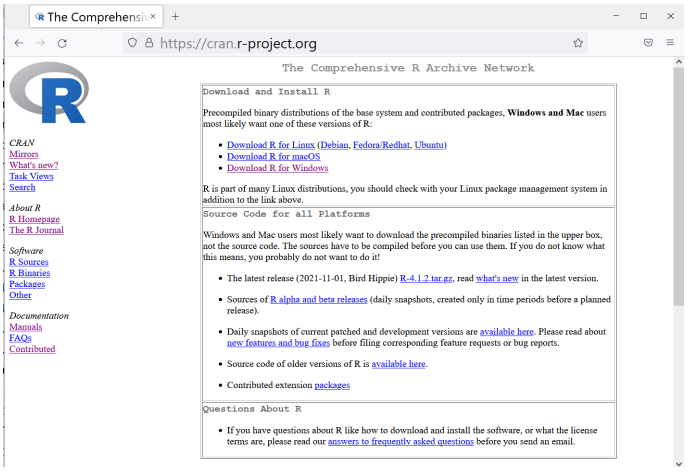

Figura 2.1: Sitio CRAN (*Comprehensive R Archive Network*).

Estas notas han sido escritas en referencia a la versión de Windows[7]. Si instalas R por primera vez descargarías el sistema base (subdirectorio *base*) desde el ejecutable `R-x.x.x-win.exe`[8]. Una vez hayas descargado el ejecutable harías doble clic sobre él y, eligiendo las opciones por defecto[9], esperar hasta que aparezca la confirmación de que se ha instalado correctamente. Se trata de un proceso sencillo que realiza cambios mínimos y copia los ficheros necesarios en un subdirectorio con el nombre de la versión instalada. En la instalación por defecto también podremos crear un acceso directo en el escritorio.

2.2. Mi primera sesión con R

Después de los preliminares anteriores estamos listos para una primera sesión de trabajo en R. Nuestro objetivo en ella es tener una primera impresión del entorno de trabajo de R y experimentar algunos conceptos iniciales.

Empezamos la sesión abriendo el programa. Si seguiste las indicaciones para la instalación anteriores, posiblemente tengas un icono de acceso directo en el escritorio (que se encuentra en el directorio de instalación `bin\Rgui.exe`) que te permitirá ejecutar R en modo interactivo[10]. Una vez ejecutado se abre la ventana principal del programa que contiene un menú y dentro la ventana de "la consola" (*R console*), desde la cual teclearemos y ejecutaremos instrucciones del lenguaje R.

[7] Para las otras versiones encontrarás ligeras diferencias. Puedes obtener más información y ayuda en la misma web, con detalles más avanzados en el documento *R Installation and Administration* disponible en pdf entre los manuales disponibles en el mismo sitio CRAN.

[8] La última versión de R a fecha de creación de estas notas era la 4.1.2 (`R-4.1.2-win.exe`) pero es posible que cuando las leas estas notas la versión haya avanzado, descárgate la última disponible.

[9] Salvo que quieras controlar algunas especificaciones como el directorio donde se instala o preferencias de configuración. Si no estás seguro/a sobre si quieres hacerlo quizá lo mejor sea que no lo hagas.

[10] Esta ejecución interactiva no es la única forma de trabajar en R. Puedes ver por ejemplo el documento de ayuda "An Introduction to R" (Apéndice B) en CRAN para más información sobre otras formas de ejecutar R en Windows, así como en otros sistemas operativos.

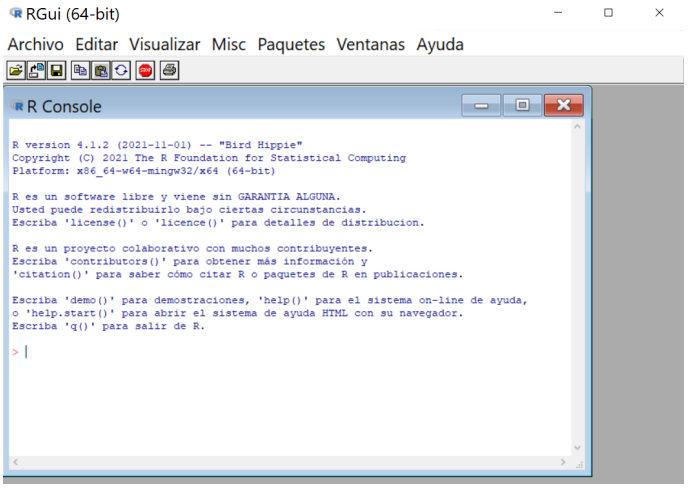

Figura 2.2: Interfaz básico de R en Windows: Rgui.

Lo primero que vemos en la ventana de la consola es una cabecera con el número y fecha de la versión de R que hemos ejecutado, así como un texto que hace referencia a los términos de la licencia de distribución y al proyecto colaborativo R. También ofrece indicaciones básicas de cómo actuar. Debajo del texto encontrarás una línea en blanco que comienza con el símbolo `>` (*prompt*) y a continuación el cursor. Este es el modo en el que R te dice que está esperando tus instrucciones.

Antes comentábamos que R es un lenguaje interpretado, esto supone que puedes escribir instrucciones y R las ejecutará directamente. Vamos a comenzar con algo sencillo, escribe una operación aritmética por ejemplo `2+2` y a continuación pulsa la tecla intro (*enter* o retorno), verás que obtienes el resultado esperado. Puedes repetir con algo como `7/2` ó `2*4`, etc. Esto nos dice dos cosas, la primera es que R se puede usar como una calculadora y la segunda es que `2`, `7`, `4`, `/` y `*` son elementos que el lenguaje R entiende. Intenta ahora algo más complicado, por ejemplo escribe las siguientes líneas (pulsado intro al finalizar cada una) y observa el resultado:

```
5^(3/4)-log(2)
# operador división entera
10%/%4
# operador módulo
5%%4
# números complejos (i representa la parte imaginaria)
(1+1i)*(3-2i)
(1+1i)*(3-2i);(1+1i)/(3-2i)
```

Verás que las líneas que comienzan con el símbolo `#` no producen ningún resultado, se trata de líneas de comentarios. En este caso te indican el significado de los operadores `%%` (módulo) y `%/%` (división entera) así como la sintaxis de los números complejos donde `i` representa la parte imaginaria. Finalmente también habrás comprobado que se pueden

escribir dos instrucciones independientes en la misma línea física utilizando el separador ;. Comprueba también que puedes recuperar líneas de instrucciones introducidas anteriormente pulsando la tecla con la flecha ascendente del teclado. Esto te permite agilizar la escritura ya que puedes reutilizarlas haciendo las modificaciones necesarias.

Hay que tener en cuenta que R no evalúa una instrucción hasta que entiende que hemos terminado de escribirla (aunque pulsemos intro). Por ejemplo si hemos abierto un paréntesis la instrucción no terminaría hasta que lo cerremos etc. Prueba a escribir una expresión incompleta, pulsa intro y observa el resultado. Por ejemplo:

```
> exp(2-3*

+ pi

+ )

[1] 0.0005962933
```

Observa que mientras la instrucción está incompleta el símbolo que aparece al comienzo de la línea es + en lugar del prompt >, el cual sólo aparece una vez que se ha completado. A veces esto ocurrirá porque hemos cometido un error escribiendo la instrucción, en ese caso siempre podremos terminar cualquier instrucción pulsando la tecla `Esc`.

Muchas de las tareas que realizamos en R se hacen a través de funciones. Una de las más útiles para empezar puede ser la función `help`. Escribe (una a una) las siguientes líneas en la consola y observa los resultados:

```
Help
help
help()
help("log")
```

Lo primero es indicar que R distingue entre mayúsculas y minúsculas de modo que `help` no es lo mismo que `Help` (esta última no corresponde a ninguna función de R). Después podemos ver que escribiendo el nombre (correcto) de una función produce resultados distintos dependiendo de si va seguido o no de paréntesis `()`. Escribiendo la función con paréntesis, el sistema como respuesta intentaría ejecutarla. En el caso de `help()` su ejecución consistiría en mostrarnos la ayuda en línea de R, y `help("log")` la ayuda[11] referente a la función `log`. Por otro lado si escribimos `help`, sin paréntesis, lo que el sistema nos devuelve es la definición de la propia función `help`. Esto se aplica a cualquier función de R. Otra función de ayuda interesante es la función `help.search` que permite solicitar ayuda escribiendo entre paréntesis algún término general (no necesariamente el nombre de una función de R) por ejemplo `help.search("read data")` nos mostraría

[11] De forma abreviada también se puede escribir `?log` para obtener el mismo resultado. La función sobre la que se solicita la ayuda puede escribirse entre comillas dobles, como en este ejemplo, o simples, `help('log')`. Además en muchos casos no es necesario usar las comillas.

todos los temas relacionados con la lectura de datos etc. Otra forma de acceder a la ayuda de R es utilizar la opción correspondiente del menú en la ventana principal del programa.

La ayuda de R para cada función proporciona ejemplos de su uso. Para aprender el funcionamiento de una función suele ser útil ejecutar dichos ejemplos en la consola. Esto lo puedes hacer copiándolos de la ayuda y pegándolos en la consola, o también usando la función `example` como se muestra a continuación para la función que calcula la media artimética (`mean`):

```
> example(mean)


mean> x <- c(0:10, 50)

mean> xm <- mean(x)

mean> c(xm, mean(x, trim = 0.10))
[1] 8.75 5.50
```

Es posible que no entiendas lo que significan las líneas mostradas en dichos ejemplos, no te preocupes por ello de momento.

Para terminar vamos a explorar otra función de R, la función `q`. Puedes escribir `help("q")` y verás que se trata de una función que permite terminar tu sesión de R. Si escribes ahora `q()` y pulsas intro habrás lo estarás solicitando, en cuyo caso el sistema te preguntará si deseas guardar el área de trabajo. Si respondes afirmativamente te pedirá el nombre del fichero (con extensión `RData`) que almacenará todos los objetos que has creado durante la sesión[12] y luego terminará. Observa que también puedes terminar la sesión cerrando la ventana principal del programa.

Hasta aquí podría llegar nuestra primera sesión con R, no obstante si quieres experimentar un poco más el funcionamiento de R puedes ejecutar (línea a línea) las sentencias que aparecen a continuación (en otro caso termina tal y como hemos indicado antes). Hemos incluido algunos comentarios para ayudarte a seguir la ejecución. Todo lo estudiaremos más adelante por lo que no te preocupes demasiado por la sintaxis de momento.

```
# 1. Generamos dos vectores de datos desde sendas distribuciones normales
#    y los guardamos en sendos objetos 'x' e 'y'
x <- rnorm(50)
y <- rnorm(50,mean=10,sd=2)
#    Imprimimos en la consola el contenido de los objetos 'x' e 'y'
x
y
# 2. Resumen descriptivo de los datos
```

[12]Sobre este tema incidiremos más adelante pero para una sesión como esta probablemente no te interese guardarlo.

```
summary(x);summary(y)
# 3. Histograma de x
hist(x)
# 4. Representamos la distribuci\'on conjunta de los datos (scatterplot)
plot(x, y)
# 5. Ajustamos una recta de regresi\'on a los datos
fit<-lm(y~x)
summary(fit)
# 6. Superponemos la recta de regresi\'on al gr\'afico anterior
abline(fit)
# 7. Creamos dos nuevos vectores con secuencias de valores
x<-1:10
y<-seq(-pi,pi,length.out=10)
# 8. Escribimos una matriz con los vectores anteriores por columnas
cbind(x,y)
# 9. Escribimos una matriz con dos filas y cinco columnas con los elementos de x
matrix(x,2,5)
# 10. Creamos una matriz con filas y columnas indexadas por x e y,
#     cuyos valores son cos(y)/(1 + x^2))
f <- outer(x, y, function(x, y) cos(y)/(1 + x^2))
f
# 11. Dos representaciones tridimensional de f como funci\'on de x e y
#     primero un diagrama de contornos
contour(x, y, f)
#     añadimos m\'as niveles
contour(x, y, f, nlevels=15, add=TRUE)
#     y ahora un mapa de colores
image(x,y,f)
# 12. Demostraci\'on de otras funciones gr\'aficas
demo(graphics)
demo(persp)
# 13. Terminamos la sesi\'on
q()
```

2.3. Entorno de trabajo

2.3.1. Scripts

En nuestra primera sesión con R hemos utilizado la ventana consola para escribir y ejecutar una a una cada línea de instrucciones. En muchos casos puede ser más conveniente guardar todo el conjunto de instrucciones en un único fichero de texto para su posterior ejecución. Nos referiremos a este tipo de fichero como un *script* de R.

Para ilustrar su uso vamos escribir parte del código de la sesión anterior en un script

y describiremos cómo ejecutarlo. Podemos crear un fichero script desde el menú de la ventana principal del programa: `Archivo >Nuevo script`. El script creado se mostrará en una nueva ventana tal y como se muestra en la Figura 2.3. Se trata de una ventana de tipo texto (como la ventana de la consola) dentro de la cual estará el cursor[13] preparado para teclear el código.

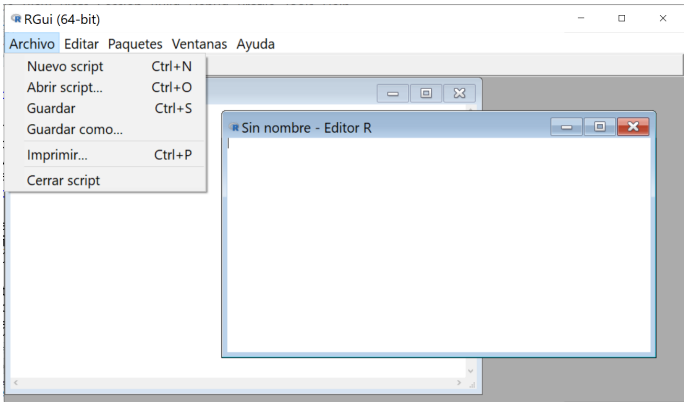

Figura 2.3: Script de R.

A continuación podemos copiar y pegar instrucciones de la sesión anterior en el script (o escribirlas de nuevo). Una vez hecho esto podemos guardar el fichero eligiendo en el menú Archivo > Guardar como... e indicando un nombre[14] y una ubicación en el disco duro del ordenador. Finalmente podemos elegir entre ejecutar el script completo o parte de él. Esto lo podemos hacer desde el menú principal, por ejemplo Archivo > Ejecutar todo (ver Figura 2.4), o bien situar el cursor en una línea concreta y elegir Archivo > Correr línea o seleccionar. También es posible ejecutar algunas de las líneas seleccionándolas previamente y eligiendo la misma opción en el menú. Incluso puedes acceder a dichas opciones pinchando sobre la ventana del script con el botón derecho del ratón. Además, una vez creado y guardado, la función `source` permite ejecutar todas las sentencias de un script.

[13] Para ello tiene que estar activa la ventana, si no lo está basta con hacer click sobre ella.

[14] Observa que la extensión de los scripts es `.R`. Estos ficheros los podrás después abrir en R o con cualquier editor ya que se trata de un fichero de texto.

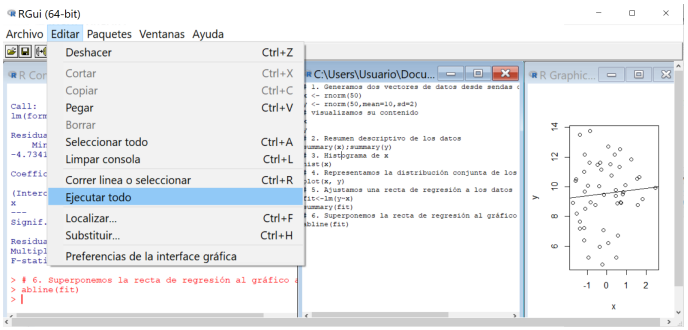

Figura 2.4: Ejecución de un script de R.

2.3.2. Área de trabajo y resultados

Además de guardar el código en un fichero script, como hemos descrito antes, en algunas ocasiones será de utilidad guardar todos los *objetos*[15] que hayamos creado o manipulado en la sesión de R (y acceder posteriormente a ellos). Esto es lo que se conoce como el *Workspace*, traducido como área de trabajo, y consiste en todos los objetos que el sistema tiene en memoria[16] en una sesión de R.

Para guardar el área de trabajo, al final o en cualquier momento de la sesión, se puede utilizar el menú de la ventana principal del programa Archivo > Guardar área de trabajo[17]. Esto ejecutaría la siguiente instrucción del lenguaje: `save.image(file = "nombre.RData")`, escribiendo el "nombre" que queramos darle al archivo.

Si no indicamos el camino (directorio) donde queremos guardar el archivo, este se guardará en el "directorio de trabajo". Para ver cuál es dicho directorio basta con escribir `getwd()`. Si queremos cambiarlo usaremos la función[18] `setwd` indicando el directorio deseado, por ejemplo `setwd("C:/aqui")` elegiría el directorio "aqui" en el disco C (si este existiera).

Es importante modificar el directorio de trabajo al empezar la sesión, especificando aquel donde queramos almacenar nuestros resultados, así como localizar en su caso los ficheros necesarios para nuestro trabajo (ficheros de datos, scripts etc.).

En una sesión de R se pueden generar distintos tipos de resultados. Por un lado están los objetos que hayamos creado y modificado, a los que nos hemos referido antes, y por otro los resultados de los análisis estadísticos que hayamos podido r realizar. Estos últimos son los que habitualmente se muestran en la ventana de la consola (de tipo texto) y los

[15] Más adelante definiremos formalmente este concepto, por el momento puede ayudarte pensar en los objetos como variables que están en la memoria.

[16] R mantiene todos los objetos creados en memoria, existiendo límites basados en la cantidad de memoria usada por todos ellos. Puedes consultar la ayuda, por ejemplo `help(Memory)`, para conocer más sobre cómo gestiona R la capacidad del *workspace* así como las limitaciones de memoria para almacenar objetos en R.

[17] Es necesario que esté activa la ventana de la consola para que puedas ver esta opción en el menú. De hecho el menú muestra distintas opciones dependiendo de la ventana que esté activa (consola, script, gráfico etc.) y en relacióna la misma.

[18] También se puede hacer desde el menú Archivo > Cambiar dir...

gráficos que se muestran en ventanas gráficas. Cuando guardamos el área de trabajo (tal y como describíamos antes) no se almacenan estos resultados como tal. La forma más simple de guardar los resultados de tipo texto en R es seleccionarlos en la consola y copiarlos en algún editor de textos conveniente. Desde dicho editor podremos con facilidad elaborar un informe de la sesión, añadiendo por ejemplo algún comentario y editando el texto. En el caso de los gráficos, es posible guardarlos en diversos formatos (jpg, bmp, pdf etc.) desde el menú principal o desde el menú que se despliega pinchando con el botón derecho del ratón en la ventana gráfica. Luego podemos también insertarlos adecuadamente en el editor de textos junto a los otros resultados. Una forma más elegante y eficiente es crear lo que se denomina un documento dinámico, esto es, un documento que contiene el código en R y el resultado de su ejecución tal y como se muestra en la sesión interactiva. Esto se puede hacer de una forma relativamente sencilla usando distintas extensiones de R como Rmarkdown, Sweave o knitr. Cuando hayamos adquirido mayor experiencia con R nos ocuparemos de esta posibilidad, de momento nos bastará con y guardar el código en scripts y, posiblemente en algún caso, guardar también el área de trabajo.

2.3.3. Paquetes de R

Al ejecutar R se cargan por defecto el sistema base.[19]. En ocasiones nos interesará extender el sistema incorporando alguno de los múltiples paquetes adicionales que hay disponibles en CRAN. Para ver una lista actualizada de los paquetes disponibles puedes visitar este sitio[20], en concreto `http://cran.r-project.org/web/packages/`, tal y como muestra la siguiente figura.

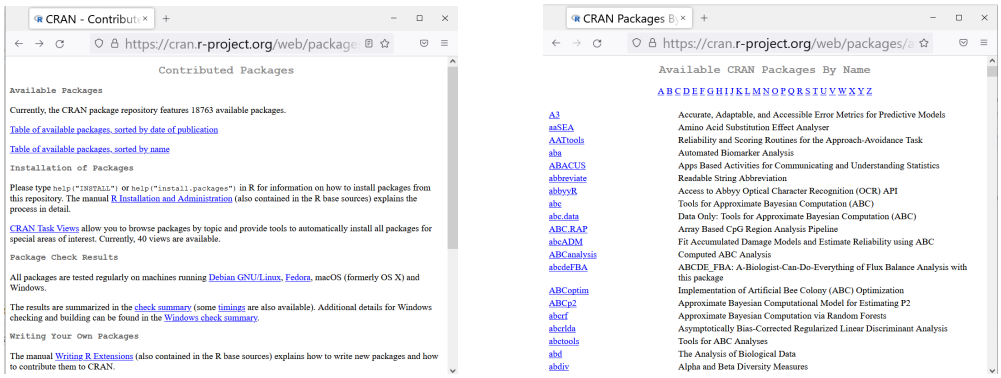

Figura 2.5: Paquetes adicionales en CRAN (fecha de acceso: 19/01/2022).

Para cargar alguno de estos paquetes en nuestra sesión de R lo primero es instalarlo. Esto lo podemos hacer fácilmente desde el menú principal Paquetes > Instalar paquete(s)... Con esto se abrirá un cuadro donde seleccionaremos uno de los servidores (CRAN

[19] Podemos escribir `search()` para ver los paquetes cargados además del sistema base.

[20] Además *CRAN task views* en `http://cran.rstudio.com/web/views/` proporciona información sobre qué paquetes son relevantes para tareas concretas.

mirrors), y después otro donde seleccionaremos el nombre del paquete que queremos instalar[21]. Para ilustrar este proceso prueba por ejemplo a instalar el paquete *maxLik* (u otro que te llame la atención). Al finalizar el proceso puedes ver un mensaje indicando si la instalación se ha realizado correctamente. En lugar de usar el menú también podríamos haber realizado la instalación escribiendo en la consola `install.packages("maxLik")`.

Los paquetes de R se instalan dentro de bibliotecas (*libraries*), las cuales corresponden a directorios en los ficheros del sistema. Por defecto R se instala con una única biblioteca[22], en el directorio `library`, que contiene los paquetes básicos y recomendados (cada uno en un subdirectorio).[23]

Una vez que el paquete está instalado podemos cargarlo escribiendo `library(maxLik)` (o bien usando el menú Paquetes > Cargar paquete...). Si obtenemos la confirmación de que se ha cargado correctamente ya podemos empezar a utilizar su contenido[24]. Si quieres saber un poco más de él puedes también solicitar ayuda escribiendo `help(maxLik)`.

Al cargar un paquete con la función `library`, durante la sesión de trabajo, este aparece en la lista de búsqueda que podemos comprobar escribiendo `search()`. Es posible que una vez usado un paquete queramos "descargarlo", para ello podemos usar la función `detach`. Por ejemplo si hemos cargado el paquete *maxLik* podemos descargarlo escribiendo `detach(package:maxLik)`. Ten en cuenta que esto no es posible hacerlo para el paquete base.

2.3.4. Otros interfaces y entornos para trabajar con R

Bajo Windows, R se ejecuta por defecto a través de la interfaz gráfica RGui (*R Graphical User Interface*) mostrada antes en la Figura 2.2. No obstante es posible usar otros interfaces más intuitivos o con capacidades adicionales como RStudio y Microsoft R Open, entre otros.

RStudio es uno de los interfaces más utilizados actualmente por muchos usuarios de R. De hecho resulta muy útil para principiantes por lo que puede ser muy adecuado para este curso introductorio. RStudio está disponible en la dirección `http://www.rstudio.com`, en sus versiones libre (como R) y comercial. En la misma web se puede encontrar documentación completa del programa así como recursos adicionales e ilustraciones de sus capacidades.

RStudio es mucho más que un interfaz o editor para R, se define como un entorno de desarrollo integrado (*Integrated Development Environment*, IDE). RStudio incluye un editor de sintaxis que permite la ejecución directa del código así como diversas herramientas para la representación gráfica, depuración, manipulación del entorno de trabajo. RStudio

[21] Este proceso solo lo haremos la primera vez que usamos un nuevo paquete.

[22] Si quieres ampliar este tema puedes ver el documento en CRAN *R Installation and Administration*.

[23] Para ver los paquetes que tenemos instalados podemos escribir `library()`.

[24] Este proceso lo tenemos que hacer cada vez que comenzamos una sesión de R y queramos usar el paquete. Aunque este esté ya instalado de sesiones anteriores, no se cargará en memoria por defecto. Para ver qué paquetes carga R por defecto (además del paquete base) puedes escribir `getOption('defaultPackages')`

ofrece herramientas para extender el programa creando paquetes que se pueden compartir por ejemplo en el sitio CRAN. RStudio simplifica la creación de informes estadísticos o documentos dinámicos con RMarkdown, Sweave o knitr, así como aplicaciones web interactivas con shiny.

Por otra parte, Microsoft R Open (`https://mran.revolutionanalytics.com/open/`) es la distribución mejorada de R de Microsoft. Al igual que R es libre y permite todas sus capacidades, con el valor añadido de incluir mejoras relativas por ejemplo a la computación en paralelo, lo que permite reducir los tiempos computacionales.

2.4. Algunas nociones básicas del lenguaje

2.4.1. Objetos y reglas de sintaxis básicas

Hemos comentado anteriormente que R es un lenguaje de programación orientado a objetos. En R todo es un objeto y el trabajo en R consiste básicamente en crear y manipular objetos. Hay que distinguir dos tipos de objetos básicos, por un lado las funciones y por otro los objetos de datos. Los **objetos de datos** son lo que habitualmente denominamos variables. Las **funciones** son objetos que realizan tareas sobre objetos de datos, habitualmente consistentes en cálculos, representaciones gráficas o análisis de datos en general. Estas funciones se obtienen como parte del sistema base de R (o desde paquetes adicionales), pero también es posible crearlas durante una sesión de trabajo.

El lenguaje R tiene una sintaxis simple e intuitiva. En la primera sesión (propuesta en la Sección 2.2) pudimos experimentar escribiendo y ejecutando algunas instrucciones del lenguaje, ahora empezaremos a fijar algunos conceptos elementales.

En lo anterior nos hemos referido a instrucciones o sentencias de R que ejecutamos durante la sesión. Con estos términos hacemos referencia a expresiones con una sintaxis correcta que el lenguaje puede procesar. Las instrucciones básicas en R son por un lado evaluación de expresiones (como por ejemplo las aritméticas[25] que escribíamos en la Sección 2.2), y por otro las denominadas **asignaciones**[26]. En general si escribimos y ejecutamos una expresión, el sistema la evalúa y muestra (imprime) el resultado en la ventana de la consola, pero no lo guarda. Por el contrario la asignación permite pasar dicho valor a un objeto de R, aunque no lo mostraría. El siguiente ejemplo muestra ambos tipos de sentencias para evaluar una misma expresión aritmética, en este caso $\exp(2-3\pi)$:

```
> exp(2-3*pi)

[1] 0.0005962933

> x<-exp(2-3*pi)
```

[25] Puedes consultar los operadores aritméticos en la ayuda de R escribiendo por ejemplo `help(Arithmetic)`. En la siguiente sección, la Tabla 2.1 resume los más importantes y la Tabla 2.2 las funciones matemáticas más habituales.

[26] Del inglés *assignments*, también traducido como redireccionamiento.

La primera sentencia evalúa y muestra el resultado, mientras que la segunda también lo evalúa pero en lugar de mostrarlo lo redirecciona a `x`. Si queremos ver el resultado en este último caso tendríamos que visualizar el contenido de `x` por ejemplo imprimiéndolo en la consola como sigue:

```
> x

[1] 0.0005962933
```

El operador de asignación es `<-` y puede usarse en ambos sentidos por ejemplo `x<-exp(2-3*pi)` sería equivalente a `exp(2-3*pi)->x`. Un operador equivalente para la asignación es el símbolo `=`, no obstante en estas notas preferimos `<-` por razones que veremos más adelante.

Hay que comentar que si la asignación se hace en un objeto que ya existe, esto es, está almacenado en el entorno de trabajo, este se borraría y se reemplazaría por el resultado de la asignación. Por otro lado, si el objeto no existía R lo crearía al ejecutar la sentencia.

Para nombrar objetos en R se pueden combinar letras, números, `.` y `_`. Hay que tener en cuenta que un nombre nunca debe comenzar por un número ni por el símbolo `_`, ni contener espacios en blanco. Tampoco es posible usar palabras reservadas[27]. Además R distingue entre mayúsculas y minúsculas.

En R disponemos de dos funciones básicas para gestionar los objetos durante una sesión: `ls` y `rm`. La primera[28] muestra los objetos que hay en el entorno de trabajo, mientras que la segunda permite eliminar[29] objetos. A continuación puedes ver algún ejemplo de su uso.

```
> x<-5;y<-2*x; z<-log(y);k<-x*y*z
> ls()

[1] "k" "x" "y" "z"

> rm(x,y)
> ls()

[1] "k" "z"

> rm(list=ls())
> ls()

character(0)
```

Para ejecutar una función de R debemos utilizar correctamente su sintaxis. Esto lo podemos ver en la ayuda disponible. Algunas funciones se pueden ejecutar sin especificar

[27] puedes consultarlas escribiendo `help(Reserved)`

[28] Otra función equivalente es `objects`.

[29] Ojo que este proceso es irreversible y R no pregunta antes de borrar.

ningún argumento, como por ejemplo `ls()`. En otros casos será necesario especificar alguno(s) argumento(s). Como ejemplo vamos a considerar la función `matrix` que permite crear una matriz de valores, y vamos a ilustrar su uso creando una matriz con una fila y tres columnas conteniendo el valor π en cada posición. Inspeccionando la ayuda de R podemos ver que la sintaxis de la función es:

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)
```

por lo que tiene cinco argumentos con nombres `data`, `nrow`, `ncol`, `byrow` y `dimnames`, y todos tienen un valor por defecto (`NA`, `1`, `1`, `FALSE` y `NULL`, respectivamente). Esto implica que podemos usar la función sin especificar sus argumentos, esto es, escribir y ejecutar `matrix()`, y que el resultado en dicho caso será el valor programado por defecto, esto es, una matriz con un solo elemento igual a `NA`[30]. Para obtener la matriz que buscamos necesitamos especificar algunos de sus argumentos, por ejemplo:

```
> matrix(data=pi,nrow=1,ncol=3)

         [,1]     [,2]     [,3]
[1,] 3.141593 3.141593 3.141593
```

El mismo resultado se podría haber obtenido escribiendo:

```
> matrix(pi,1,3)

         [,1]     [,2]     [,3]
[1,] 3.141593 3.141593 3.141593
```

Esto ilustra las dos formas posibles de especificar los argumentos de una función en R: por nombre o por posición. En la primera sentencia lo hicimos utilizando los nombres y en la segunda por posición, esto es, se hace coincidir cada uno de valores dentro de los paréntesis con los argumentos que ocupan la misma posición en la definición de la función. En principio puede resultar más cómoda la segunda opción no obstante la primera puede resultar más segura a la hora de detectar posibles errores en el código. Hemos de mencionar también que cuando se especifican los argumentos por el nombre, el orden es irrelevante, siendo por ejemplo equivalentes `matrix(data=pi,nrow=1,ncol=3)` y `matrix(nrow=1,data=pi,ncol=3)`. Un comentario adicional sobre este ejemplo es que especificar el segundo argumento (`nrow=1`) no es necesario ya que le estamos dando el valor por defecto que veíamos antes en la sintaxis. De este modo las dos versiones anteriores, omitiendo ese argumento, se podrían escribir de forma mas breve como sigue:

```
matrix(data=pi,ncol=3)
matrix(pi,,3)
```

[30] En breve veremos que `NA` (*Not Available*) es la forma en que R denota a un valor omitido (o faltante).

Observa que cuando se especifican los argumentos por posición (segunda sentencia anterior) hay que indica con las comas si nos saltamos algún argumento, en este caso el segundo argumento.

2.4.2. Cálculo aritmético y expresiones lógicas

Hemos descrito antes que R permite evaluar expresiones aritméticas cuyo resultado se puede o no asignar a un objeto. La Tabla 2.1 muestra los operadores aritméticos, y la Tabla 2.2 algunas funciones matemáticas básicas.

Operador	Descripción
`+ - * / ^`	más, menos, multiplicación, división y potencia
`%/% %%`	división entera y módulo
`> >= < <= == !=`	mayor, mayor o igual, menor, menor o igual, igual y distinto
`! & | xor(x,y)`	(lógica) negación, y, o, o exclusivo

Tabla 2.1: Operadores aritméticos y lógicos en R.

Función	Descripción
`log(x)`, `log(x,n)`, `log10(x)`	logaritmo neperiano de `x`, en base `n` y en base 10
`exp(x)`	inversa de `log(x)`
`sqrt(x)`	raíz cuadrada de `x`
`abs(x)`	valor absoluto de `x`
`cos(x), sin(x), tan(x)`	funciones trigonométricas (`x` en radianes)
`factorial(x)`	factorial de `x`
`choose(n,x)`	$n!/(x!(n-x)!)$
`gamma(x)`	Función $\Gamma(x)$
`floor(x)`	mayor entero más cercano a `x`
`ceiling(x)`	menor entero más cercano a `x`
`trunc(x)`	como `floor` para positivos y `ceiling` para negativos
`round(x,digits=n)`	redondeo con `n` decimales
`Re(x), Im(x), Mod(x)`	parte real, imaginaria y módulo del número complejo `x`
`runif(x)`	`n` valores aleatorios desde una uniforme (0,1)

Tabla 2.2: Algunas funciones matemáticas elementales en R.

Una parte esencial en la programación en cualquier lenguaje es la comprobación de condiciones (por ejemplo, ¿es una variable mayor que otra?, ¿son dos variables iguales?, etc.). Estas habitualmente se formulan usando las palabras "y", "o" y "no"; y se evalúan como "si" (`TRUE`), "no" (`FALSE`), o también "quizá" (cuando la respuesta no está disponible[31]). En R es posible también escribir los valores `TRUE` y `FALSE` en modo abreviado como `T` y `F`,

[31] `NA` (*not available*), como definiremos después.

respectivamente. Si bien no es muy recomendable ya que dichos caracteres pueden haber sido utilizados en el código para nombrar algún objeto. Los operadores para escribir expresiones lógicas en R se motraron antes en la Tabla 2.1 y también se pueden consultar en la ayuda (`help(Logical)`). Así por ejemplo, `==` representa la igualdad y `!=` la no igualdad. Además para dos expresiones `c1` y `c2`, `c1 & c2` representa la intersección (se cumplen `c1` y `c2`); `c1 | c2` representa la unión (se cumple `c1` o `c2`); y `!c1` representa la negación (no se cumple `c1`). Observa a continuación algunos ejemplos de uso:

```
> 10>2

[1] TRUE

> 10>2 | 3<=1

[1] TRUE

> TRUE!=FALSE

[1] TRUE

> TRUE==T

[1] TRUE

> floor(1.5)==1

[1] TRUE

> floor(1.5)==ceiling(1.5)

[1] FALSE

> floor(1.5)!=abs(ceiling(-1.5))

[1] FALSE

> x<-sqrt(3)
> x^2 == 3

[1] FALSE
```

Observa que mientras que las primeras sentencias son muy elementales y el resultado obvio, el de las dos últimas sentencias resulta algo alarmante. De hecho nos revela algo importante a tener en cuenta en el cálculo aritmético en R. Mientras que la aritmética entre enteros será exacta, no ocurre igual para el resto de números reales con los que

perdemos precisión debido a errores de redondeo (en la siguiente sección comentamos este tema con un poco más de detalle). En el ejemplo el cálculo de la raíz cuadrada de 3 devuelve un resultado que no es exacto y por tanto al elevarlo al cuadrado no obtenemos de nuevo 3 como esperábamos. Esto nos dice que hemos de programar con cuidado y sobre todo cuando queremos comprobar si dos números (resultado de alguna operación aritmética) son iguales. Para R el operador == que usamos antes significa "exactamente igual" y lo que esto implica depende de la precisión. Podemos preguntarnos ahora cuánto difieren realmente `x^2` ($\sqrt{(3)^2}$) y `3`, observa el resultado que nos da R en dicho caso:

```
> x^2 - 3

[1] -4.440892e-16
```

La diferencia claramente no es un número muy grande, pero no es cero. Observa otro ejemplo más:

```
> 0.3-0.2==0.1

[1] FALSE

> # y sin embargo:
> 0.3-0.2

[1] 0.1
```

La cuestión entonces es cómo podemos evaluar si dos números son iguales o no teniendo en cuenta esta situación. Una respuesta sencilla es no utilizar el operador == sino la función `all.equal`. Esta función permite evaluar la igualdad salvo diferencia insignificantes. Observa el resultado con los ejemplos que hemos presentado:

```
> all.equal(0.3-0.2,0.1)

[1] TRUE

> all.equal(x^2,3)

[1] TRUE
```

2.4.3. Precisión de la máquina, errores de redondeo y valores especiales

La representación de los números en un ordenador no es exacta. Dado que los números se almacenan en formato binario, incluso un número tan simple como 0.1 no se representa

de forma exacta sino que consiste en una secuencia indefinida de valores 0 y 1:

$$0.1 = (0.0\ 0011\ 0011\ 0011\ 0011\ 0011\ldots)_2$$

La precisión de los cálculos en el ordenador se caracteriza por lo que se denomina el "épsilon de la máquina" definido como el menor número ε para el cual se satisface $1 < 1+\varepsilon$. En ordenadores con aritmética de coma flotante con doble precisión $\varepsilon = 2.2 \times 10^{-16}$, lo que implica que los resultados son (como mucho) precisos hasta el decimal 15. En R este número está almacenado en `.Machine$double.eps` (puedes escribirlo en la consola para ver su valor, o en general `.Machine` para ver además otros valores relacionados).

Teniendo esto en cuenta podemos comprender el resultado que se observa a continuación:

```
> 1+10^{-15}==1

[1] FALSE

> 1+10^{-17}==1

[1] TRUE
```

Otra restricción que tenemos que tener en cuenta es que la máquina tiene limitaciones para producir y usar números muy grandes (*overflow*). Este límite puede consultarse en R escribiendo `.Machine$double.xmax`. Cualquier operación cuyo resultado esté por encima de ese valor dará lugar a *overflow*, y R lo identifica como infinito usando la palabra especial `Inf`. De forma similar existe lo que se denomina *underflow* para valores muy pequeños, y en R corresponde al valor `.Machine$double.xmin`. En este caso, si realizamos un cálculo cuyo resultado (en valor absoluto) está por debajo de dicho valor R lo identifica como 0. Las restricciones que acabamos de describir suponen un problema bien conocido en el cálculo en ordenador que es la "pérdida de precisión". Esto puede llevar a (posiblemente) sorprendentes resultados. Por ejemplo, mientras que "en papel" $1 + 10^{-17} - 1 = 10^{-17}$, si escribes estas sentencia en la consola verás que el resultado es 0. Y sin embargo, $1 - 1 + 10^{-17} = 10^{-17}$ tanto en "en papel" como en R (compruébalo).

Además del valor especial `Inf` que introducíamos antes, algunos cálculos en R pueden dar lugar a otro valor especial. Nos referimos ahora al valor `NaN` (del inglés *Not a Number*). Esto ocurrirá por ejemplo en cálculos del tipo $0/0$, ∞/∞, etc. Tanto `NaN` como `Inf` se pueden usar en expresiones aritméticas. Aunque cualquier operación con `NaN` dará lugar a `NaN`. Observa por ejemplo:

```
> Inf-Inf

[1] NaN

> 2*Inf
```

```
[1] Inf

> 2*NaN

[1] NaN

> 0/0

[1] NaN
```

Estos valores especiales también se pueden utilizar en expresiones lógicas, por ejemplo:

```
> 2*Inf==Inf

[1] TRUE

> 0/0==NaN

[1] NA
```

Observa que mientras que en la última sentencia esperaríamos como resultado el valor `TRUE`, R devuelve el valor `NA` (del inglés *Not Available*). Se trata de otro valor especial a tener en cuenta ya que R lo utiliza para identificar valores perdidos. En este ejemplo el valor perdido es el resultado de evaluar una condición lógica que involucra `NaN`.