

**Herramientas integradas en R****RStudio****Tareas habituales**

# 1. Herramientas de trabajo

En este primer capítulo conoceremos el software necesario para trabajar con R, así como aplicaciones adicionales que pueden facilitar nuestra tarea de manera considerable, como es el caso de RStudio.

## 1.1 R

La denominación *R* se utiliza tanto para referirse al lenguaje R como al motor encargado de ejecutar los programas escritos en dicho lenguaje. Podemos interactuar con dicho motor mediante una consola de comandos, así como a través de una interfaz de usuario básica aportada por el propio R.

R es una herramienta de trabajo para el tratamiento y análisis de datos. Frente a otras herramientas similares, R nos ofrece:

- R es Open Source (multiplataforma, libre, abierto, etc.)
- Gran número de paquetes disponibles
- Extensa comunidad de usuarios
- Ciclo completo de trabajo: Implementación de algoritmos, preparación de datos, análisis de resultados y generación de documentación

### 1.1.1 Instalación de R

La versión binaria (ejecutables) de R está disponible para Windows<sup>1</sup>, OS X<sup>2</sup> y múltiples distribuciones de GNU/Linux<sup>3</sup>. También es posible obtener el código fuente y compilarlo específicamente para la plataforma en la que vayamos a trabajar. Tanto binarios como fuentes se pueden descargar desde <http://www.r-project.org>.

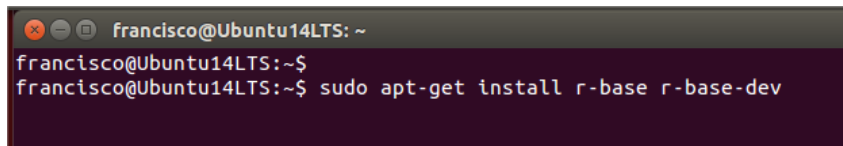
Si nuestro sistema es Windows u OS X, una vez descargado el paquete de software no tenemos más que abrirlo y seguir las instrucciones para completar el proceso de instalación.

<sup>1</sup>La descarga directa del instalador para Windows se encuentra en CRAN: <http://cran.r-project.org/bin/windows/base/>.

<sup>2</sup>La descarga directa para OS X 10.6 y posterior se encuentra en CRAN: <http://cran.r-project.org/bin/macosx/>.

<sup>3</sup>La descarga directa para Debian, RedHat, Suse y Ubuntu se encuentra en CRAN: <http://cran.r-project.org/bin/linux/>.

En el caso de Linux, también podemos efectuar la instalación de R desde el repositorio de software recurriendo a la herramienta correspondiente. Si usamos Ubuntu, o algún otro derivado de Debian, usaremos `apt-get` tal y como se aprecia en la Figura 1.1.

A terminal window with a dark background and light text. The prompt is 'francisco@Ubuntu14LTS: ~'. The user has entered the command 'sudo apt-get install r-base r-base-dev' and the prompt is still there, indicating the command has been executed or is being processed.

```
francisco@Ubuntu14LTS: ~  
francisco@Ubuntu14LTS:~$  
francisco@Ubuntu14LTS:~$ sudo apt-get install r-base r-base-dev
```

Figura 1.1: INSTALACIÓN DE R EN UBUNTU



Además del paquete adecuado para nuestro sistema operativo, también debemos tener en cuenta si necesitamos la versión de 32 o de 64 bits de R. Para esta última es imprescindible que el sistema sea también de 64 bits. La ventaja es que nos permitirá trabajar con bases de datos mucho mayores, siempre que el equipo donde usemos R tenga memoria suficiente. Para OS X únicamente está disponible la versión de 64 bits.

## 1.2 Herramientas integradas en R

Completada la instalación de R, en la ruta `/usr/bin` (Linux) o `\Archivos de programa\R\versión\bin` (Windows) encontraremos varios ejecutables:

- **R<sup>4</sup>** Es la aplicación principal de R. Con ella accederemos a la consola y ejecutaremos programas escritos en R.
- **Rscript** Motor de *scripting* de R.
- **Rgui.exe** Interfaz de usuario específica para Windows. El acceso directo que se agrega al menú **Inicio** de Windows durante la instalación apunta a este ejecutable.
- **R.app** Interfaz de usuario específica para OS X. El acceso directo a esta aplicación lo encontraremos habitualmente en la carpeta **Aplicaciones** del sistema.
- **Rterm.exe/Rcmd.exe** Ejecutables obsoletos específicos de Windows, mantenidos por compatibilidad.

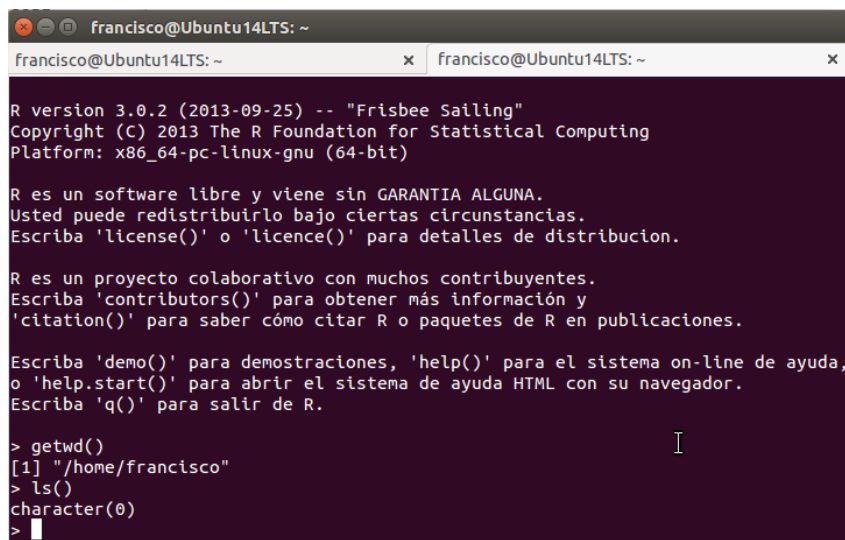
En los apartados de esta sección se introduce someramente cada una de las tres mecánicas de uso de R: como una consola de comandos, mediante una interfaz de usuario básica y como motor de ejecución de *scripts*.

### 1.2.1 La consola de R

Es la herramienta básica para operar con R de manera interactiva. Es similar a una línea de comandos del sistema operativo, pero el intérprete procesa sentencias en R en lugar de en Bash o PowerShell. La Figura 1.2 muestra la consola de R en Linux y la Figura 1.3 la misma herramienta en Windows. A pesar de la diferente apariencia, el funcionamiento de la consola es fundamentalmente idéntico en todos los sistemas operativos.

Para ejecutar cualquier comando R, no tenemos más que introducirlo y pulsar **Intro**. Un comando puede extenderse por varias líneas. El indicador (*prompt*) mostrado

<sup>4</sup>En Linux el ejecutable no tiene extensión, en Windows la extensión será `.exe`.



```

francisco@Ubuntu14LTS: ~
R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R es un software libre y viene sin GARANTIA ALGUNA.
Usted puede redistribuirlo bajo ciertas circunstancias.
Escriba 'license()' o 'licence()' para detalles de distribucion.

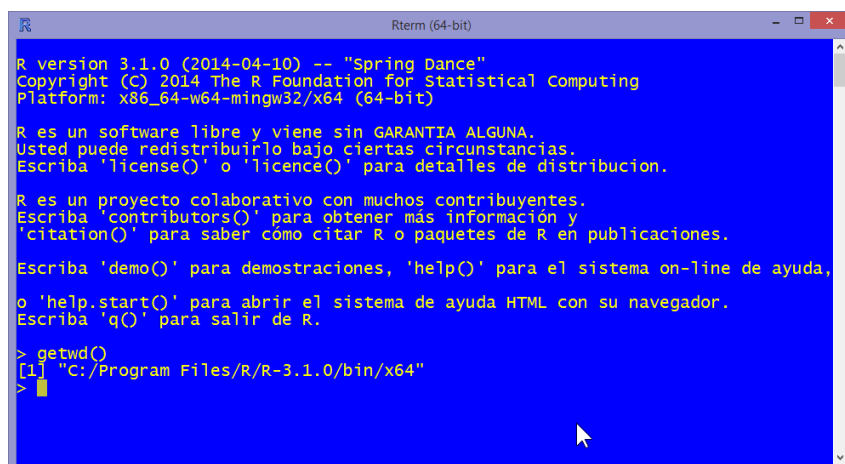
R es un proyecto colaborativo con muchos contribuyentes.
Escriba 'contributors()' para obtener más información y
'citation()' para saber cómo citar R o paquetes de R en publicaciones.

Escriba 'demo()' para demostraciones, 'help()' para el sistema on-line de ayuda,
o 'help.start()' para abrir el sistema de ayuda HTML con su navegador.
Escriba 'q()' para salir de R.

> getwd()
[1] "/home/francisco"
> ls()
character(0)
>

```

Figura 1.2: CONSOLA DE R EN LINUX



```

Rterm (64-bit)
R version 3.1.0 (2014-04-10) -- "Spring Dance"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R es un software libre y viene sin GARANTIA ALGUNA.
Usted puede redistribuirlo bajo ciertas circunstancias.
Escriba 'license()' o 'licence()' para detalles de distribucion.

R es un proyecto colaborativo con muchos contribuyentes.
Escriba 'contributors()' para obtener más información y
'citation()' para saber cómo citar R o paquetes de R en publicaciones.


Escriba 'demo()' para demostraciones, 'help()' para el sistema on-line de ayuda,
o 'help.start()' para abrir el sistema de ayuda HTML con su navegador.
Escriba 'q()' para salir de R.

> getwd()
[1] "C:/Program Files/R/R-3.1.0/bin/x64"
>

```

Figura 1.3: CONSOLA DE R EN WINDOWS

en la consola cambiará de `>` a `+` siempre que se detecte que aún faltan parámetros por facilitar para completar el comando.

-  Puedes cerrar la consola y salir de R en cualquier momento usando el comando `quit()`. Antes de salir R nos preguntará si deseamos guardar nuestro espacio de trabajo<sup>5</sup>, de forma que al iniciar de nuevo la consola podamos recuperar el estado en la que la dejamos.

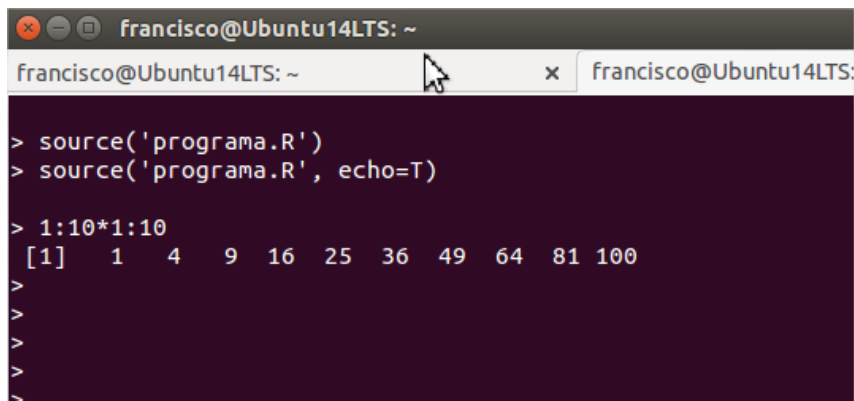
### Ejecución de módulos R

Asumiendo que tenemos código R almacenado en un módulo, este puede crearse con cualquier editor de texto y, por norma, tendrá extensión `.R`, encontrándonos en la consola de R podemos ejecutarlo mediante la función `source()`, tal y como se aprecia en la Figura 1.4.

<sup>5</sup>Nos ocuparemos más adelante del almacenamiento y recuperación de nuestro espacio de trabajo.

**Sintaxis 1.1** `source('modulo.R'[,echo=T|F, print.eval=T|F])`

Lee el contenido de un módulo de código R y lo ejecuta. El parámetro `echo` determina si los comandos ejecutados serán enviados a la salida o no. El parámetro `print.eval` determina si el resultado de la ejecución se envía a la salida o no.



```
francisco@Ubuntu14LTS: ~
francisco@Ubuntu14LTS: ~
> source('programa.R')
> source('programa.R', echo=T)

> 1:10*1:10
[1] 1 4 9 16 25 36 49 64 81 100
>
>
>
>
```

Figura 1.4: EJECUCIÓN DEL CONTENIDO DE UN MÓDULO R

Si únicamente nos interesa ejecutar el contenido del módulo y obtener el correspondiente resultado, sin interactuar con la consola de R, podemos invocar el ejecutable con las opciones CMD BATCH, facilitando dos parámetros:

- El nombre del archivo correspondiente al módulo R, incluyendo la ruta (si no está almacenado en la ruta actual) y la extensión.
- El nombre de un archivo en el que se escribirá el resultado obtenido.

La Figura 1.5 es un ejemplo de cómo utilizar R de esta forma. Además de los ya citados, R acepta muchos más parámetros que configuran el modo en que se procesará el archivo.



```
francisco@Ubuntu14LTS: ~
francisco@Ubuntu14LTS: ~
francisco@Ubuntu14LTS:~$ cat programa.R
1:10*1:10
francisco@Ubuntu14LTS:~$ R CMD BATCH programa.R salida.txt
francisco@Ubuntu14LTS:~$ tail salida.txt
Escriba 'demo()' para demostraciones, 'help()' para el sistema on-line de ayuda,
o 'help.start()' para abrir el sistema de ayuda HTML con su navegador.
Escriba 'q()' para salir de R.

> 1:10*1:10
[1] 1 4 9 16 25 36 49 64 81 100
>
> proc.time()
  user system elapsed
0.149  0.027  0.202
francisco@Ubuntu14LTS:~$
```

Figura 1.5: EJECUCIÓN DEL CONTENIDO DE UN MÓDULO R Y ALMACENAMIENTO DEL RESULTADO

### 1.2.2 Una interfaz gráfica básica

A pesar que desde la consola de R podemos realizar cualquier tarea de forma interactiva, siempre que conozcamos los comandos adecuados y sus parámetros,

algunas operaciones como la localización y almacenamiento de archivos resultan más cómodas cuando se cuenta con una GUI (*Graphics User Interface*), aunque sea muy básica. Una alternativa, también relativamente sencilla, consiste en integrar R con editores como Emacs o Vim.

La versión de R para Windows incorpora una GUI específica (véase la Figura 1.6) de tipo MDI (*Multiple Document Interface*). Al iniciar esta aplicación (`Rgui.exe`) nos encontramos con la consola de R, pero como ventana hija de una ventana marco en la que encontramos múltiples opciones, entre ellas las necesarias para crear módulos de código R, instalar paquetes<sup>6</sup>, acceder a la documentación integrada, etc.

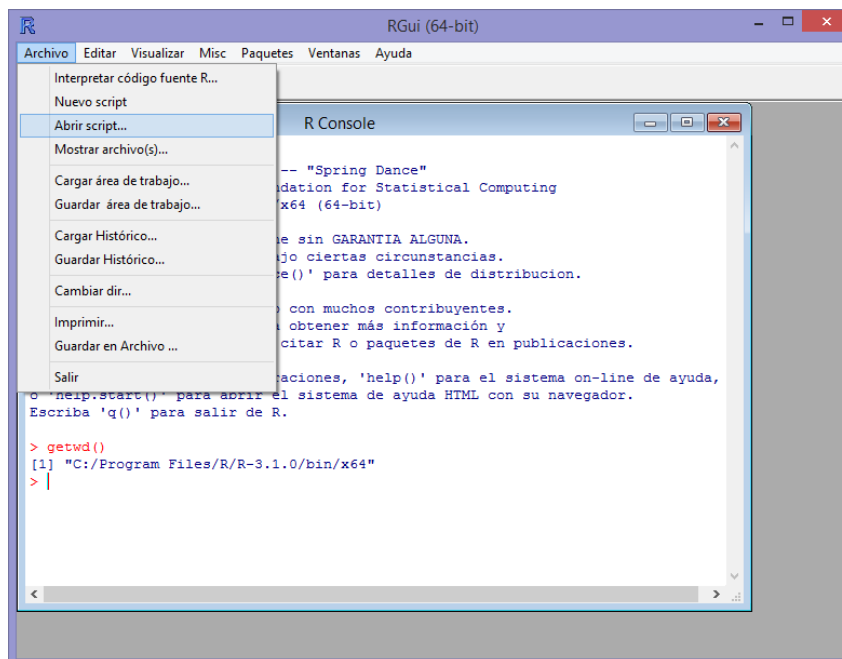


Figura 1.6: LA INTERFAZ DE USUARIO DE R EN WINDOWS

Aunque no incorpora todas las posibilidades integradas en la GUI para Windows, la versión de R para los demás sistemas también cuenta con una interfaz de usuario básica. Esta está desarrollada en Tcl/Tk y se abre añadiendo la opción `-gui=Tk` al iniciar R. Como puede apreciarse en la Figura 1.7, esta interfaz es básicamente la consola de R con un menú de opciones muy básico, con apenas media docena de opciones y un mecanismo de acceso a la documentación integrada.

Al igual que en Windows, la instalación de R para OS X también incluye una interfaz de usuario específica: `R.app`. Su funcionalidad se encuentra a medio camino entre lo que nos ofrece la GUI de Linux y la de Windows. En la barra de herramientas (véase la Figura 1.8) se ofrecen opciones para crear módulos de código R, ejecutarlos, acceder a la documentación integrada, etc.

### 1.2.3 R como lenguaje de script

Además de usarse de manera interactiva, con la consola ya mencionada, o para escribir programas completos, que pueden ejecutarse según los procedimientos antes

<sup>6</sup>La funcionalidad inicial de R se extiende añadiendo complementos, denominados genéricamente *paquetes*, que contienen código R, bases de datos, documentación, ejemplos, etc. El repositorio de paquetes oficial de R se denomina CRAN (*The Comprehensive R Archive Network*).



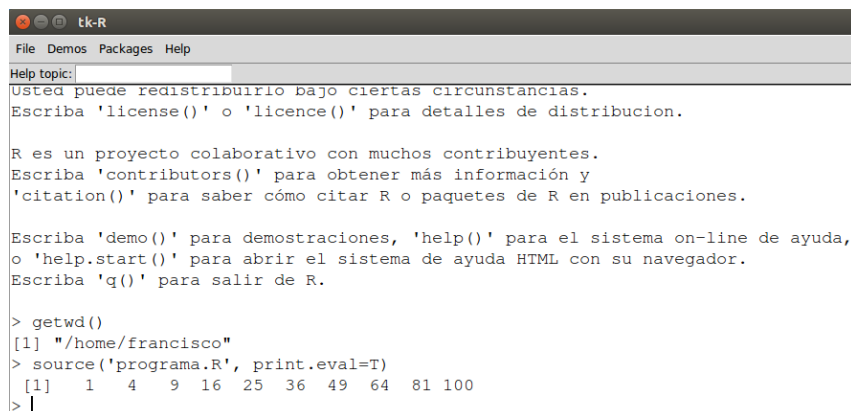


Figura 1.7: INTERFAZ DE USUARIO TK DE R EN LINUX

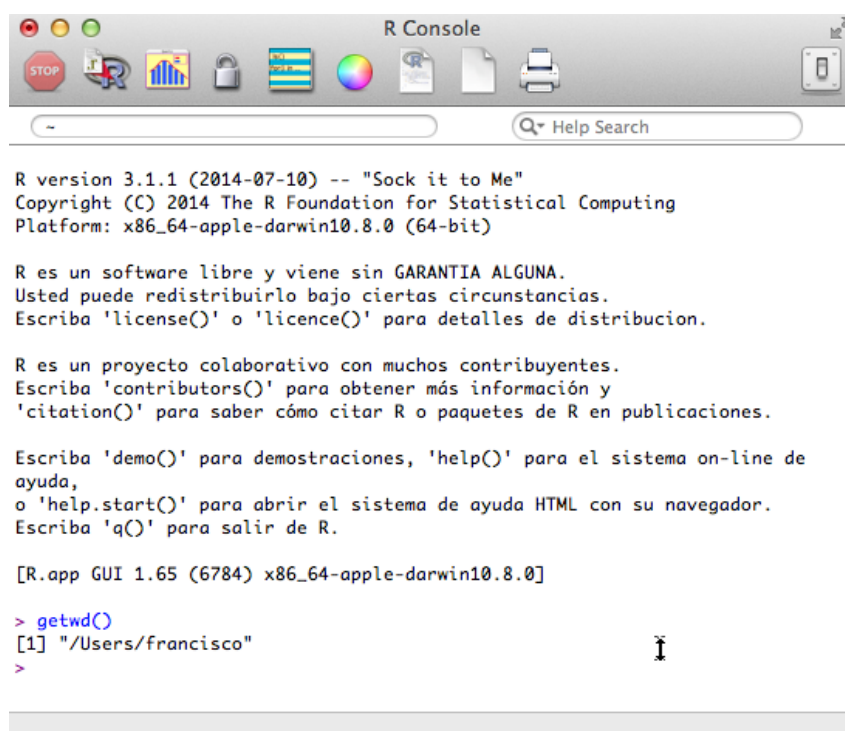


Figura 1.8: INTERFAZ DE USUARIO DE R EN OS X

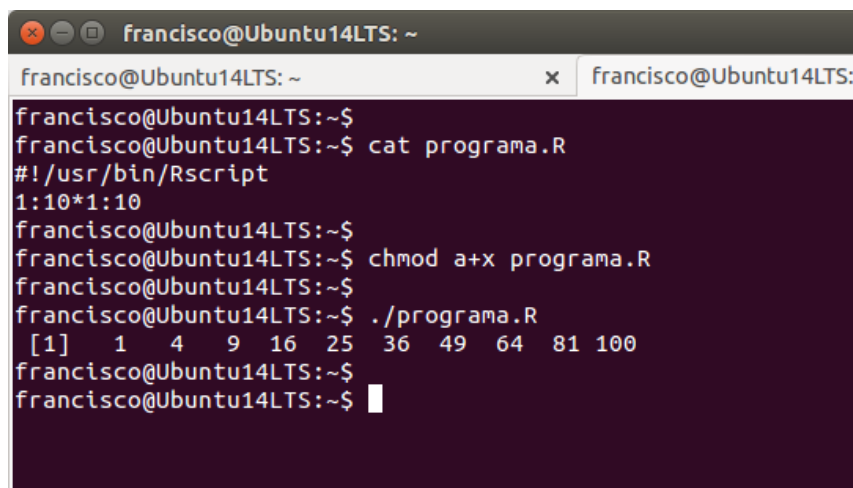
mencionados, R también puede utilizarse como un potente lenguaje de *script*. Esta es la finalidad de **RScript**, un intérprete de R que puede ser invocado facilitándole directamente el nombre de un archivo de texto conteniendo algunas sentencias R, sin más, a fin de ejecutarlas y obtener el resultado que produzcan. Por defecto ese resultado se obtendría en la salida estándar, es decir, la propia consola desde la que se ha invocado al guión. Un ejemplo podría ser el siguiente:

### Ejercicio 1.1 Ejecución de un guión en R

**Rscript** programa.r

En sistemas operativos como Linux la ejecución del guión puede automatizarse, al igual que se hace con los guiones de Bash o Perl. Para ello es necesario incluir en la

primera línea del archivo que contiene el código R la secuencia `#!/usr/bin/Rscript`<sup>7</sup> y dando permisos de ejecución del guión. En la Figura 1.9 puede verse un ejemplo de este uso. El guión en R, usando las funciones que conoceremos en los capítulos siguientes, podría tomar cualquier contenido que se facilite como entrada y procesarlo, extrayendo información estadística, generando gráficos, etc.




```
francisco@Ubuntu14LTS: ~
francisco@Ubuntu14LTS:~$
francisco@Ubuntu14LTS:~$ cat programa.R
#!/usr/bin/Rscript
1:10*1:10
francisco@Ubuntu14LTS:~$
francisco@Ubuntu14LTS:~$ chmod a+x programa.R
francisco@Ubuntu14LTS:~$
francisco@Ubuntu14LTS:~$ ./programa.R
[1] 1 4 9 16 25 36 49 64 81 100
francisco@Ubuntu14LTS:~$
francisco@Ubuntu14LTS:~$
```

Figura 1.9: EJECUCIÓN DE UN MÓDULO R COMO SI FUESE UN *script* CUALQUIERA

## 1.3 RStudio

A pesar de que desde la consola de R y un editor de texto cualquiera es posible realizar cualquier tarea: escribir guiones y programa, crear nuevos paquetes, generar gráficos y documentación, etc., la mayor parte de esas tareas pueden simplificarse de manera considerable contando con un entorno de trabajo (IDE) adecuado.

RStudio es el IDE por excelencia para R. También se trata de un proyecto de código abierto, aunque puede adquirirse con una licencia comercial que incluye soporte, y está disponible para múltiples sistemas operativos.

 Este no es un manual sobre RStudio, sino sobre R. Los procedimientos descritos en capítulos sucesivos pueden completarse desde la consola de R y, salvo contadas excepciones, se prescindirá de las opciones ofrecidas por la interfaz de RStudio en favor del uso de la línea de comandos.

### 1.3.1 Instalación de RStudio

En <http://www.rstudio.com/products/rstudio/download> encontraremos dos ediciones distintas de RStudio:

- **Desktop** Es la versión de RStudio adecuada para su ejecución local, en un equipo de escritorio o portátil, por parte de un único usuario. Cuenta con una GUI con el aspecto nativo del sistema operativo en que se instala.
- **Server** Esta versión de RStudio está diseñada para su instalación en un servidor, a fin de dar servicio a uno o más usuarios que accederían a la GUI de forma remota, desde su navegador web habitual.

<sup>7</sup>En caso necesario cambiar la ruta según dónde se haya instala **Rscript** en nuestro sistema.

De la versión para escritorio podemos tanto obtener versiones binarias de RStudio como el código fuente, en caso de que optemos por compilar nuestra propia versión de este software. Se ofrecen instaladores para Windows, OS X, Debian/Ubuntu y Fedora/Open Suse. No tenemos más que descargar la versión adecuada a nuestro sistema y ejecutar el paquete de instalación.

RStudio Server puede ser instalado directamente en Debian, Ubuntu, RedHat y CentOS. En la propia página de descarga se facilitan las instrucciones de instalación paso a paso. Para el resto de sistemas es necesario obtener el código fuente, configurarlo y compilarlo.

### 1.3.2 Introducción al IDE

Si hemos instalado la versión de escritorio de RStudio, para iniciarlo no tenemos más que usar el acceso directo que se habrá añadido o directamente ejecutar RStudio desde la consola. Se abrirá el IDE y podremos comenzar a trabajar con él. En caso de que tengamos RStudio Server instalado en un servidor, para acceder a él abriremos nuestro navegador por defecto e introduciremos el nombre o IP del servidor indicando que ha de usarse el puerto 8787. Por ejemplo: `http://localhost:8787`.

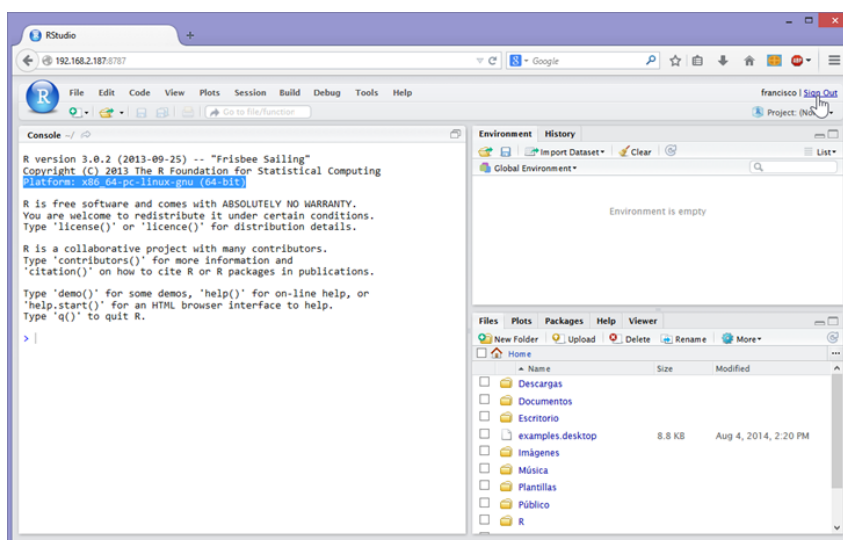


Figura 1.10: ACCESO A RSTUDIO SERVER DESDE UN NAVEGADOR EN WINDOWS

Al iniciar RStudio nos encontraremos habitualmente con tres paneles abiertos (véase la Figura 1.10):

- **Console** Ocupará la mitad izquierda de la ventana de RStudio. Es la consola de R que ya conocemos y, por tanto, la herramienta a usar para todo trabajo interactivo.
- **Environment/History** En la parte superior derecha tenemos un panel con dos páginas. En la página **Environment** irá apareciendo información sobre los objetos activos en la sesión actual: variables que se han definido, bases de datos cargadas, funciones definidas, etc. La página History es un historial de los comandos ejecutados, con opciones para recuperarlos en la consola o en un módulo de código R.
- **Files/Plots/Packages/Help/Viewer** Este panel, situado en la parte inferior derecha, ofrece páginas con opciones para acceder al sistema de archivos,



visualizar gráficas, operar con paquetes R, acceder a la ayuda y obtener una vista previa de documentos.

Mediante la opción **File>New File>R Script** podemos crear tantos módulos con código R como necesitemos. Todos ellos aparecerán en un nuevo panel en la mitad izquierda, quedando la consola en la parte inferior de dicha area (véase la Figura 1.11).

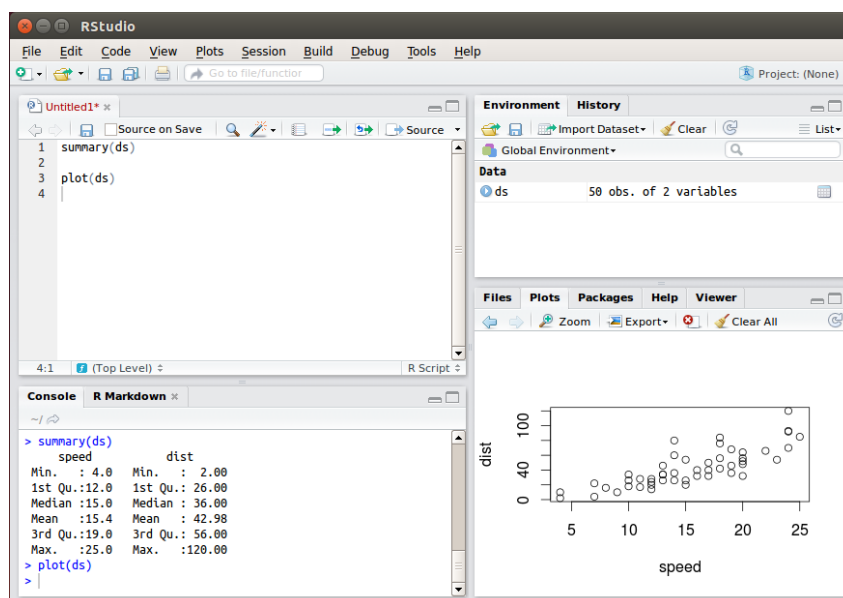



Figura 1.11: INTERFAZ DE USUARIO DE RSTUDIO EN LINUX

La edición de guiones R en RStudio cuenta con múltiples asistencias a la escritura de código: coloreado de código, autocompletado, ayuda sobre parámetros de funciones y miembros de objetos, etc.

Con las opciones ofrecidas en el menú de RStudio, y las barras de herramientas de los distintos paneles, podemos ejecutar módulos completos de código, instalar paquetes R, guardar las gráficas en distintos formatos, acceder a la documentación integrada, configurar el repositorio en el que residirá el código, crear nuevos paquetes, etc.

-  Mientras editamos código R en un módulo, podemos seleccionar una o más sentencias y pulsar **Control+Intro** para enviarlas a la consola y ejecutarlas. Esta opción nos resultará útil para ir probando porciones de código mientras escribimos un guión R.

## 1.4 Tareas habituales

En esta última sección del capítulo se explica cómo completar algunas tareas que necesitaremos llevar a cabo de manera habitual. Los procedimientos descritos tanto en este como en los capítulos siguientes son, salvo indicación en contrario, independientes de la herramienta que estemos utilizando. Se trata de comandos que introduciremos en la consola de R para su ejecución inmediata. Si utilizamos RStudio

podemos usar el panel **Console** o, como alternativa, introducir las órdenes en un módulo y después ejecutarlo usando el botón **Source**<sup>8</sup>.

### 1.4.1 Acceso a la ayuda

Al comenzar a trabajar con R necesitaremos información sobre cada instrucción, función y paquete. Toda la documentación se encuentra integrada, no tenemos más que usar la función `help()` para acceder a ella.

#### Sintaxis 1.2 `help(tema[, package = paquete])`

Facilita ayuda sobre comandos, operadores, funciones y paquetes de R. Si se incluye el parámetro `package` se buscará en el paquete indicado.

Dependiendo de la herramienta que estemos utilizando, la ayuda aparecerá directamente en la consola de R, se abrirá en nuestro navegador web por defecto o será mostrada en el panel correspondiente de RStudio. Independientemente de ello, el contenido en sí será exactamente el mismo.

#### Ejercicio 1.2 Acceso a la ayuda (No se muestra el resultado)

```
> help('source')
```



Podemos recurrir al comando `help('help')` para conocer todas las opciones de acceso a la ayuda.

### Vignettes

Muchos paquetes R incluyen, además de la ayuda estándar accesible mediante la anterior función, documentación extendida en forma de artículos y manuales de uso, generalmente en formato PDF y HTML. Este material es accesible desde la consola de R mediante la función `vignette()`:

#### Sintaxis 1.3 `vignette([tema[, package = paquete]])`

Abre la documentación adicional sobre el tema indicado, generándola si fuese preciso. Si se incluye el parámetro `package` se buscará en el paquete indicado.

Al ejecutarse sin parámetros facilita una lista de todos los temas para los que hay disponibles *vignettes*.

#### Ejercicio 1.3 Abrir el PDF asociado al sistema de gráficos grid

```
> vignette('grid') # Se abrirá un PDF sobre 'grid Graphics'
```

### Demos

El tercer recurso que nos ofrece R para aprender a usar sus posibilidades son las demostraciones, ejemplos prácticos en los que se muestran los resultados de utilizar ciertas funciones. Muchos paquetes incorporan elaboradas demostraciones de su funcionalidad, especialmente los correspondientes a gráficos.

<sup>8</sup>Este botón guardará el módulo actual y a continuación introducirá en la consola de R un comando `source('modulo.R')` para ejecutarlo.

**Sintáxis 1.4** `demo([tema[, package = paquete]])`

Pone en marcha la demostración asociada al tema indicado. Si se incluye el parámetro `package` se buscará en el paquete indicado.

Al ejecutarse sin parámetros facilita una lista de todas las demos que hay disponibles.

**Ejercicio 1.4** Ejecutar la demo asociada a la función `image`

```
> demo('image') # Se ejecutará la demo interactiva
```

### 1.4.2 Establecer la ruta de trabajo

Siempre que vayamos a trabajar con módulos de código R, generar gráficas, cargar datos de archivos externos, etc., hemos de tener en cuenta cuál es la ruta de trabajo actual y, si fuese preciso, cambiarla a la que sea adecuada. Esto es especialmente importante si pretendemos utilizar rutas relativas en un guión R.

La ruta actual se obtiene mediante la función `getwd()`. Esta devuelve como resultado una cadena de caracteres.

**Sintáxis 1.5** `getwd()`

Devuelve la ruta de trabajo actual.

Para cambiar la ruta de trabajo usaremos la función `setwd()`. La ruta facilitada puede ser relativa o absoluta.

**Sintáxis 1.6** `setwd(ruta)`

Cambia la ruta de trabajo según el parámetro `ruta`. Este será una cadena de caracteres con una ruta relativa o absoluta válida.

En el siguiente ejercicio se muestra cómo usar estas funciones para cambiar temporalmente la ruta de trabajo. `rutaPrevia` es una variable en la que guardamos la ruta actual y `<-` es el operador de asignación en R.

**Ejercicio 1.5** Obtención y modificación de la ruta de trabajo actual

```
> getwd()

[1] "D:/FCharte/Estudios/CursoUNIA/book"

> rutaPrevia <- getwd()
> setwd('../data')
> getwd()

[1] "D:/FCharte/Estudios/CursoUNIA/data"

> # Trabajar con los datos y después restablecer la ruta previa
> setwd(rutaPrevia)
```

### 1.4.3 Guardar y recuperar el espacio de trabajo

Cuando se utiliza un módulo de código R, lo corriente es que este contenga todo lo necesario para cargar los datos que necesita, procesarlos, etc., sin depender de un estado previo. Durante una sesión de trabajo interactiva en R, por el contrario, iremos creando objetos que probablemente necesitemos en una sesión posterior. Si recrear dichos objetos requiere un cierto tiempo, porque sean obtenidos a partir de cálculos, almacenar su estado actual a fin de recuperarlo cuando se necesiten resultará mucho más eficiente.

Almacenar objetos R también nos permitirá eliminar dichos objetos de la memoria, por ejemplo antes de crear otros que precisen mayor espacio de memoria libre. En R no existe un mecanismo de liberación automática de objetos cuando estos no son necesarios, como en otros lenguajes de programación, ya que durante una sesión de trabajo interactiva es imposible saber si el usuario requerirá dichos objetos o no más adelante.

A fin de guardar, eliminar y recuperar objetos recurriremos a las tres siguientes funciones:

#### Sintaxis 1.7 `save(objetos, file = archivo)`

Guarda uno o más objetos en el archivo indicado, usando un formato de archivo binario y comprimido. El parámetro `file` establece la ruta y nombre del archivo en que se guardarán los objetos. Habitualmente la extensión para archivos que almacenan objetos R es `.RData`.

#### Sintaxis 1.8 `rm(objetos)`

Elimina de la memoria los objetos facilitados como parámetros, liberando la memoria que estos tuviesen asignada.

#### Sintaxis 1.9 `load(archivo)`

Recupera los objetos almacenados en un archivo de datos R y los incorpora al entorno actual. El parámetro facilitado ha de incluir la extensión del archivo y, si fuese preciso, también la ruta donde se encuentra.

En el siguiente ejercicio<sup>9</sup> se muestra cómo usar estos métodos durante una hipotética sesión de trabajo con R:

#### Ejercicio 1.6 Guardar objeto en un archivo, eliminar el objeto y recuperarlo

```
> rutaPrevia

[1] "D:/FCharte/Estudios/CursoUNIA/book"

> save(rutaPrevia, file = 'ruta.RData')
> rm(rutaPrevia)

> rutaPrevia

Error in eval(expr, envir, enclos) : objeto 'rutaPrevia' no
encontrado
```

<sup>9</sup>La mayoría de los ejercicios propuestos en cada capítulo de este libro dependen de los propuestos anteriormente. En este caso, por ejemplo, se asume que el objeto `rutaPrevia` ya existe, puesto que fue creado en el ejercicio previo.

```
> load('ruta.RData')
> rutaPrevia


[1] "D:/FCharte/Estudios/CursoUNIA/book"
```

Almacenar variables individuales tiene sentido en casos concretos, por ejemplo al trabajar con objetos complejos generados tras cálculos más o menos largos. De esta forma se puede recuperar el objeto fruto de ese procesamiento siempre que sea necesario, sin volver a realizar todos los cálculos. Habrá ocasiones, sin embargo, en los que necesitemos guardar todos los objetos existentes en la sesión de trabajo en curso, antes de cerrar la consola, a fin de poder continuar trabajando en una sesión posterior en el punto en el que lo habíamos dejado, sin partir de cero.

Aunque podríamos obtener una lista de los objetos existentes en el entorno y guardarlos con la anterior función `save()`, nos resultará más cómodo recurrir a la siguiente función:

**Sintaxis 1.10** `save.image()`

Guarda todos los objetos existentes en el espacio de trabajo actual en un archivo llamado `.RData`. Dicho archivo se creará en la ruta actual (la devuelta por `getwd()`).

 En realidad no tenemos que invocar a `save.image()` explícitamente justo antes de cerrar la consola o RStudio, ya que en ese momento la propia herramienta nos preguntará si deseamos guardar el estado de nuestro entorno de trabajo. Al responder afirmativamente se llamará a `save.image()`.

Al abrir RStudio en una sesión de trabajo posterior, lo primero que hará esta herramienta será cargar el estado de la sesión previa, recuperando el contenido del archivo `.RData` y recreando a partir de él todos los objetos que teníamos. De esta forma podremos seguir trabajando como si no hubiese existido interrupción alguna. También podemos cargar explícitamente dicho archivo mediante la función `load()` antes indicada.

Además de los objetos obtenidos a partir de la ejecución de sentencias R, también podemos guardar y recuperar la propia secuencia de órdenes que hemos usado. Trabajando en la consola de R, no tenemos más que usar la tecla de movimiento arriba para recuperar sentencias previas del historial. En caso de que estemos usando RStudio, el panel **History** nos ofrece más funciones de acceso al historial, tal y como se explica en [Pau13b]. El historial puede guardarse y recuperarse, usando para ello las dos siguientes funciones:

**Sintaxis 1.11** `savehistory([file = archivo])`

Guarda el contenido del historial de trabajo en el archivo indicado. Si no se facilita un nombre de archivo, por defecto se usará `.Rhistory`. Dicho archivo se creará en la ruta actual (la devuelta por `getwd()`).

**Sintaxis 1.12** `loadhistory([file = archivo])`

Recupera el contenido del historial de trabajo del archivo indicado. Si no se facilita un nombre de archivo, por defecto se usará `.Rhistory`. Dicho archivo se buscará en la ruta actual (la devuelta por `getwd()`) si no especifica otra explícitamente.

#### 1.4.4 Cargar e instalar paquetes

El paquete base de R, siempre disponible desde que abrimos la consola, incorpora un gran abanico de funcionalidades con las que podemos cargar datos de fuentes externas, llevar a cabo análisis estadísticos y también obtener representaciones gráficas. No obstante, hay multitud de tareas para las que necesitaremos recurrir a paquetes externos, incorporando al entorno de trabajo las funciones y objetos definidos en ellos. Algunos de esos paquetes ya se encontrarán instalados en el sistema, pero otros será preciso descargarlos e instalarlos.

##### Cargar un paquete

Solamente pueden cargarse aquellos paquetes que estén instalados en el sistema. Tenemos dos funciones para llevar a cabo esta acción:

###### Sintaxis 1.13 `library(nombrePaquete)`

Carga el paquete indicado si está disponible o genera un error en caso contrario. `nombrePaquete` puede ser una cadena de caracteres o el nombre del paquete tal cual.

###### Sintaxis 1.14 `require(nombrePaquete)`

Carga el paquete indicado si está disponible o devuelve `FALSE` en caso contrario. `nombrePaquete` puede ser una cadena de caracteres o el nombre del paquete tal cual.

Habitualmente usaremos `library()` cuando estemos trabajando de manera interactiva, mientras que en guiones suele utilizarse `require()` comprobando el valor devuelto y evitando la interrupción abrupta del *script* por no encontrar un paquete. En el siguiente ejercicio puede apreciarse la diferencia entre ambas funciones:

##### Ejercicio 1.7 Cargar un paquete

```
> library(utils) # El paquete está disponible, no hay problema
> library(openxlsx) # El paquete no está disponible

Error in library(openxlsx) : there is no package called 'openxlsx'

> require(openxlsx)
```

##### Comprobar si un paquete está instalado

Mediante la función `installed.packages()` se obtiene información sobre todos los paquetes instalados en R:

###### Sintaxis 1.15 `installed.packages()`

Devuelve una matriz con información relativa a los paquetes instalados actualmente.

En lugar de examinar visualmente todos los elementos devueltos por la anterior función, podemos usar la función `is.element()` para verificar la presencia de un paquete concreto:

###### Sintaxis 1.16 `is.element(elemento, conjunto)`

Comprueba si `elemento` aparece en el `conjunto` o no, devolviendo `TRUE` o `FALSE` respectivamente.



Utilizando las dos anteriores funciones es fácil escribir una propia que nos facilite la comprobación de si un paquete está instalado o no. Es lo que se hace en el siguiente ejercicio, en el que definimos una nueva función a la que llamamos `is.installed()`. Esta precisa como único parámetro el nombre del paquete que nos interesa, devolviendo `TRUE` o `FALSE` según corresponda.

**Ejercicio 1.8** Comprobar si un paquete está instalado

```
> is.installed <- function(paquete) is.element(  
+   paquete, installed.packages())  
> is.installed('XLConnect')  
  
[1] TRUE
```

**Instalar un paquete**

Cualquier paquete disponible en el repositorio oficial (CRAN) puede ser instalado fácilmente desde el propio R. No hay más que facilitar su nombre a la función `install.packages()`:

**Sintaxis 1.17** `install.packages(paquetes, repos=repositorio)`

Busca, descarga e instala los paquetes indicados por el parámetro `paquetes`. Por defecto se buscará en CRAN, pero es posible facilitar la dirección de otro repositorio mediante el parámetro `repos`.

En el siguiente ejercicio se comprueba si un cierto paquete está instalado antes de cargarlo, procediendo a su instalación si fuese preciso:

**Ejercicio 1.9** Cargar un paquete, verificando antes si está disponible e instalándolo en caso necesario

```
> if(!is.installed('sos')) # Ayuda para buscar en paquetes  
+   install.packages("sos")  
> library("sos")
```

En este ejercicio se ha utilizado el condicional `if` para comprobar el valor devuelto por la función `is.installed()`. El símbolo `!` es el operador NOT en R.

Uno de los problemas a los que solemos enfrentarnos los usuarios de R es cómo saber qué paquete debemos instalar para satisfacer una cierta necesidad. Es una situación en la que nos será de utilidad el paquete `sos` instalado en el ejercicio previo. Dicho paquete facilita una función que se encargará de buscar en todos los paquetes disponibles en CRAN la cadena que facilitemos como parámetro, generando como resultado una página web (que se abrirá automáticamente en nuestro navegador por defecto) con información de cada paquete y enlaces a páginas con documentación adicional. De esta forma es fácil decidir qué paquete nos interesa. El siguiente ejercicio muestra cómo utilizar la citada función:

**Ejercicio 1.10** Buscar información de paquetes relacionados con Excel

```
> findFn("excel")
```

