

Ráster con R

Danilo A. Verdugo Chaura

2022-02-08

Contents

1	Plataforma R	5
1.1	Historia y Origen	5
1.2	Diseño de la Plataforma R	8
1.3	RStudio	8
1.4	Características Principales	9
1.5	Instalación de Plataforma R y RStudio	9
1.6	Consideraciones Preliminares	10
2	Paquetes	19
2.1	Búsqueda	19
2.2	Instalación	20
2.3	Actualización	20
3	Sintaxis Básica de R	25
3.1	Operaciones Numéricas	25
3.2	Asignación	28
3.3	Naturaleza Vectorial	30

Chapter 1

Plataforma R

1.1 Historia y Origen

R es un *sistema* o una *plataforma* que surge de un entorno computacional diseñado originalmente para el cálculo estadístico. Nace como un dialecto de S el cual es un lenguaje de programación desarrollado por John Chambers (Figura 1.1) y otros en la antigua “Bell Telephone Laboratorios”, originalmente miembro de AT & T Corp. S se inicia en 1976 como un programa interno de análisis estadístico. En 1988 el sistema fue reescrito desde FORTRAN a C y empezó a parecerse al sistema que tenemos hoy en día (esto fue en la versión 3 del lenguaje S). Lo importante de destacar es el espíritu original por el cual fue diseñado el sistema: facilitar el análisis de datos, primero para ellos y eventualmente, para otros.

John Chambers describe el anhelo al momento de diseñar el lenguaje S con que “los usuarios comenzaran en un entorno interactivo, donde no pensarán conscientemente en sí mismos como programadores. Entonces, cuando sus necesidades fuesen más claras y su sofisticación aumentara, deberían ser capaces de transitar gradualmente a la programación, cuando lenguaje y sistema serían más importantes”. La clave fue la transición de usuario a desarrollador. Ellos querían construir un lenguaje que fácilmente podría dar servicio a ambos tipos de usuario. Técnicamente, necesitaban construir un lenguaje que fuese adecuado para el análisis de datos interactivo (basada en línea de comandos), así como para la escritura de programas (como los lenguajes tradicionales de programación).

En 1991, Ross Ihaka y Robert Gentleman (Figura 1.2) en el Departamento de Estadística de la Universidad de Auckland crean R. Inspirado en la filosofía de S pero como un proyecto totalmente abierto. En 1993, se hace público. La experiencia de Robert y Ross sobre el desarrollo de R está documentada en un interesante artículo de 1996 en el Journal of Computational and Graphical Statistics [Ihaka and Gentleman, 1996].



Figure 1.1: John Chambers creador de S.



Figure 1.2: Ross Ihaka y Robert Gentleman, los creadores de R.

Hoy R se ejecuta en casi cualquier sistema operativo o plataforma informática estándar ¡incluso en una playstation 3! Una característica interesante en muchos proyectos de código abierto son las versiones frecuentes. Estos días se produce una liberación anual importante, por lo general en octubre, donde se incorporan nuevas características principales y lanzadas al público. A lo largo del año, se realizarán lanzamientos de corrección de errores en pequeña escala, según sea necesario. Los lanzamientos frecuentes y ciclo de liberación regular indican un permanente y activo desarrollo del software donde se asegura que los errores serán tratados de una manera oportuna. Por supuesto, el árbol principal de código fuente de R se encuentra bajo el control de un pequeño grupo de desarrolladores. Al momento de escribir estas líneas (mayo 2021) nos encontramos en la versión 4.0.5 o **Shake and Throw**.

Es interesante notar el dato anecdótico que los nombres propios con que se han designado desde la primera versión liberada *Great Pumpkin* (versión 2.14.0, noviembre 2011) o *Trick or Treat* (versión 2.15.2, octubre 2012) entre muchas otras hacen referencia a un capítulo específico de la serie animada *Peanuts* (Figura 1.3).



Figure 1.3: Serie animada Peanuts (Snoopy) basada en los comics de Charles Schulz.

Otra de las ventajas clave que R posee avanzadas capacidades gráficas con *calidad de publicación* cuya existencia se ha asegurado desde el principio del proyecto mediante el control muy fino en todos los aspectos de la composición gráfica.

1.2 Diseño de la Plataforma R

El éxito alcanzado por el proyecto R no tiene nada que ver con las herramientas en sí mismas, sino más bien con lo activa que sea la comunidad de usuarios, hoy miles de ellos en todo el mundo se han unido para realizar contribuciones como también ayudar a otros a usar R para todo tipo de nuevas e insospechadas aplicaciones.

El sistema R base o core o núcleo de la plataforma se compone de las herramientas mínimas para su funcionamiento que se encuentra disponible para varios sistemas operativos: Linux, Windows, Mac e incluso su Código Fuente en la denominada *Red Exhaustiva de Archivos R* (Comprehensive R Archive Network, CRAN).

cran.r-project.org, es una red de servidores ftp y web en todo el mundo que almacenan versiones idénticas y actualizadas de código y documentación para R)

1.3 RStudio

Aunque R posee un diseño muy avanzado e inteligente a medida que se realizan rutinas para análisis de datos, gráficos, o se generan archivos temporales de esos mismos procesos o se descargan e instalan diversos paquetes de código adicionales se hace muy difícil de mantener y menos aún generar procedimientos claros, organizados y replicables en distintos entornos informáticos, siendo esta última capacidad crítica para la aproximación científica al análisis de datos. RStudio también es una herramienta libre que hace más fácil el trabajo con R, van der Loo and De Jonge [2012] definen sus principales características como:

- Editor de texto, explorador de archivos, visualizador de gráficos todo en el mismo entorno.
- Trabaja directo con una instalación subyacente de R.
- Organiza el código y mantiene múltiples proyectos.
- Mantiene mi investigación reproducible.
- Mantiene los paquetes en la instalación de R.
- Crea y comparte reportes.
- Comparte el código y colabora con otros usuarios.

RStudio hoy es una aplicación y una fundación dedicada al soporte de sus productos de código abierto ¹ como al servicio comercial de capacitación y consultoría en temas estadísticos y R. Su fundador J.J. Allaire también inventa el lenguaje de programación web *ColdFusion*, *Windows Live Writer* (programa para escribir y publicar blog), *FitNow* y *LoseIt* ambas aplicaciones móviles para ejercicio y pérdida de peso entre otros proyectos tecnológicos.

¹www.rstudio.com

1.4 Características Principales

Algunas propiedades que hacen de RStudio el editor ideal para trabajar con R (tomado de van der Loo and De Jonge [2012]):

Integración con la consola R: Escribe comandos directamente en R dentro de RStudio.

Ejecución de código: Ejecuta código directo desde el editor.

Paréntesis inteligente: Cierre automático de paréntesis, ilumina selección, cierre de comillas automático. Termina de escribir la palabra previamente iluminada de un menú intellisense.

Ayuda en línea: Acceso directo a la ayuda de lenguaje y sintaxis.

Atajos de teclado: Tareas repetitivas son asociadas a combinación de teclas. Integra Ayuda, Permite navegar y buscar en las páginas de ayuda nativas de R.

Explorador de objetos e Historia: Se puede inspeccionar cada objeto creado en la sesión actual de R y su explorador de historia permite recorrer los comandos utilizados desde el actual hasta el primero de la sesión.

Navegar por el código: Saltar entre funciones, llamadas y reportes, visor de datos y visualización tipo grilla para explorar el contenido de los objetos en la sesión actual.

Menús de Importar datos: Para los tipos más comunes de archivo, ofrece un sistema de menú que genera el código R necesario. Integración gráfica con zoom, paneo y capacidades de exportación.

Gestión de proyectos: Facilita el manejo y control de varios proyectos. Control de versión e integra los sistemas de control git y svn.

Generación de Documentos: Genera pdf, html y otros formatos de reporte usando RMarkdown, Sweave o knitr.

Publicación: Publicar reportes y rutinas directo a la web de *Rpubs.com*.

1.5 Instalación de Plataforma R y RStudio

Primero se debe descargar el paquete de instalación desde el sitio web www.r-project.org siguiendo el link **CRAN** y buscar en el listado de servidores espejo el más cercano a la ubicación del usuario. Luego, seleccionar la versión correspondiente al sistema operativo de la máquina a instalar. A continuación, prestar atención y ubicar el link **install R for the first time** en el apartado *base*.

Finalmente, encontramos el link para la descarga del paquete instalador, siempre será la versión más reciente. Ejecutar el programa y seguir los pasos indicados por el asistente.²

Una vez terminada la instalación de R procedemos a instalar el programa RStudio desde el sitio www.rstudio.com y buscar en la lista de productos *RStudio Desktop*. A continuación, seleccionar la versión *Open Source Edition* que es

²Es importante señalar que el *nombre de usuario* de la sesión Windows donde se realizará la instalación **no existan** espacios e idealmente no contenga tildes. De ser así, es recomendable crear una nueva sesión con un nombre sin espacios ni tildes.

libre de pago. Finalmente seguir la secuencia de botones correspondiente hasta llegar al punto de descarga del paquete de instalación. Poner especial atención en seleccionar el que corresponda a su versión de sistema operativo. Ejecutar el programa y seguir los pasos indicados por el asistente.

1.6 Consideraciones Preliminares

El entorno de trabajo de RStudio posee una configuración por defecto que presenta una distribución de funcionalidades orientada al uso como entorno interactivo de cálculo y proceso de datos mediante línea de comandos contenido en la ventana *Console* y una vista general de los objetos y variables creadas en la sesión actual en la ventana *Environment* (Figura 1.4).

El tipo de trabajo descrito en el presente texto utiliza metodologías más complejas y extensas que trabajar solo como calculadora o proceso de datos. Por lo tanto, vamos a sugerir una nueva distribución de ventanas que optimicen el trabajo en un tipo de archivos que permiten ir escribiendo largas secuencias de comandos que van desarrollando las tareas de manera secuencial y por seguridad buscamos grabar en disco un archivo que la próxima vez me permita repetir dicha secuencia y continuar o modificar según el caso. Los archivos creados tendrán la extensión *.R* y los denominaremos *scripts*.

Una mejor disposición de ventanas es la sugerida en la figura 1.5. Donde se privilegia el panel izquierdo para extender a altura completa el editor de texto y maximizar el área disponible para escribir y editar scripts y mantiene el panel derecho la consola con su línea de comando.

Otro aspecto importante es el uso de un color de fondo oscuro que reduce la fatiga visual cuando se sienta frente al monitor durante períodos muy prolongados.

Otra característica configurada por defecto es que al momento de salir de una sesión, es guardar el *espacio de trabajo* lo que significa que se crea una imagen de las variables y funciones actuales en un archivo llamado ".RData".

Cuando se vuelve a abrir R desde el mismo directorio de trabajo, el espacio de trabajo se cargará y todas estas cosas estarán disponibles. Pero no recomendamos ese comportamiento. Cargar un espacio de trabajo guardado convierte el script cuidadosamente escrito donde todo sucede lógicamente de acuerdo con un plan a algo parecido a un cajón de sastre, lleno de páginas y cuadernos variados que pueden ser o no pertenecientes al trabajo actual.

Para configurar nuestro programa recomendamos cambiar algunos parámetros en la configuración general de la aplicación, para ellos debemos ir a la opción de menú **Tools/Global Options**. (Figura 1.6).

El cuadro de dialogo resume un gran número de parámetros que abarcan detalles del funcionamiento general de RStudio (Figura 1.7).

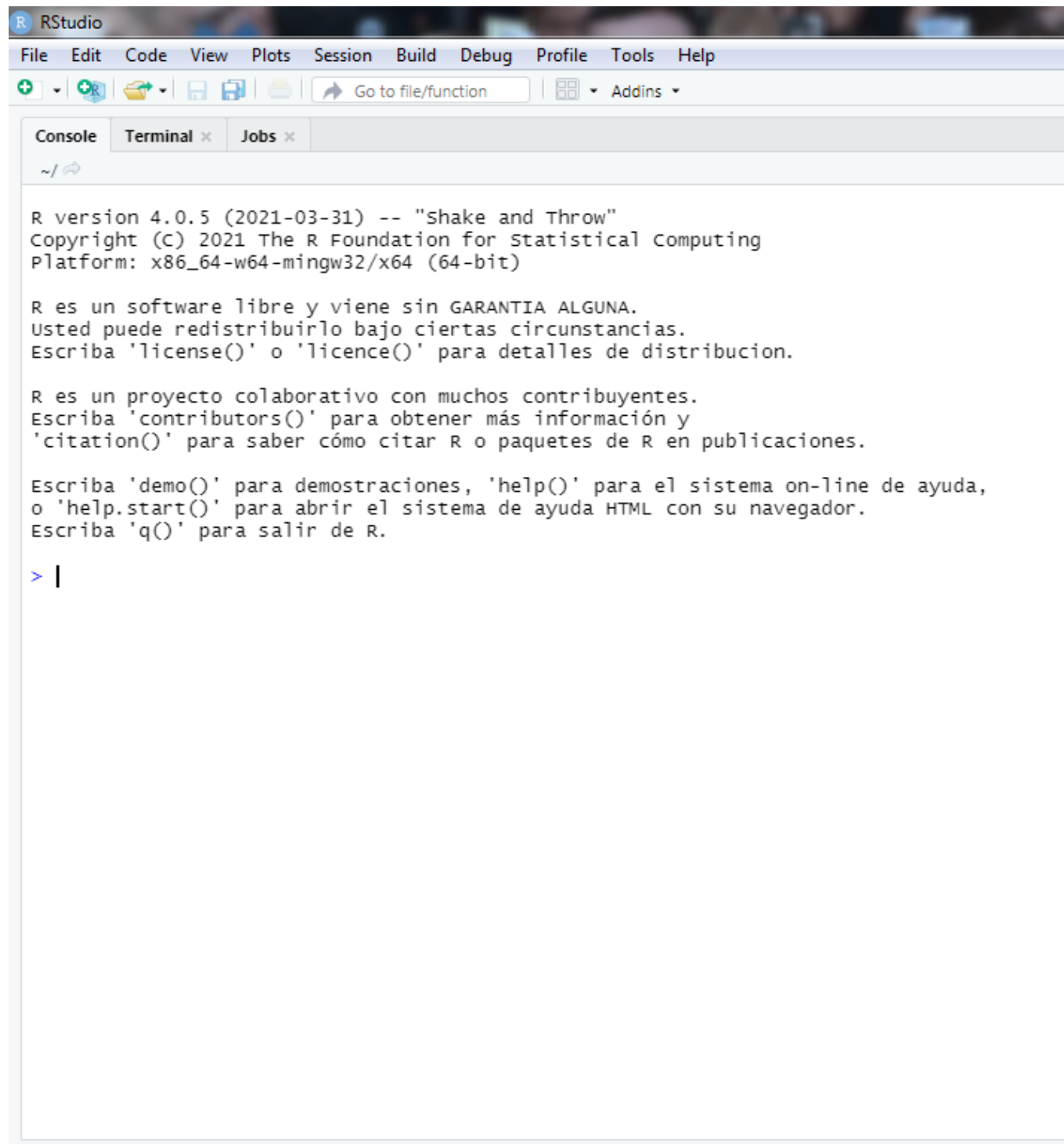


Figure 1.4: Configuración por defecto.

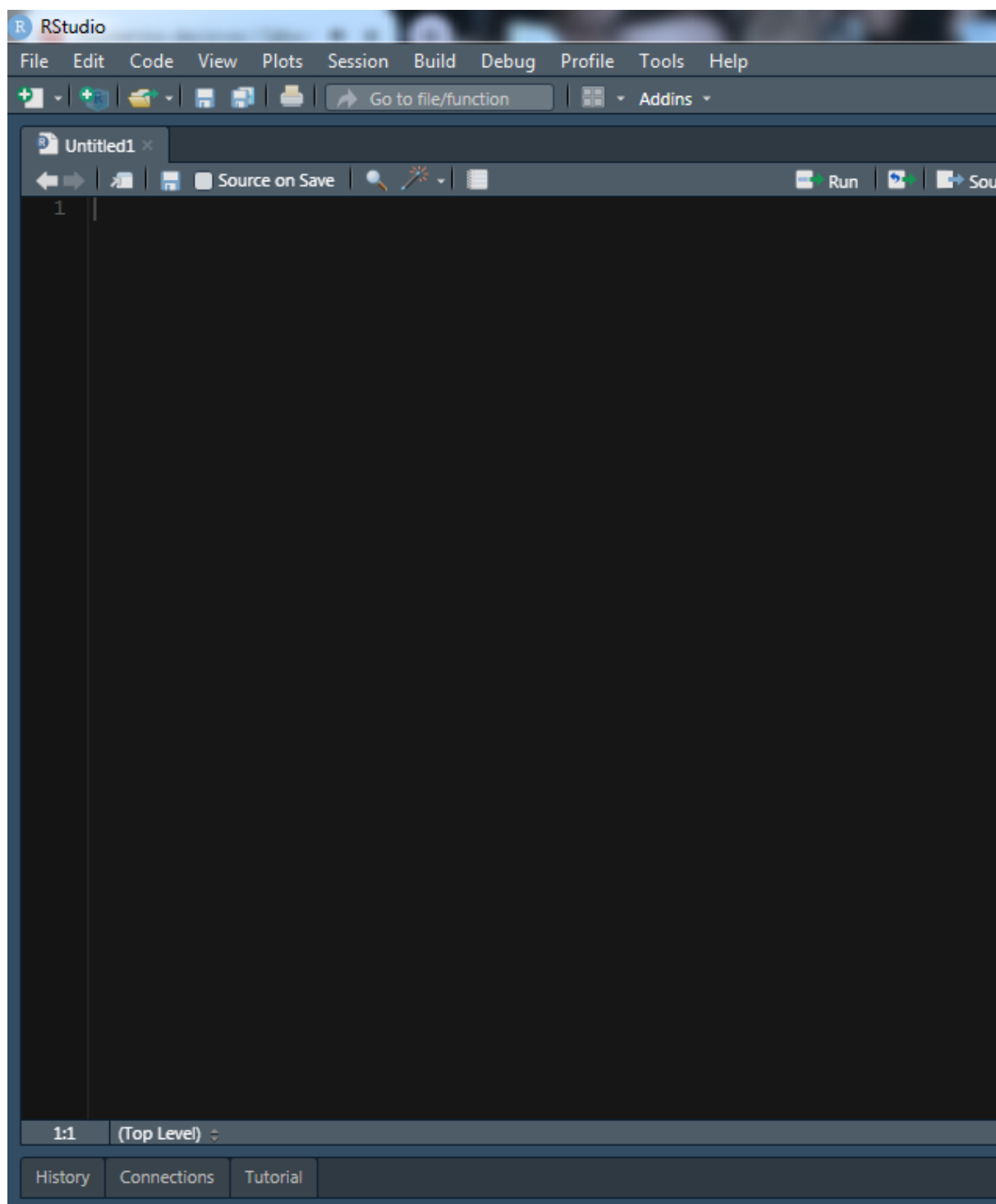


Figure 1.5: Configuración sugerida.

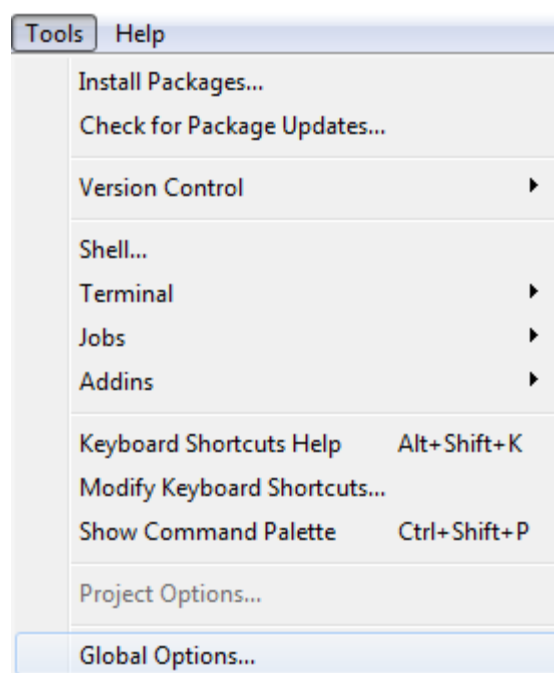


Figure 1.6: Opción de menú para configuración general.

Los siguientes cambios son necesarios para configurar los aspectos mencionados en punto anterior:

- *General*: Para quitar la carga automática de variables y funciones de la última sesión, debemos en el grupo **Workspace** quitar el check de *Restore .RData into...* y seleccionar la opción **Never** de la lista *Save workspce to .RData...* (Figura @ref(fig:img_gui_op_1)).
- *Appearance*: La configuración de colores en el editor de texto y los otros cuadros se selecciona en la lista *Editor theme*: y se debe tener especial cuidado en elegir aquel con un fondo negro o muy oscuro (Figura 1.8).
- *Pane Layout*: La distribución de los espacios disponibles para optimizar el editor de texto y mantener la línea de comando visible se configura siguiendo las opciones en indicadas en la figura 1.9).

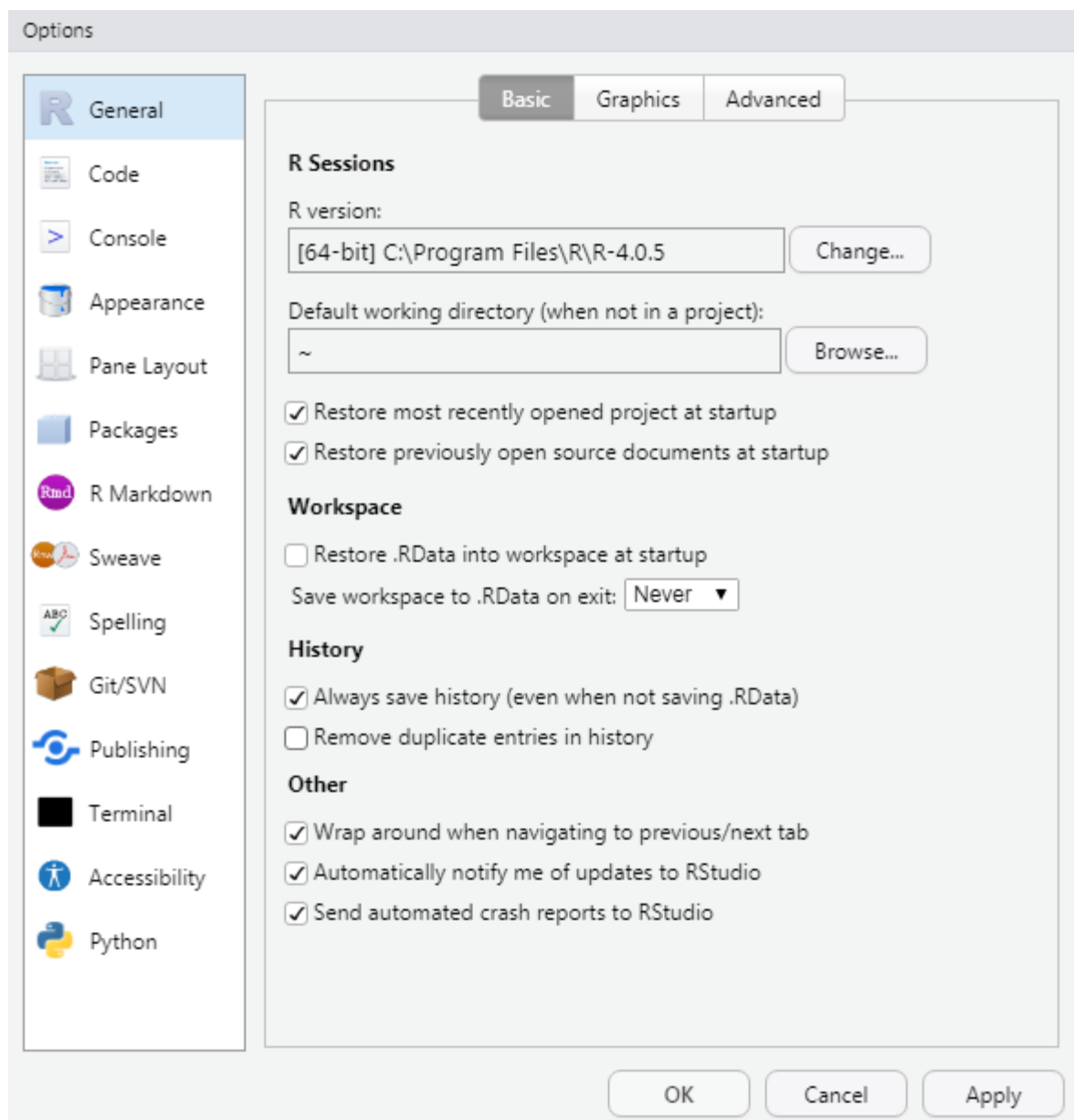


Figure 1.7: Panel de configuración general.

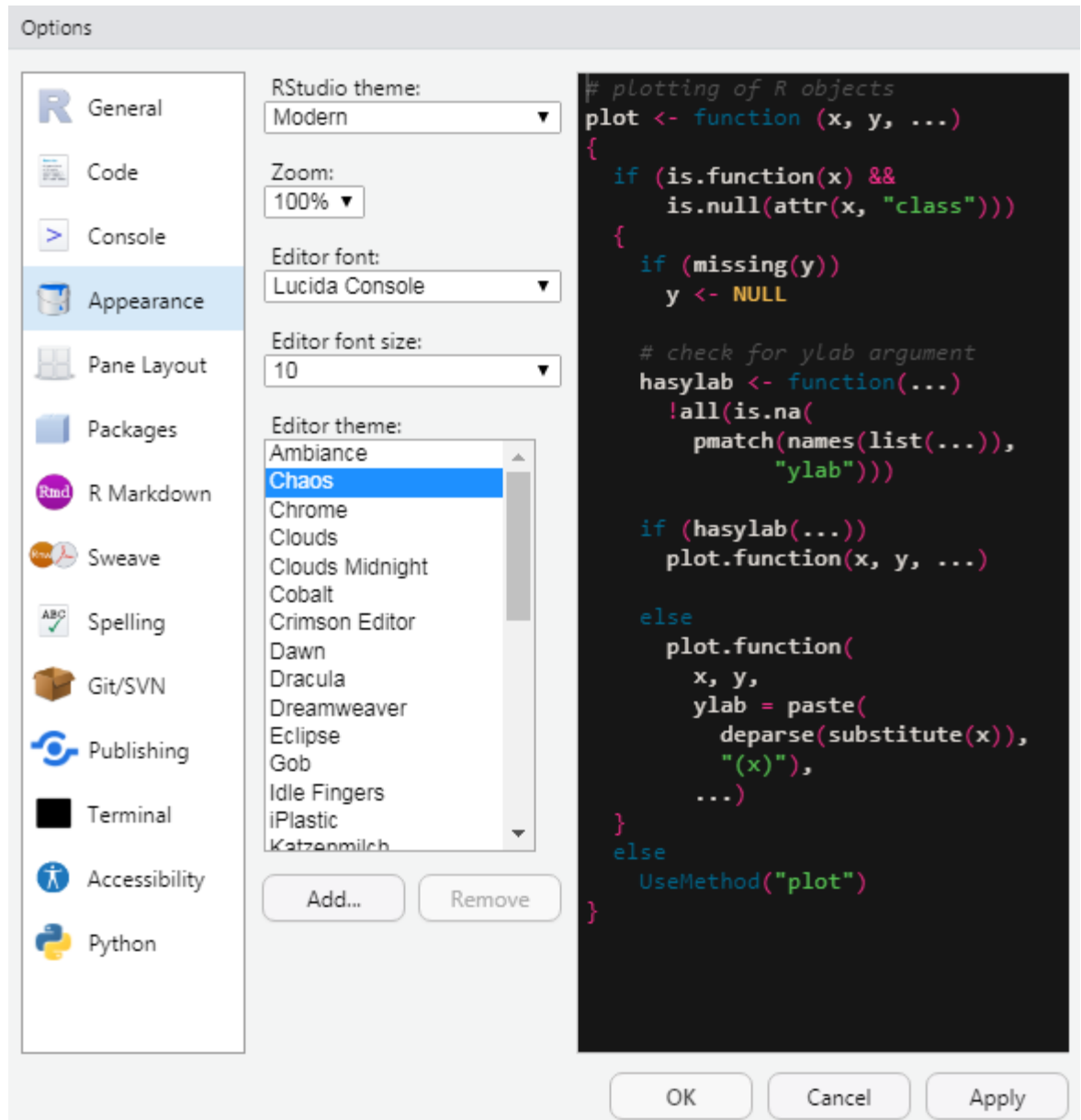


Figure 1.8: Selección para configuración de colores.

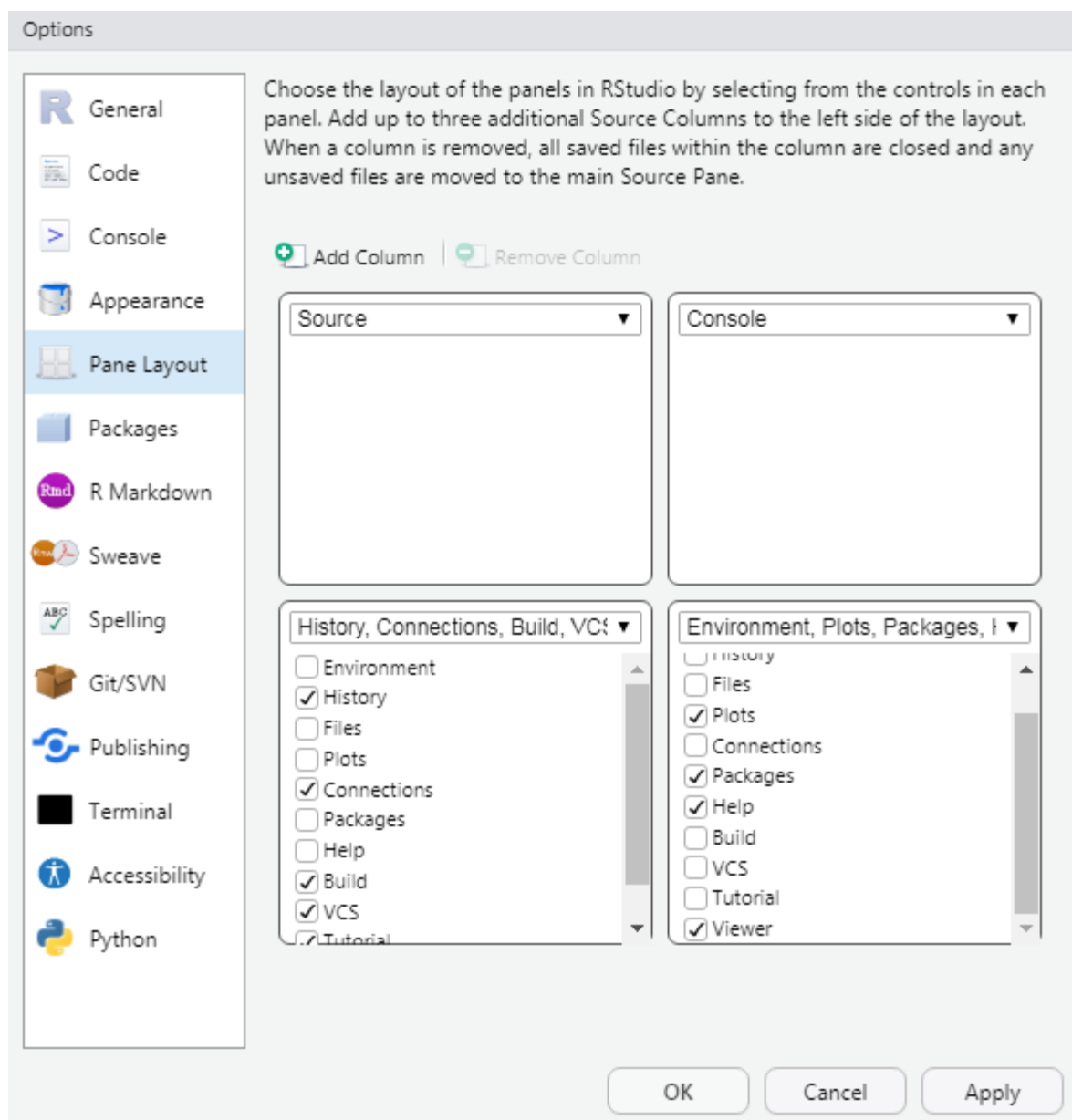


Figure 1.9: Configuración de paneles y cuadros.

Chapter 2

Paquetes

El sistema base, tal como se describe en 1.2, es capaz de ejecutar rutinas y código encapsulado en forma de módulos o paquetes (*packages*) que amplifican y diversifican las posibilidades de cómputo, gráfica, conexiones remotas, generación de documentos y un gran, gran etc.

El **sistema base o core** también se forma de un conjunto de paquetes.

Además de las funciones más fundamentales el sistema base incluye los paquetes `utils`, `stats`, `datasets`, `graphics`, `grDevices`, `grid`, `methods`, `tools`, `parallel`, `compiler`, `splines`, `tcltk`, `stats4` y algunos paquetes *recomendados*: `boot`, `class`, `cluster`, `codetools`, `foreign`, `KernSmooth`, `lattice`, `mgcv`, `nlme`, `rpart`, `survival`, `MASS`, `spatial`, `nnet`, `Matrix`.

Cuando se descarga una instalación nueva de R desde CRAN, se obtiene todos los paquetes mencionados, que representa un gran porcentaje de la funcionalidad del sistema.

2.1 Búsqueda

Además de la gran cantidad de paquetes instalados por defecto, existe una gran cantidad de paquetes opcionales disponibles.

- A noviembre 2021 existen 18.407 paquetes en CRAN que han sido desarrollados por los usuarios y programadores alrededor del mundo cada uno de ellos diseñado para un campo en especial.
- También hay muchos paquetes asociados con el proyecto **Bioconductor**¹, (detalles en ??).

¹www.bioconductor.org



- Las personas a menudo hacen sus propios paquetes y los hacen disponibles en sus sitios web personales; en realidad no hay manera confiable de realizar un seguimiento de cuántos paquetes están disponibles de este modo.
- Hay una serie de paquetes que se desarrollan en los repositorios de GitHub y BitBucket pero no existe ninguna lista fiable de todos estos paquetes.

El mejor método para buscar en la web, nuevos paquetes es utilizar el sitio rseek.org (Figura 2.1) usando palabras claves y potenciado por el motor de búsqueda de Google.

Generalmente los resultados siempre serán múltiples y variados.

2.2 Instalación

Una vez identificado un paquete que ofrece las funcionalidades deseadas se debe instalar en nuestro ambiente R y dejarlo disponible para cuando se necesite. El entorno RStudio cuenta con un panel específico (Figura 2.2).

Que junto al listado de paquetes ya instalados se dispone de opciones para instalar  **Install** y actualizar  **Update** (2.3) los existentes.

2.2.1 Cuadro Diálogo


El botón **Install** levanta un cuadro de diálogo con las opciones de configuración (Figura 2.3) para instalar paquetes disponibles en CRAN o los paquetes descargados en formato *.zip* de sitios no oficiales.


2.2.2 Función *install.packages()*

Un segundo método de instalación es utilizar la función R escrita directamente en la consola y reemplazando *nombre* por el paquete deseado.

```
install.packages("nombre")
```

2.3 Actualización

El botón  **Update** del panel paquetes levanta el cuadro de diálogo diseñado para la selección de los paquetes a actualizar. La lista solo muestra aquellos disponibles de actualización (Figura 2.4).



2

raster

All results Issues Function Wikipedia R-project Vignette Blog Documentation Source [Package](#)

RStudio

About 626 results (0.23 seconds) Sort by

CRAN - Package raster
[cran.r-project.org](https://cran.r-project.org/package=raster) › [package=raster](#)
 Oct 11, 2021 ... **raster**: Geographic Data Analysis and Modeling. Reading, writing, manipulating, analyzing and modeling spatial data. The **package** ...
 Labeled [Package](#) [R-project](#) [Spatial](#) [SpatioTemporal](#)


CRAN - Package rasterImage
[cran.r-project.org](https://cran.r-project.org/package=rasterImage) › [package=rasterImage](#)
 Oct 14, 2019 ... **rasterImage**: An Improved Wrapper of image(). This is a wrapper function for image(), which makes **raster** plots with nice axis and ...
 Labeled [Package](#) [R-project](#)

CRAN - Package rasterKernelEstimates
[cran.r-project.org](https://cran.r-project.org/package=rasterKernelEstimates) › [package=rasterKernelEstimates](#)
 Aug 4, 2016 ... Performs kernel based estimates on in-memory **raster** images from the **raster package**. These kernel estimates include local means variances, ...
 Labeled [Package](#) [R-project](#)


CRAN - Package rts
[cran.r-project.org](https://cran.r-project.org/package=rts) › ...
 rts: **Raster** Time Series Analysis. This framework aims to provide classes and methods for manipulating and processing time series data (e.g. a time ...
 Labeled [Package](#) [R-project](#)

rassta: Raster-Based Spatial Stratification Algorithms
[cran.r-project.org](https://cran.r-project.org/package=rassta) › ...
 Oct 14, 2021 ... Algorithms for the spatial stratification of landscapes, sampling and modeling of spatially-varying phenomena. These algorithms offer a simple ...
 Labeled [Package](#) [R-project](#)

1 2 3 4 5

 Search for **raster** on Google ENHANCE

Created and maintained by [Sasha Goodman](#).
 Serving the R community since 2007. Version 2.0.

 [Privacy Policy](#)

[Download and Install R](#)

Figure 2.1: Sitio web de búsqueda de contenido asociado a R, modo 'Package'.

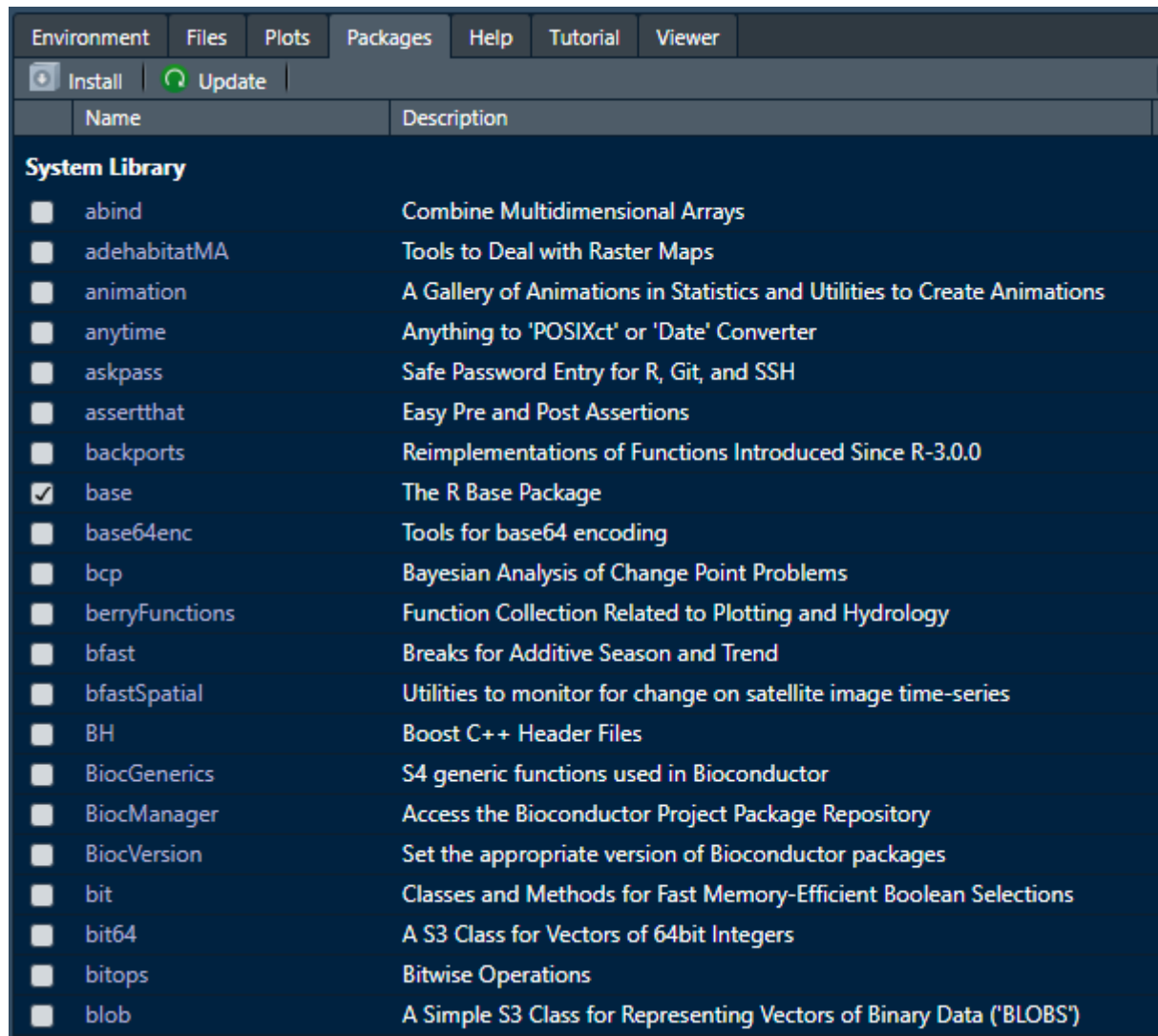


Figure 2.2: Panel Packages.

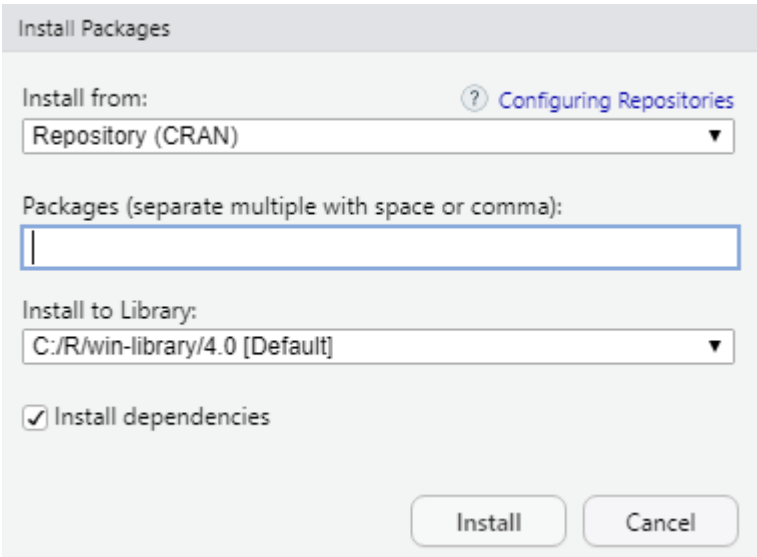


Figure 2.3: Opciones de Instalación vía RStudio.

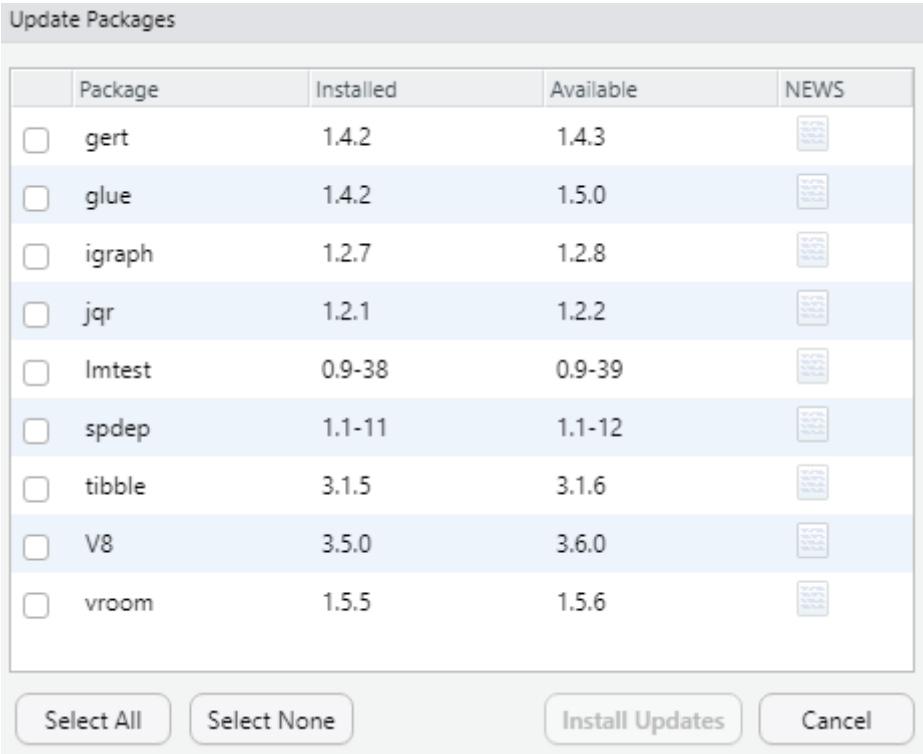


Figure 2.4: Opciones de Instalación vía RStudio.

Chapter 3

Sintaxis Básica de R

3.1 Operaciones Numéricas

De acuerdo con la configuración de ventanas descrita en el punto anterior la ventana *Console* se encuentra en la esquina superior derecha junto a las ventanas *Terminal* y *Jobs*. En ella trabajaremos interactivamente mediante la línea de comando, su uso puede ser comparado con el uso de una calculadora electrónica que siempre está a la espera del ingreso de una instrucción a continuación del símbolo `>`.

La dinámica de uso es ir ingresando el texto que define la operación o instrucción a realizar y finalizar con la tecla **Intro**:

```
1 + 2 + 3
```

Y R responde con la evaluación inmediata de la operación:

```
## [1] 6
```

Los espacios en blanco son removidos automáticamente y la orden es evaluada normalmente:

```
1+2      +      3
```

```
## [1] 6
```

Cuando la instrucción se extiende por más de una línea, R se encarga de interpretar la estructura y completitud para ser evaluada:

```
12 + 3 -  
  5 +  
  5
```

```
## [1] 15
```

También puedes usar el símbolo ‘punto y coma’ (;) para separar secuencias de instrucciones, pero escritas en la misma línea de código.

El editor ofrece algunas facilidades al usuario como el *historial de órdenes* de las instrucciones digitadas en orden reverso para evitar el retipeo de una instrucción muy extensa. Recorrer con las flechas verticales hasta encontrar el texto buscado y solo editar los cambios requeridos. También cuenta con un potente *motor de finalización inteligente* que permite obtener ayuda a medida que se escribe nombre de funciones o comandos y cuando se trabaja con scripts también nombres de funciones o variables. Por último, para limpiar todo el texto escrito presionar la tecla *escape*.

Se pueden utilizar los símbolos aritméticos tradicionales $+$, $-$, $*$, $/$ y $^$ ¹. Además se encuentran disponibles las funciones *log*, *exp*, *sin*, *cos*, *tan*, *sqrt* entre muchas otras bien conocidas. También existe el valor de *pi*.

```
1 + 2 - 3 * 4 / 5**6; sin(0.33); acos(0.253) ; sqrt(4)
```

```
## [1] 2.999232
```

```
## [1] 0.324043
```

```
## [1] 1.315016
```

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

```
9 %% 4
```

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

```
13.5 %/% 2
```

```
## [1] 6
```

```
pi
```

```
## [1] 3.141593
```

También se debe tener en cuenta que los números muy grandes o pequeños serán automáticamente convertidos a notación científica.

```
1/1000000000000000000
```

```
## [1] 1e-17
```

```
1*1000000000000000000
```

```
## [1] 1e+17
```

En operaciones más complejas se recomienda el uso de *()* para fijar el orden o *precedencia* de las operaciones. Se acepta *anidar* los paréntesis *(())*.

¹El acento circunflejo $^$ se utiliza para la potenciación (se obtiene combinando las teclas alt+94) también se puede reemplazar con $**$. Además, existe la *operación módulo* ($%%$) y la *división entera* ($%/%$).

```
3 * 5 + 1
```

```
## [1] 16
```

```
3 * (5 + 1)
```

```
## [1] 18
```

```
1 + (3 + (1 + 2) / 3)
```

```
## [1] 5
```

Las operaciones lógicas se realizan utilizando ciertos símbolos o combinaciones de ellos para obtener resultados lógicos:

```
2 == 3
```

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

```
2 < 3
```

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

Revisa cuadro ?? para una lista completa de los caracteres y su significado en las operaciones lógicas.

Otros componentes del lenguaje R son: valor lógico verdadero *TRUE* y falso *FALSE*, infinitud positiva **Inf** y negativa **-Inf**, no disponible **NA**, no es un número **NaN** y nulo o vacío **NULL**.

Finalmente, no podemos intercambiar mayúsculas o minúsculas al momento de escribir los nombres de instrucciones (por ej. *sin* no es igual a *Sin*). Y en R todo es un *objeto* que pertenecen a cierto tipo de *clases*. Así cada uno de ellos tendrá comportamientos y usos diferentes.

```
class(1)
```

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

```
class("palabra")
```

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

```
class(sqrt)
```

```
## [1] "function"
```

```
class(1 != 1)
```

```
## [1] "logical"
```

```
class(TRUE)
```

```
## [1] "logical"
```

```
class(Sys.Date())
```

```
## [1] "Date"
```

3.2 Asignación

El proceso de almacenar cualquiera de los objetos ya descritos, o el resultado de las operaciones en memoria RAM y hacerlo persistente a lo largo de la sesión actual de trabajo se denomina *asignación*. Resultados intermedios se deben ir asignado con un nombre fácilmente identificable y memorizables.²

Es conveniente adoptar un estilo de escritura para los nombres de los objetos y para ellos existen varios métodos entre los cuales podemos destacar:

- ***lowercase***: todominusculas.
- ***UPPERCASE***: TODOMAYUSCULAS.
- ***UpperCamelCase***: PrimeraMayusculaSiguientesBajas.
- ***lowerCamelCase***: primeraMinusculaSiguientesMayusculas.
- ***period.separated***: separada.por.punto.
- ***snake_case***: separada_por_guion_bajo.

En el sitio web style.tidyverse.org donde se discuten buenas prácticas para el uso de R se recomienda el último tipo.

La forma de realizar una asignación tiene la forma general:

$$\text{nombre} = \text{valor} \quad (3.1)$$

Donde *nombre* es el identificador o nombre del objeto o variable que se busca crear y almacenar en memoria, *=* el operador de asignación (no confundir con *==*) y *valor* es el contenido a almacenar.³

```
radio_estimado = 135.45
```

nombre será sintácticamente válido cuando consta de letras, números o punto y comienza con **una letra o el punto**, pero *no seguido de un número*. Los nombres como “2var” no son válidos, ni tampoco las palabras reservadas.

En la actualidad el uso *<-* o *=* es casi indistinto, pero en casos muy específicos (y técnicos) existen diferencias. En nuestro libro recomendamos el uso de *<-* ya que es un poco más cómodo de tipear que el carácter *=*.

²Los nombres de los objetos deben en lo posible *resumir* el contenido, evitar nombres *genéricos* como *a* u *X* y no ambiguos como *variable* o *constante*. Pueden contener letras, números, puntos y guiones o *barra baja* (*_*). Siempre debe comenzar con una letra o punto.

³En R se ha utilizado tradicionalmente como operador de asignación *<-* que viene desde la primera versión de R, dado que su origen proviene del lenguaje **S** y a la vez éste fue inspirado por el lenguaje *APL* (A Programming Language, *Kenneth Iverson en 1957-62*) que utilizaba dicho carácter ya que el símbolo *=* se utilizaba solo para prueba de igualdad.

```
variable <- (112.33 - 45) / 105
```

Una vez escrita la asignación y presionado *intro* se realiza la operación, se crea el objeto en memoria con nombre *variable* y la línea de comando se limpia. Para conocer el resultado de la operación podemos:

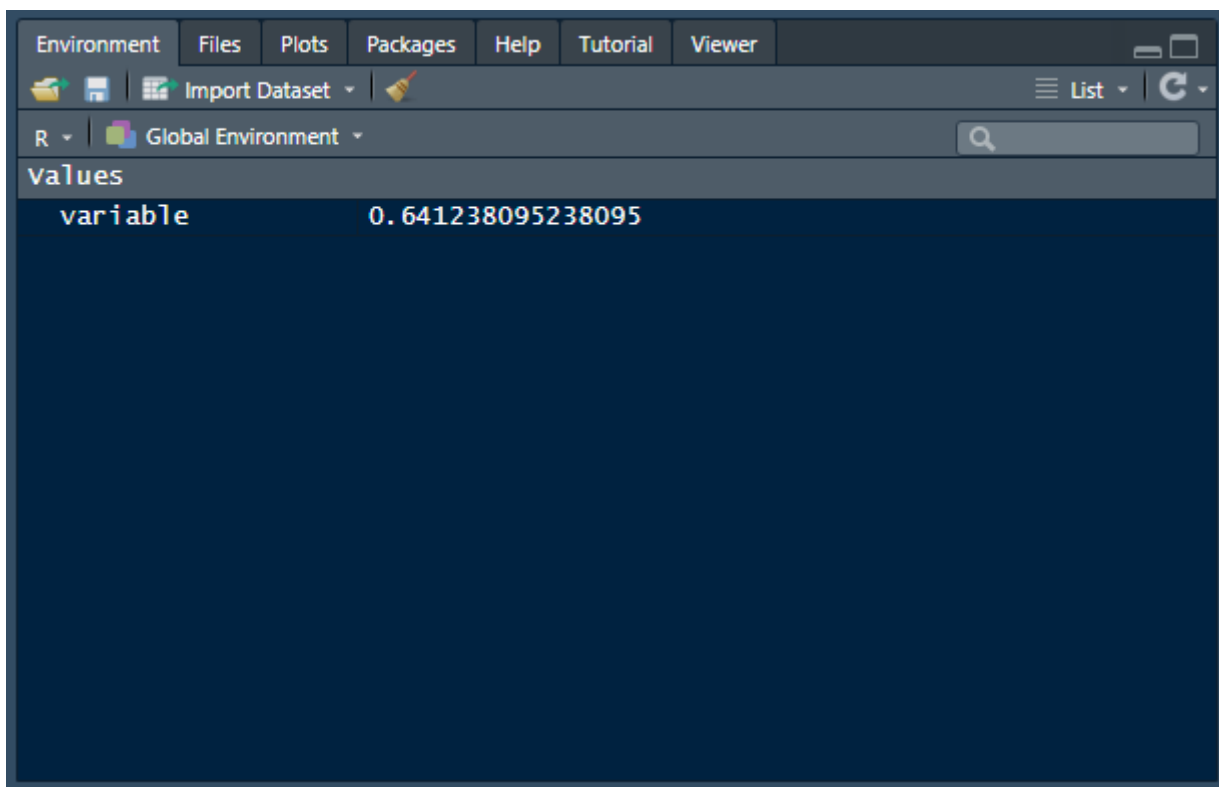


Figure 3.1: Panel Environment.

1. Buscar el resultado en el panel *Environment* en la esquina inferior derecha (Figura 3.1).
2. Escribir la asignación dentro de un juego de paréntesis redondos () que obliga imprimir el resultado de la evaluación de la expresión.

```
(variable <- (112.33 - 45) / 105)
```

```
## [1] 0.6412381
```

3. Volver a escribir el nombre y presionar intro.

```
variable
```

```
## [1] 0.6412381
```

4. Utilizar el comando `print()`.

```
print(variable)
```

```
## [1] 0.6412381
```

El principal objetivo de almacenar en memoria es la reutilización de dichos contenidos:

```
radio_inicial <- 15.43 + 36.315 * (3 * 5)
(area_base <- pi * radio_inicial ** 2)
```

```
## [1] 985748.9
```

Al asignar un nuevo contenido con el mismo nombre, el anterior se elimina de memoria y se crea un nuevo objeto.

```
x <- 5
x <- "hola"
x
```

```
## [1] "hola"
```

El comando `ls()` me permite listar todos los objetos almacenados en mi sesión actual y `rm(list = ls())` borrará la totalidad de objetos de mi sesión actual.

3.3 Naturaleza Vectorial

En todas las ocasiones que hemos escrito una operación aritmética o impreso un resultado R crea automáticamente un *vector*⁴ con una longitud que corresponde a los valores por almacenar.

Si estudiamos el ejemplo siguiente:

```
5
```

```
## [1] 5
```

```
ls()
```

```
## [1] "area_base"      "radio_inicial"  "variable"      "x"
```

Vemos un `[1]` antecediendo al número 5 que indica que se trata del primer (y único en este caso) elemento, mientras que en el caso del comando `ls()` la lista de respuesta por cada renglón que se extienda en la consola antecederá su correspondiente `[]` señalando la posición en el vector generado automáticamente.

⁴Un **Vector** es el tipo de objeto más básico del lenguaje R. Un *vector* puede contener cero o más objetos, y siempre serán de la misma clase.

3.3.1 Función `c()` para crear vectores

La creación manual de vectores usando elementos individuales se realiza mediante la función `c()`.

En el siguiente ejemplo hemos incorporado dos nuevos elementos:

- `;` (*punto y coma*) que nos permite separar dos instrucciones en la misma línea.
- `#` (*almohadilla o numeral*) sirve para crear COMENTARIOS: textos que no serán evaluados por R.

```
c(1, 4, 8, 11) #Sin asignar a nombre de objeto
```

```
## [1] 1 4 8 11
```

```
x <- c(2, 4, 6, 8); print(x)
```

```
## [1] 2 4 6 8
```

```
y <- c(x, 100, 101, 102); y
```

```
## [1] 2 4 6 8 100 101 102
```

```
z <- c("azul", "rojo", "verde", "amarillo"); (z)
```

```
## [1] "azul" "rojo" "verde" "amarillo"
```

3.3.2 Uso de Secuencias para crear vectores

Una secuencia numérica se construye usando el operador `:`. Por ejemplo:

```
1:30
```

Es equivalente a `c(1, 2, 3, ..., 30)`, también es capaz de crear secuencias negativas y/o usar valores decimales. Comprueba las siguientes operaciones y sus resultados:

```
30 : 1
```

```
-10 : -8
```

```
1.5 : 5
```

La prioridad del comando `:` es máxima y siempre será evaluado al inicio. Compara las secuencias obtenidas entre `1:5-1` y `1:(5-1)`.

3.3.3 Usar Repetición para la creación de vectores

El comando `rep()` genera un vector con la repetición de un valor un número determinado de veces. También se puede repetir un texto, una fecha o una secuencia definida con la opción `:`.

Revisar usando alguna de las formas de ayuda en línea los argumentos, nombres y usos para el comando `rep`:

```
rep(1, 5); rep('a', 5)

## [1] 1 1 1 1 1
## [1] "a" "a" "a" "a" "a"
rep(c(1:3, 7), 3); rep(x = 1:2, times = 3, each=3)

## [1] 1 2 3 7 1 2 3 7 1 2 3 7
## [1] 1 1 1 2 2 2 1 1 1 2 2 2 1 1 1 2 2 2
```

3.3.4 Uso de Patrones para crear vectores

El comando `seq()` genera patrones regulares. Posee múltiples parámetros o *argumentos con nombre* para configurar el patrón. Los dos primeros *desde (from)*, *hasta (to)* tendrían el mismo comportamiento que los parámetros del comando `:`, un tercero *paso (by)* define el incremento y posee un valor por defecto de 1 y un cuarto argumento define el número de elementos que debe contener la secuencia (*length.out*)⁵.

En los ejemplos siguientes vemos el uso del comando usando los argumentos en orden preciso *sin utilizar sus nombres*:

```
seq(1, 10) #from=2, to=10

## [1] 1 2 3 4 5 6 7 8 9 10
seq(1, 10, 2) #from=2, to=10, by=2

## [1] 1 3 5 7 9
```

También al poseer valores por defecto se puede abreviar su uso:

```
seq(to = 5)

## [1] 1 2 3 4 5
seq(from = 3)
```

⁵La mayoría de funciones en R poseen parámetros o argumentos con nombre que permiten ajustar el funcionamiento. El orden y sus nombres lo encuentras usando la ayuda en línea. En R el sistema es muy poderoso y fácil de utilizar, basta ubicar el cursor sobre el nombre de comando y presionar **F1** y un panel con información detallada con:

Descripción: Breve descripción de funcionamiento.

Uso: Sintaxis con el orden y nombres de argumentos.

Argumentos: Lista detallada y en orden correspondiente de los argumentos.

Detalles: Información adicional.

Valor: El objeto que retorna.

Ejemplos: Código R con alternativas de uso.

Será desplegada en el panel inferior derecho. También puedes escribir en la consola el símbolo de interrogación y a continuación el nombre del comando: `?seq` o usar el comando `help('seq')`.


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

Pero la verdadera utilidad de los argumentos con nombres es la libertad en el orden de uso:

```
seq(to = 9, from = 1)
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

```
seq(by = 1.5, from = 1, to = 9)
```

```
## [1] 1.0 2.5 4.0 5.5 7.0 8.5
```

```
seq(from = 10, by = -3, length.out= 5)
```

```
## [1] 10 7 4 1 -2
```

```
seq(by = 3, length.out= 6)
```

```
## [1] 1 4 7 10 13 16
```

La misma lógica se aplica al trabajo con fechas:

```
seq(as.Date("1910/1/1"), as.Date("1912/1/1"), "years")
```

```
## [1] "1910-01-01" "1911-01-01" "1912-01-01"
```

```
seq(from= as.Date("2000/1/1"), by = "month", length.out = 3)
```

```
## [1] "2000-01-01" "2000-02-01" "2000-03-01"
```

```
inicio <- as.Date("1999-12-17")
```

```
fin <- as.Date("2000-3-7")
```

```
seq(from= fin,
    to= inicio,
    by = "-1 month")
```

```
## [1] "2000-03-07" "2000-02-07" "2000-01-07"
```

3.3.5 Vectores Lógicos y Caracteres

Además de los vectores numéricos y fechas R soporta del tipo **lógico** y **texto**. En el caso de vectores lógicos solo puede tomar dos valores: **FALSE** (falso) y **TRUE** (verdadero), también son válidos **F** y **T**.

Generalmente son contruidos a partir del uso de los operadores lógicos (ver Cuadro ??) en *condiciones*. Por ejemplo:

```
enteros <- 1:6
(prueba <- enteros <=3)
```

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

Donde `prueba` almacena un vector de la misma longitud que enteros y cuyo contenido será tanto *True* o *False* dependiendo si o no cumplen la condición. También vectores de texto o caracteres pueden ser creados. Los textos o frases deben ser escritos entre comillas (") o comillas simples ('), por ejemplo `'Este texto está entre comillas.'`. Generalmente utilizada en la construcción de vectores con `c()`.

```
rep(x=c('x','y'), times= 4)
```

```
## [1] "x" "y" "x" "y" "x" "y" "x" "y"
```

Bibliography

Ross Ihaka and Robert Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996. doi: 10.1080/10618600.1996.10474713. URL <https://www.tandfonline.com/doi/abs/10.1080/10618600.1996.10474713>.

M.P.J. van der Loo and E. De Jonge. *Learning RStudio for R Statistical Computing*. Community experience distilled. Lightning Source, 2012. ISBN 9781782160601. URL <https://books.google.cl/books?id=vOTHmAEACAAJ>.