

Chapter 1

Obtaining R and Downloading Packages

This chapter is written for the user who has never downloaded R onto his or her computer, much less opened the program. The chapter offers some brief background on what the program is, then proceeds to describe how R can be downloaded and installed completely free of charge. Additionally, the chapter lists some internet-based resources on R that users may wish to consult whenever they have questions not addressed in this book.

1.1 Background and Installation

R is a platform for the object-oriented statistical programming language S. S was initially developed by John Chambers at Bell Labs, while R was created by Ross Ihaka and Robert Gentleman. R is widely used in statistics and has become quite popular in Political Science over the last decade. The program also has become more widely used in the business world and in government work, so training as an R user has become a marketable skill. R, which is shareware, is similar to S-plus, which is the commercial platform for S. Essentially R can be used as either a matrix-based programming language or as a standard statistical package that operates much like the commercially sold programs Stata, SAS, and SPSS.

Electronic supplementary material: The online version of this chapter (doi: [10.1007/978-3-319-23446-5_1](https://doi.org/10.1007/978-3-319-23446-5_1)) contains supplementary material, which is available to authorized users.

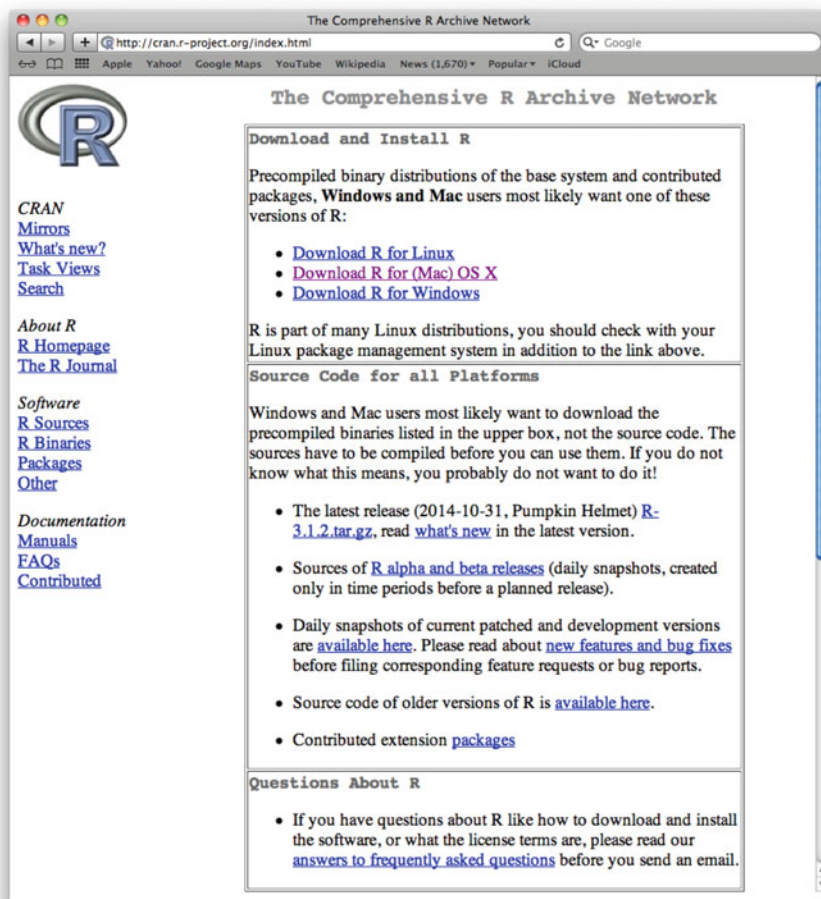


Fig. 1.1 The comprehensive R archive network (CRAN) homepage. The top box, “Download and Install R,” offers installation links for the three major operating systems

1.1.1 Where Can I Get R?

The beauty of R is that it is shareware, so it is free to anyone. To obtain R for Windows, Mac, or Linux, simply visit the comprehensive R archive network (CRAN) at <http://www.cran.r-project.org/>. Figure 1.1 shows the homepage of this website. As can be seen, at the top of the page is an inset labeled *Download and Install R*. Within this are links for installation using the Linux, Mac OS X, and Windows operating systems. In the case of Mac, clicking the link will bring up a downloadable file with the `pkg` suffix that will install the latest version. For Windows, a link named `base` will be presented, which leads to an `exe` file for

download and installation. In each operating system, opening the respective file will guide the user through the automated installation process.¹ In this simple procedure, a user can install R on his or her personal machine within five minutes.

As months and years pass, users will observe the release of new versions of R. There are not update patches for R, so as new versions are released, you must completely install a new version whenever you would like to upgrade to the latest edition. Users need not reinstall every single version that is released. However, as time passes, add-on libraries (discussed later in this chapter) will cease to support older versions of R. One potential guide on this point is to upgrade to the newest version whenever a library of interest cannot be installed due to lack of support. The only major inconvenience that complete reinstallation poses is that user-created add-on libraries will have to be reinstalled, but this can be done on an as-needed basis.

1.2 Getting Started: A First Session in R

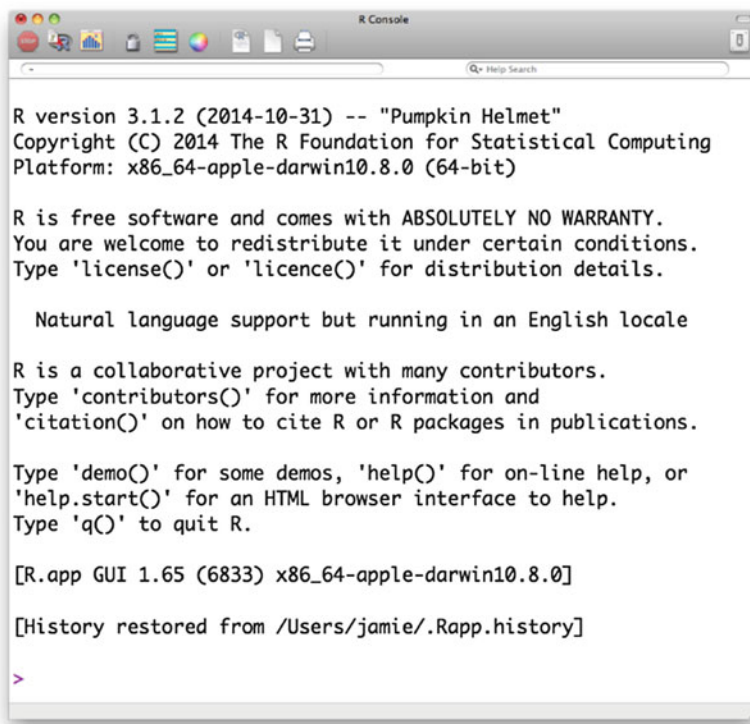
Once you have installed R, there will be an icon either under the Windows Start menu (with an option of placing a shortcut on the Desktop) or in the Mac Applications folder (with an option of keeping an icon in the workspace Dock). Clicking or double-clicking the icon will start R. Figure 1.2 shows the window associated with the Mac version of the software. You will notice that R does have a few push-button options at the top of the window. Within Mac, the menu bar at the top of the workspace will also feature a few pull-down menus. In Windows, pull down menus also will be presented within the R window. With only a handful of menus and buttons, however, commands in R are entered primarily through user code. Users desiring the fallback option of having more menus and buttons available may wish to install RStudio or a similar program that adds a point-and-click front end to R, but a knowledge of the syntax is essential. Figure 1.3 shows the window associated with the Mac version of RStudio.²

Users can submit their code either through script files (the recommended choice, described in Sect. 1.3) or on the command line displayed at the bottom of the R console. In Fig. 1.2, the prompt looks like this:

>

¹The names of these files change as new versions of R are released. As of this printing, the respective files are `R-3.1.2-snowleopard.pkg` or `R-3.1.2-mavericks.pkg` for various versions of Mac OS X and `R-3.1.2-win.exe` for Windows. Linux users will find it easier to install from a terminal. Terminal code is available by following the *Download R for Linux* link on the CRAN page, then choosing a Linux distribution on the next page, and using the terminal code listed on the resulting page. At the time of printing, Debian, various forms of Red Hat, OpenSUSE, and Ubuntu are all supported.

²RStudio is available at <http://www.rstudio.com>.



```
R version 3.1.2 (2014-10-31) -- "Pumpkin Helmet"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin10.8.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[R.app GUI 1.65 (6833) x86_64-apple-darwin10.8.0]

[History restored from /Users/jamie/.Rapp.history]

>
```

Fig. 1.2 R Console for a new session

Whenever typing code directly into the command prompt, if a user types a single command that spans multiple lines, then the command prompt turns into a plus sign (+) to indicate that the command is not complete. The plus sign does not indicate any problem or error, but just reminds the user that the previous command is not yet complete. The cursor automatically places itself there so the user can enter commands.

In the electronic edition of this book, input syntax and output printouts from R will be color coded to help distinguish what the user should type in a script file from what results to expect. Input code will be written in **blue teletype font**. R output will be written in **black teletype font**. Error messages that R returns will be written in **red teletype font**. These colors correspond to the color coding R uses for input and output text. While the colors may not be visible in the print edition, the book's text also will distinguish inputs from outputs. Additionally, names of variables will be written in **bold**. Conceptual keywords from statistics and programming, as well as emphasized text, will be written in *italics*. Finally, when the meaning of a command is not readily apparent, identifying initials will be underlined and bolded in the text. For instance, the **lm** command stands for **linear model**.

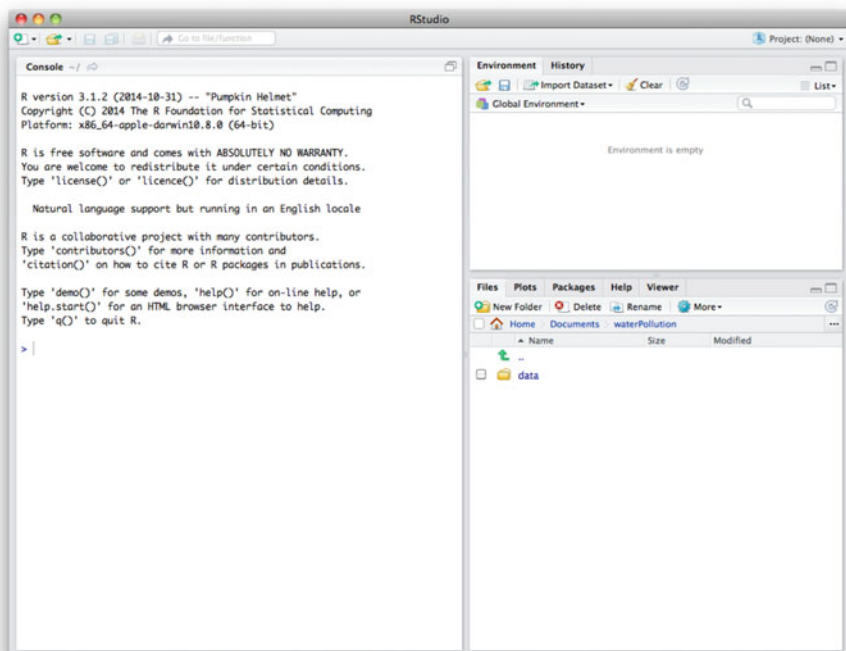


Fig. 1.3 Open window from an RStudio session

When writing R syntax on the command line or in a script file, users should bear a few important preliminaries in mind:

- Expressions and commands in R are case-sensitive. For example, the function `var` returns the variance of a variable: a simple function discussed in Chap. 4. By contrast, the `VAR` command from the `vars` library estimates a Vector Autoregression model—an advanced technique discussed in Chap. 9. Similarly, if a user names a dataset `mydata`, then it cannot be called with the names `MyData`, `MyData`, `MYDATA`, or `myData`. R would assume each of these names indicates a different meaning.
- Command lines do not need to be separated by any special character like a semicolon as in Limdep, SAS, or Gauss. A simple hard return will do.
- R ignores anything following the pound character (`#`) as a comment. This applies when using the command line or script files, but is especially useful when saving notes in script files for later use.
- An object name must start with an alphabetical character, but may contain numeric characters thereafter. A period may also form part of the name of an object. For example, `x.1` is a valid name for an object in R.
- You can use the arrow keys on the keyboard to scroll back to previous commands. One push of the up arrow recalls the previously entered command and places it in

the command line. Each additional push of the arrow moves to a command prior to the one listed in the command line, while the down arrow calls the command following the one listed in the command line.

Aside from this handful of important rules, the command prompt in R tends to behave in an intuitive way, returning responses to input commands that could be easily guessed. For instance, at its most basic level R functions as a high-end calculator. Some of the key *arithmetic* commands are: addition (+), subtraction (-), multiplication (*), division (/), exponentiation (^), the modulo function (%%), and integer division (%/%). Parentheses () specify the order of operations. For example, if we type the following input:

```
(3+5/78)^3*7
```

Then R prints the following output:

```
[1] 201.3761
```

As another example, we could ask R what the remainder is when dividing 89 by 13 using the modulo function:

```
89%%13
```

R then provides the following answer:

```
[1] 11
```

If we wanted R to perform integer division, we could type:

```
89%/13
```

Our output answer to this is:

```
[1] 6
```

The `options` command allows the user to tweak attributes of the output. For example, the `digits` argument offers the option to adjust how many digits are displayed. This is useful, for instance, when considering how precisely you wish to present results on a table. Other useful built-in functions from algebra and trigonometry include: `sin(x)`, `cos(x)`, `tan(x)`, `exp(x)`, `log(x)`, `sqrt(x)`, and `pi`. To apply a few of these functions, first we can expand the number of digits printed out, and then ask for the value of the constant π :

```
options(digits=16)
pi
```

R accordingly prints out the value of π to 16 digits:

```
[1] 3.141592653589793
```

We also may use commands such as `pi` to insert the value of such a constant into a function. For example, if we wanted to compute the sine of a $\frac{\pi}{2}$ radians (or 90°) angle, we could type:

```
sin(pi/2)
```

R correctly prints that $\sin(\frac{\pi}{2}) = 1$:

```
[1] 1
```

1.3 Saving Input and Output

When analyzing data or programming in R, a user will never get into serious trouble provided he or she follows two basic rules:

1. Always leave the original datafiles intact. Any revised version of data should be written into a *new* file. If you are working with a particularly large and unwieldy dataset, then write a short program that winnows-down to what you need, save the cleaned file separately, and then write code that works with the new file.
2. Write all input code in a script that is saved. Users should usually avoid writing code directly into the console. This includes code for cleaning, recoding, and reshaping data as well as for conducting analysis or developing new programs.

If these two rules are followed, then the user can always recover his or her work up to the point of some error or omission. So even if, in data management, some essential information is dropped or lost, or even if a journal reviewer names a predictor that a model should add, the user can always retrace his or her steps. By calling the original dataset with the saved program, the user can make *minor* tweaks to the code to incorporate a new feature of analysis or recover some lost information. By contrast, if the original data are overwritten or the input code is not saved, then the user likely will have to restart the whole project from the beginning, which is a waste of time.

A *script file* in R is simply plain text, usually saved with the suffix `.R`. To create a new script file in R, simply choose *File→New Document* in the drop down menu to open the document. Alternatively, the console window shown in Fig. 1.2 shows an icon that looks like a blank page at the top of the screen (second icon from the right). Clicking on this will also create a new R script file. Once open, the normal *Save* and *Save As* commands from the *File* menu apply. To open an existing R script, choose *File→Open Document* in the drop down menu, or click the icon at the top of the screen that looks like a page with writing on it (third icon from the right in Fig. 1.2). When working with a script file, any code within the file can be executed in the console by simply highlighting the code of interest, and typing the keyboard shortcut `Ctrl+R` in Windows or `Cmd+Return` in Mac. Besides the default script file editor, more sophisticated text editors such as Emacs and RWinEdt also are available.

The product of any R session is saved in the *working directory*. The working directory is the default file path for all files the user wants to read in or write out to. The command `getwd` (meaning **get** **working** **directory**) will print R's current working directory, while `setwd` (**set** **working** **directory**) allows you to change the working directory as desired. Within a Windows machine the syntax for checking, and then setting, the working directory would look like this:

```
getwd()
setwd("C:/temp/")
```

This now writes any output files, be they data sets, figures, or printed output to the folder `temp` in the `C:` drive. Observe that **R** expects forward slashes to designate subdirectories, which contrasts from Windows's typical use of backslashes. Hence, specifying `C:/temp/` as the working directory points to `C:\temp\` in normal Windows syntax. Meanwhile for Mac or Unix, setting a working directory would be similar, and the path directory is printed exactly as these operating systems designate them with forward slashes:

```
setwd("/Volumes/flashdisk/temp")
```

Note that `setwd` can be called multiple times in a session, as needed. Also, specifying the full path for any file overrides the working directory.

To *save output* from your session in **R**, try the `sink` command. As a general computing term, a **sink** is an output point for a program where data or results are written out. In **R**, this term accordingly refers to a file that records all of our printed output. To save your session's output to the file `Rintro.txt` within the working directory type:

```
sink("Rintro.txt")
```

Alternatively, if we wanted to override the working directory, in Windows for instance, we could have instead typed:

```
sink("C:/myproject/code/Rintro.txt")
```

Now that we have created an output file, any output that normally would print to the console will instead print to the file `Rintro.txt`. (For this reason, in a first run of new code, it is usually advisable to allow output to print to the screen and then rerun the code later to print to a file.) The `print` command is useful for creating output that can be easily followed. For instance, the command:

```
print("The mean of variable x is...")
```

will print the following in the file `Rintro.txt`:

```
[1] "The mean of variable x is..."
```

Another useful printing command is the `cat` command (short for **catenate**, to connect things together), which lets you mix objects in **R** with text. As a preview of simulation tools described in Chap. 11, let us create a variable named `x` by means of simulation:

```
x <- rnorm(1000)
```

By way of explanation: this syntax draws randomly 1000 times from a standard normal distribution and assigns the values to the vector `x`. Observe the arrow (`<-`), formed with a *less than* sign and a *hyphen*, which is **R**'s assignment operator. Any time we assign something with the arrow (`<-`) the name on the left (`x` in this case) allows us to recall the result of the operation on the right (`rnorm(1000)`)

in this case).³ Now we can print the mean of these 1000 draws (which should be close to 0 in this case) to our output file as follows:

```
cat("The mean of variable x is...", mean(x), "\n")
```

With this syntax, objects from R can be embedded into the statement you print. The character `\n` puts in a carriage return. You also can print any statistical output using the either `print` or `cat` commands. Remember, your output does not go to the log file unless you use one of the print commands. Another option is to simply copy and paste results from the R console window into Word or a text editor. To turn off the sink command, simply type:

```
sink()
```

1.4 Work Session Management

A key feature of R is that it is an *object-oriented* programming language. Variables, data frames, models, and outputs are all stored in memory as *objects*, or identified (and named) locations in memory with defined features. R stores in working memory any object you create using the name you define whenever you load data into memory or estimate a model. To list the objects you have created in a session use either of the following commands:

```
objects()
ls()
```

To remove all the objects in R type:

```
rm(list=ls(all=TRUE))
```

As a rule, it is a good idea to use the `rm` command at the start of any new program. If the previous user saved his or her workspace, then they may have used objects sharing the same name as yours, which can create confusion.

To quit R either close the console window or type:

```
q()
```

At this point, R will ask if you wish to save the workspace image. Generally, it is advisable not to do this, as starting with a clean slate in each session is more likely to prevent programming errors or confusion on the versions of objects loaded in memory.

Finally, in many R sessions, we will need to load *packages*, or batches of code and data offering additional functionality not written in R's base code. Throughout this book we will load several packages, particularly in Chap. 8, where

³The arrow (`<-`) is the traditional assignment operator, though a single equals sign (`=`) also can serve for assignments.

our focus will be on example packages written by prominent Political Scientists to implement cutting-edge methods. The necessary commands to load packages are `install.packages`, a command that automatically downloads and installs a package on a user's copy of R, and `library`, a command that loads the package in a given session. Suppose we wanted to install the package `MCMCpack`. This package provides tools for Bayesian modeling that we will use in Chap. 8. The form of the syntax for these commands is:

```
install.packages("MCMCpack")  
library(MCMCpack)
```

Package installation is case and spelling sensitive. R will likely prompt you at this point to choose one of the CRAN mirrors from which to download this package: For faster downloading, users typically choose the mirror that is most geographically proximate. The `install.packages` command only needs to be run once per R installation for a particular package to be available on a machine. The `library` command needs to be run for every session that a user wishes to use the package. Hence, in the next session that we want to use `MCMCpack`, we need only type: `library(MCMCpack)`.

1.5 Resources

Given the wide array of base functions that are available in R, much less the even wider array of functionality created by R packages, a book such as this cannot possibly address everything R is capable of doing. This book should serve as a resource introducing how a researcher can use R as a basic statistics program and offer some general pointers about the usage of packages and programming features. As questions emerge about topics not covered in this space, there are several other resources that may be of use:

- Within R, the *Help* pull down menu (also available by typing `help.start()` in the console) offers several manuals of use, including an “Introduction to R” and “Writing R Extensions.” This also opens an HTML-based search engine of the help files.
- UCLA’s Institute for Digital Research and Education offers several nice tutorials (<http://www.ats.ucla.edu/stat/r/>). The CRAN website also includes a variety of online manuals (<http://www.cran.r-project.org/other-docs.html>).
- Some nice interactive tutorials include `swirl`, which is a package you install in your own copy of R (more information: <http://www.swirlstats.com/>), and Try R, which is completed online (<http://tryr.codeschool.com/>).
- Within the R console, the commands `?`, `help()`, and `help.search()` all serve to find documentation. For instance, `?lm` would find the documentation for the linear model command. Alternatively, `help.search("linear model")` would search the documentation for a phrase.

- To search the internet for information, Rseek (<http://www.rseek.org/>, powered by Google) is a worthwhile search engine that searches only over websites focused on R.
- Finally, Twitter users reference R through the hashtag #rstats.

At this point, users should now have R installed on their machine, hold a basic sense of how commands are entered and output is generated, and recognize where to find the vast resources available for R users. In the next six chapters, we will see how R can be used to fill the role of a statistical analysis or econometrics software program.

1.6 Practice Problems

Each chapter will end with a few practice problems. If you have tested all of the code from the in-chapter examples, you should be able to complete these on your own. If you have not done so already, go ahead and install R on your machine for free and try the in-chapter code. Then try the following questions.

1. Compute the following in R:

- (a) -7×2^3
- (b) $\frac{8}{8^2+1}$
- (c) $\cos \pi$
- (d) $\sqrt{81}$
- (e) $\ln e^4$

- 2. What does the command `cor` do? Find documentation about it and describe what the function does.
- 3. What does the command `runif` do? Find documentation about it and describe what the function does.
- 4. Create a vector named `x` that consists of 1000 draws from a standard normal distribution, using code just like you see in Sect. 1.3. Create a second vector named `y` in the same way. Compute the correlation coefficient between the two vectors. What result do you get, and why do you get this result?
- 5. Get a feel for how to decide when add-on packages might be useful for you. Log in to <http://www.rseek.org> and look up what the `stringr` package does. What kinds of functionality does this package give you? When might you want to use it?