2

Installing and Updating R

When you purchase SAS, WPS or SPSS, they sell you a "binary" version. That is one that the company has compiled for you from the "source code" version they wrote using languages such as C, FORTRAN, or Java. You usually install everything you purchased at once and do not give it a second thought. Instead, R is modular. The main installation provides Base R and a recommended set of add-on modules called packages. You can install other packages later when you need them. With thousands to choose from, few people need them all.

To download R itself, go to the Comprehensive R Archive Network (CRAN) at http://cran.r-project.org/. Choose your operating system under the web page heading, *Download and Install R*. The binary versions install quickly and easily. Binary versions exist for many operating systems including Windows, Mac OS X, and popular versions of Linux such as Ubuntu, RedHat, Suse, and others that use either the RPM or APT installers.

Since R is an Open Source project, there are also source code versions of R for experienced programmers who prefer to compile their own copy. Using that version, you can modify R in any way you like. Although R's developers write many of its analytic procedures (or at least parts of them) using the R language, they use other languages such as C and FORTRAN to write R's most fundamental functions.

Each version of R installs into its own directory (folder), so there is no problem having multiple versions installed on your computer. You can then install your favorite add-on packages for the new release.

2.1 Installing Add-on Packages

While the main installation of R contains many useful functions, many additional packages, written by R users, are available on the Internet. The main site for additional packages is at the CRAN web site under *Packages*. The section labeled *Task Views* organizes packages by task, such as Bayesian, Cluster Analysis, Distribution, Econometrics, and so on. While CRAN is a good place

R.A. Muenchen, *R for SAS and SPSS Users*, Statistics and Computing,
DOI 10.1007/978-1-4614-0685-3_2, © Springer Science+Business Media, LLC 2011

to read about and choose packages to install, you usually do not need to download them from there yourself. As you will see, R automates the download and installation process. A comparison of SAS and SPSS add-ons to R packages is presented at this book's web site, http://www.r4stats.com. Another useful site helps you to find useful packages and write reviews of packages you like: Crantastic at http://crantastic.org/.

Before installing packages, your computer account should have administrative privileges and you must start R in a manner that allows administrative control. If you do not have administrative privileges on your computer, you can install packages to a directory to which you have write access. For instructions, see the FAQ (*F*requently *A*sked *Q*uestions) at http://www.r-project.org/.

To start R with administrative control on Windows Vista or later, right-click its menu choice and then choose *Run as administrator*. Window's User Account Control will then ask for your permission to allow R to modify your computer.

On the R version for Microsoft Windows, you can choose *Packages> Install package(s)* from the menus. It will ask you to choose a CRAN site or "mirror" that is close you:

```
CRAN mirror
  Australia
  Austria
  Belgium
  Brazil (PR)
  ...
  USA (TX 2)
  USA (WA)
```

Then it will ask which package you wish to install:

```
Packages
  abc
  abd
  abind
  AcceptanceSampling
  ...
  zipcode
  zoo
  zyp
```

Choose one of each and click OK.

If you prefer to use a function instead of the menus, you can use the `install.packages` function. For example, to download and install Frank Harrell's `Hmisc` package [32], start R and enter the command:

```
install.packages("Hmisc")
```

R will then prompt you to choose the closest mirror site and the package you need. If you are using a graphical user interface (GUI), you click on your choice, then click *OK*. If not, R will number them for you and you enter the number of the mirror.

A common error is to forget the quotes around the package name:

```
> install.packages(Hmisc)  # Quotes are missing!

Error in install.packages(Hmisc) : object 'Hmisc' not found
```

Older versions of R also required the argument `dependencies = TRUE`, which tells R to also install any packages that this package "depends" on and those that its author "suggests" as useful. That is now the default setting and so it is usually best to avoid adding that. However, a few packages still require that setting. The best known of these packages is Fox's R Commander user interface. So you would install it using:

```
install.packages("Rcmdr", dependencies = TRUE)
```

After a package is installed, you can find out how to cite it using the `citation` function. Note that you call this function with the package name in quotes:

```
> citation("Rcmdr")

To cite package 'Rcmdr' in publications use:

  John Fox <jfox@mcmaster.ca>, with
  contributions from ...(2010). Rcmdr: R
  Commander. R package version 1.6-2.
  http://CRAN.R-project.org/package=Rcmdr

A BibTeX entry for LaTeX users is

  @Manual{,
    title = {Rcmdr: R Commander},
    author = {John Fox and with contributions from ...
```

If you use simply `citation()` it will tell you how to cite R itself.

2.2 Loading an Add-on Package

Once installed, a package is on your computer's hard drive in an area called your *library*. However, it is not quite ready to use. Each time you start R, you also have to *load* the package from your library into your computer's main memory before you can use it. The reason for this additional step is

twofold. It makes efficient use of your computer's memory and it keeps different packages conflicting with each other, or with base R functions. You can see what packages are installed and ready to load with the `library` function:

```
> library()

R Packages available
Packages in library 'C:/PROGRA~1/R/R-212~1.1/library':

anchors   Statistical analysis of surveys with...
arules    Mining Association Rules and Frequent Itemsets
base      The R Base Package
...
xtable    Export tables to LaTeX or HTML
xts       Extensible Time Series
Zelig     Everyone's Statistical Software
```

If you have just installed R, this command will show you the *Base and Recommended Packages*. They are the ones that are thoroughly tested by the R Core Team. The similar `installed.packages` function lists your installed packages along with the version and location of each.

You can load a package you need with the menu selection, *Packages> Load packages*. It will show you the names of all packages that you have installed but have not yet loaded. You can then choose one from the list.

Alternatively, you can use the `library` function. Here I am loading the `Hmisc` package. Since the Linux version lacks menus, this function is the only way to load packages.

```
library("Hmisc")
```

With the `library` function, the quotes around the package name are optional and are not usually used. However, other commands that refer to package names – such as `install.packages` – require them.

Many packages load without any messages; you will just see the ">" prompt again. When trying to load a package, you may see the error message below. It means you have either mistyped the package name (remember capitalization is important) or you have not installed the package before trying to load it. In this case, Lemon and Grosjean's `prettyR` [38] package name is typed accurately, so I have not yet installed it.

```
> library("prettyR")

Error in library("prettyR") :
  there is no package called 'prettyR'
```

To see what packages you have loaded, use the `search` function.

```
> search()

 [1] ".GlobalEnv"        "package:Hmisc"
 [3] "package:stats"     "package:graphics"
 [5] "package:grDevices" "package:utils"
 [7] "package:datasets"  "package:methods"
 [9] "Autoloads"         "package:base"
```

We will discuss this function in detail in Chapter 13, "Managing Your Files and Workspace."

Since there are so many packages written by users, two packages will occasionally have functions with the same name. That can be very confusing until you realize what is happening. For example, the `Hmisc` and `prettyR` packages both have a `describe` function that does similar things. In such a case, the package you load last will *mask* the function(s) in the package you loaded earlier. For example, I loaded the `Hmisc` package first, and now I am loading the `prettyR` package (having installed it in the meantime). The following message results:

```
> library("prettyR")

Attaching package: 'prettyR'
  The following object(s) are masked from package:Hmisc :
     describe
```

Since people usually want to use the functions in the package they loaded most recently, this is rarely a problem. However, if warnings like these bother you, you can avoid them by detaching each package as soon as you are done using it by using the `detach` function. For details, see Section 13.4, "Loading Packages."

If your favorite packages do not conflict with one anther, you can have R load them each time you start R by putting the commands in a file named *".Rprofile"*. That file can automate your settings just like the *autoexec.sas* file for SAS. For details, see Appendix C.

2.3 Updating Your Installation

Keeping your add-on packages current is very easy. You simply use the `update.packages` function.

```
> update.packages()

graph :
 Version 1.15.6 installed in C:/PROGRA~1/R/R-26~1.1/library
```

```
 Version 1.16.1 available at
   http://rh-mirror.linux.iastate.edu/CRAN

Update (y/N/c)?  y
```

R will ask you if you want to update each package. That can get tedious if you have a lot of packages to install. You can avoid that starting the update process with:

```
update.packages(ask = FALSE)
```

If you enter "y," it will do it and show you the following. This message, repeated for each package, tells you what file it is getting from the mirror you requested (Iowa State) and where it placed the file.

```
trying URL 'http://rh-mirror.linux.iastate.edu
  /CRAN/bin/windows/contrib/2.6/graph_1.16.1.zip'
Content type 'application/zip' length 870777 bytes (850 Kb)
opened URL
downloaded 850 Kb
```

This next message tells you that the file was checked for errors (its sums were checked) and it says where it stored the file. As long as you see no error messages, the update is complete.

```
package 'graph' successfully unpacked and MD5 sums checked

The downloaded packages are in
        C:/Documents and Settings/muenchen/Local Settings/
   Temp/Rtmpgf4C4B/downloaded_packages
updating HTML package descriptions
```

Moving to a whole new version of R is not as easy. First, you download and install the new version just like you did the first one. Multiple versions can coexist on the same computer. You can even run them at the same time if you wanted to compare results across versions. When you install a new version of R, I recommend also installing your add-on packages again. There are ways to point your new version to the older set of packages, I find them more trouble than they are worth. You can reinstall your packages in the step-by-step fashion discussed previously. An easier way is to define a character variable like "myPackages" that contains the names of the packages you use. The following is an example that uses this approach to install most of the packages we use in this book[1].

```
myPackages <- c("car", "hexbin", "Hmisc", "ggplot2",
  "gmodels", "gplots", "reshape2", "prettyR", "xtable")
```

[1] R Commander is left out since it requires `dependencies = TRUE`.

```
install.packages(myPackages)
```

We will discuss the details of the `c` function used above later. We will also discuss how to store programs like this so you can open and execute them again in the future. While this example makes it clear what we are storing in myPackages, a shortcut to creating it is to use the `installed.packages` function:

```
myPackages <- row.names( installed.packages() )
```

You can automate the creation of myPackages (or whatever name you choose to store your package names) by placing either code example that defines it in your .Rprofile. Putting it there will ensure that myPackages is defined every time you start R. As you find new packages to install, you can add to the definition of myPackages. Then installing all of them when a new version of R comes out is easy. Of course, you do not want to place the `install.packages` function into your .Rprofile. There is no point in installing package every time you start R! For details, see Appendix C.

2.4 Uninstalling R

When you get a new version of any software package, it is good to keep the old one around for a while in case any bugs show up in the new one. Once you are confident that you will no longer need an older version of R, you can remove it.

In Microsoft Window, uninstall it in the usual way using *Start> Control Panel*, then *Programs and Features*. To uninstall R on the Macintosh, simply drag the application to the trash. Linux users should use their distribution's package manager to uninstall R.

2.5 Uninstalling a Package

Since uninstalling R itself also removes any packages in your library, it is rarely necessary to uninstall packages separately. However, it is occasionally necessary. You can uninstall a package using the `uninstall.packages` function. First, though, you must make sure it is not in use by detaching it. For example, to remove just the `Hmisc` package, use the following code:

```
detach("package:Hmisc")  # If it is loaded.

remove.packages("Hmisc")
```

2.6 Choosing Repositories

While most R packages are stored at the CRAN site, there are other repositories. If the Packages window does not list the one you need, you may need to choose another repository. The *Omegahat Project for Statistical Computing* [59] at `http://www.omegahat.org/` and *R-Forge* [61] at `http://r-forge.r-project.org/` are repositories similar to CRAN that have a variety of different packages available. There are also several repositories associated with the *BioConductor project*. As they say at their main web site, `http://www.bioconductor.org/`, "BioConductor is an open source and open development software project for the analysis and comprehension of genomic data" [23].

To choose your repositories, choose *Packages> Select repositories...* and the *Repositories* window will appear:

```
Repositories
  CRAN
  CRAN (extras)
  Omegahat
  BioC software
  BioC annotation
  BioC experiment
  BioC extra
  R-Forge
  rforge.net
```

The two CRAN repositories are already set by default. Your operating system's common mouse commands work as usual to make contiguous or noncontiguous selections. In Microsoft Window, that is Shift-click and Ctrl-click, respectively.

You can also select repositories using the `setRepositories` function:

```
> setRepositories()
```

If you are using a GUI the result will be the same. If you are instead working without a graphical user interface, R will number the repositories and prompt you to enter the number(s) of those you need.

2.7 Accessing Data in Packages

You can get a list of data sets available in each loaded package with the `data` function. A window listing the default data sets will appear:

```
> data()

R data sets
```

```
Data sets in package 'datasets':

  AirPassengers  Monthly Airline Passenger Numbers 1949-1960
  BJsales        Sales Data with Leading Indicator
  CO2            Carbon Dioxide Uptake in Grass Plants
  ...
  volcano        Topographic Information on Auckland's...
  warpbreaks     The Number of Breaks in Yarn during Weaving
  women          Average Heights and Weights for American Women
```

You can usually use these practice data sets directly. For example, to look at the top of the CO2 file (capital letters C and O, not zero!), you can use the `head` function:

```
> head(CO2)

  Plant   Type  Treatment conc uptake
1   Qn1 Quebec nonchilled   95   16.0
2   Qn1 Quebec nonchilled  175   30.4
3   Qn1 Quebec nonchilled  250   34.8
4   Qn1 Quebec nonchilled  350   37.2
5   Qn1 Quebec nonchilled  500   35.3
6   Qn1 Quebec nonchilled  675   39.2
```

The similar `tail` function shows you the bottom few observations.

Not all packages load their example data sets when you load the packages. If you see that a package includes a data set, but you cannot access it after loading the package, try loading it specifically using the `data` function. For example:

```
data(CO2)
```

If you only want a list of data sets in a particular package, you can use the `package` argument. For example, if you have installed the `car` package [21] (from Fox's *C*ompanion to *A*pplied *R*egression), you can load it from the library and see the data sets only it has using the following statements:

```
> library("car")

> data(package = "car")

Data sets in package 'car':
  AMSsurvey  American Math Society Survey Data
  Adler      Experimenter Expectations
  Angell     Moral Integration of American Cities
  Anscombe   U. S. State Public-School Expenditures
  Baumann    Methods of Teaching Reading Comprehension
```

```
  Bfox       Canadian Women's Labour-Force Participation
  Blackmoor  Exercise Histories of Eating-Disordered...
  Burt       Fraudulent Data on IQs of Twins Raised Apart
  ...
```

You could then print the top of any data set using the `head` function:

```
> head(Adler)

  instruction expectation rating
1        GOOD        HIGH     25
2        GOOD        HIGH      0
3        GOOD        HIGH    -16
4        GOOD        HIGH      5
5        GOOD        HIGH     11
6        GOOD        HIGH     -6
```

To see all of the data sets available in all the packages you have installed, even those not loaded from your library, enter the following function call:

```
data(package = .packages(all.available = TRUE))
```

http://www.springer.com/978-1-4614-0684-6

R for SAS and SPSS Users
Muenchen, R.A.
2011, XXVIII, 686 p. 118 illus., 32 illus. in color.,
Hardcover
ISBN: 978-1-4614-0684-6