CHAPTER 1

Getting Started and Getting Help

Introduction

This chapter sets the groundwork for the other chapters. It explains how to download, install, and run R.

More importantly, it also explains how to get answers to your questions. The R community provides a wealth of documentation and help. You are not alone. Here are some common sources of help:

Local, installed documentation
: When you install R on your computer, a mass of documentation is also installed. You can browse the local documentation (Recipe 1.6) and search it (Recipe 1.8). I am amazed how often I search the Web for an answer only to discover it was already available in the installed documentation.

Task views
: A *task view* describes packages that are specific to one area of statistical work, such as econometrics, medical imaging, psychometrics, or spatial statistics. Each task view is written and maintained by an expert in the field. There are 28 such task views, so there is likely to be one or more for your areas of interest. I recommend that every beginner find and read at least one task view in order to gain a sense of R's possibilities (Recipe 1.11).

Package documentation
: Most packages include useful documentation. Many also include overviews and tutorials, called *vignettes* in the R community. The documentation is kept with the packages in package repositories, such as CRAN, and it is automatically installed on your machine when you install a package.

Mailing lists
: Volunteers have generously donated many hours of time to answer beginners' questions that are posted to the R mailing lists. The lists are archived, so you can search the archives for answers to your questions (Recipe 1.12).

Question and answer (Q&A) websites
: On a Q&A site, anyone can post a question, and knowledgeable people can respond. Readers vote on the answers, so the best answers tend to emerge over time. All this information is tagged and archived for searching. These sites are a cross between a mailing list and a social network; the Stack Overflow site is a good example.

The Web
: The Web is loaded with information about R, and there are R-specific tools for searching it (Recipe 1.10). The Web is a moving target, so be on the lookout for new, improved ways to organize and search information regarding R.

1.1 Downloading and Installing R

Problem

You want to install R on your computer.

Solution

Windows and OS X users can download R from CRAN, the Comprehensive R Archive Network. Linux and Unix users can install R packages using their package management tool:

Windows

1. Open *http://www.r-project.org/* in your browser.
2. Click on “CRAN”. You’ll see a list of mirror sites, organized by country.
3. Select a site near you.
4. Click on “Windows” under “Download and Install R”.
5. Click on “base”.
6. Click on the link for downloading the latest version of R (an *.exe* file).
7. When the download completes, double-click on the *.exe* file and answer the usual questions.

OS X

1. Open *http://www.r-project.org/* in your browser.
2. Click on “CRAN”. You’ll see a list of mirror sites, organized by country.
3. Select a site near you.
4. Click on “MacOS X”.
5. Click on the *.pkg* file for the latest version of R, under “Files:”, to download it.
6. When the download completes, double-click on the *.pkg* file and answer the usual questions.

Linux or Unix

The major Linux distributions have packages for installing R. Here are some examples:

Distribution	Package name
Ubuntu or Debian	r-base
Red Hat or Fedora	R.i386
Suse	R-base

Use the system's package manager to download and install the package. Normally, you will need the root password or `sudo` privileges; otherwise, ask a system administrator to perform the installation.

Discussion

Installing R on Windows or OS X is straightforward because there are prebuilt binaries for those platforms. You need only follow the preceding instructions. The CRAN Web pages also contain links to installation-related resources, such as frequently asked questions (FAQs) and tips for special situations ("How do I install R when using Windows Vista?") that you may find useful.

Theoretically, you can install R on Linux or Unix in one of two ways: by installing a distribution package or by building it from scratch. In practice, installing a package is the preferred route. The distribution packages greatly streamline both the initial installation and subsequent updates.

On Ubuntu or Debian, use `apt-get` to download and install R. Run under `sudo` to have the necessary privileges:

```
$ sudo apt-get install r-base
```

On Red Hat or Fedora, use `yum`:

```
$ sudo yum install R.i386
```

Most platforms also have graphical package managers, which you might find more convenient.

Beyond the base packages, I recommend installing the documentation packages, too. On my Ubuntu machine, for example, I installed `r-base-html` (because I like browsing the hyperlinked documentation) as well as `r-doc-html`, which installs the important R manuals locally:

```
$ sudo apt-get install r-base-html r-doc-html
```

Some Linux repositories also include prebuilt copies of R packages available on CRAN. I don't use them because I'd rather get my software directly from CRAN itself, which usually has the freshest versions.

In rare cases, you may need to build R from scratch. You might have an obscure, unsupported version of Unix; or you might have special considerations regarding performance or configuration. The build procedure on Linux or Unix is quite standard. Download the tarball from the home page of your CRAN mirror; it's called something like *R-2.12.1.tar.gz*, except the "2.12.1" will be replaced by the latest version. Unpack the tarball, look for a file called *INSTALL*, and follow the directions.

See Also

R in a Nutshell (O'Reilly) contains more details of downloading and installing R, including instructions for building the Windows and OS X versions. Perhaps the ultimate guide is the one entitled R Installation and Administration, available on CRAN, which describes building and installing R on a variety of platforms.

This recipe is about installing the base package. See Recipe 3.9 for installing add-on packages from CRAN.

1.2 Starting R

Problem

You want to run R on your computer.

Solution

Windows
: Click on Start → All Programs → R; or double-click on the R icon on your desktop (assuming the installer created an icon for you).

OS X
: Either click on the icon in the *Applications* directory or put the R icon on the dock and click on the icon there. Alternatively, you can just type `R` on a Unix command line in a shell.

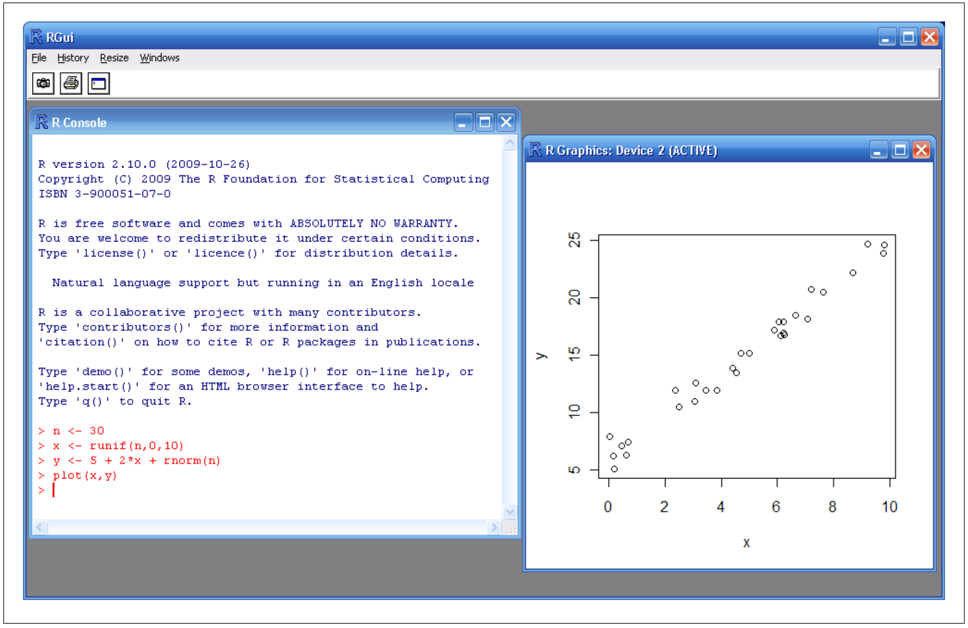
Linux or Unix
: Start the R program from the shell prompt using the `R` command (uppercase R).

Discussion

How you start R depends upon your platform.

Starting on Windows

When you start R, it opens a new window. The window includes a text pane, called the R Console, where you enter R expressions (see Figure 1-1).



```
R version 2.10.0 (2009-10-26)
Copyright (C) 2009 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

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.

> n <- 30
> x <- runif(n,0,10)
> y <- 5 + 2*x + rnorm(n)
> plot(x,y)
> |
```

Figure 1-1. R on Windows

There is an odd thing about the Windows Start menu for R. Every time you upgrade to a new version of R, the Start menu expands to contain the new version while keeping all the previously installed versions. So if you've upgraded, you may face several choices such as "R 2.8.1", "R 2.9.1", "R 2.10.1", and so forth. Pick the newest one. (You might also consider uninstalling the older versions to reduce the clutter.)

Using the Start menu is cumbersome, so I suggest starting R in one of two other ways: by creating a desktop shortcut or by double-clicking on your *.RData* file.

The installer may have created a desktop icon. If not, creating a shortcut is easy: follow the Start menu to the R program, but instead of left-clicking to run R, press and hold your mouse's right button on the program name, drag the program name to your desktop, and release the mouse button. Windows will ask if you want to Copy Here or Move Here. Select Copy Here, and the shortcut will appear on your desktop.

Another way to start R is by double-clicking on a *.RData* file in your working directory. This is the file that R creates to save your workspace. The first time you create a directory, start R and change to that directory. Save your workspace there, either by exiting or using the `save.image` function. That will create the *.RData* file. Thereafter, you can simply open the directory in Windows Explorer and then double-click on the *.RData* file to start R.

Perhaps the most baffling aspect of starting R on Windows is embodied in a simple question: When R starts, what is the working directory? The answer, of course, is that "it depends":

- If you start R from the Start menu, the working directory is normally either *C:\Documents and Settings\<username>\My Documents* (Windows XP) or *C:\Users\<username>\Documents* (Windows Vista, Windows 7). You can override this default by setting the `R_USER` environment variable to an alternative directory path.
- If you start R from a desktop shortcut, you can specify an alternative startup directory that becomes the working directory when R is started. To specify the alternative directory, right-click on the shortcut, select Properties, enter the directory path in the box labeled "Start in", and click OK.
- Starting R by double-clicking on your *.RData* file is the most straightforward solution to this little problem. R will automatically change its working directory to be the file's directory, which is usually what you want.

In any event, you can always use the `getwd` function to discover your current working directory (Recipe 3.1).

Just for the record, Windows also has a console version of R called *Rterm.exe*. You'll find it in the *bin* subdirectory of your R installation. It is much less convenient than the graphic user interface (GUI) version, and I never use it. I recommend it only for batch (noninteractive) usage such as running jobs from the Windows scheduler. In this book, I assume you are running the GUI version of R, not the console version.

Starting on OS X

Run R by clicking the R icon in the *Applications* folder. (If you use R frequently, you can drag it from the folder to the dock.) That will run the GUI version, which is somewhat more convenient than the console version. The GUI version displays your working directory, which is initially your home directory.

OS X also lets you run the console version of R by typing `R` at the shell prompt.

Starting on Linux and Unix

Start the console version of R from the Unix shell prompt simply by typing `R`, the name of the program. Be careful to type an uppercase `R`, not a lowercase `r`.

The R program has a bewildering number of command line options. Use the `--help` option to see the complete list.

See Also

See Recipe 1.4 for exiting from R, Recipe 3.1 for more about the current working directory, Recipe 3.2 for more about saving your workspace, and Recipe 3.11 for suppressing the start-up message. See Chapter 2 of *R in a Nutshell*.

1.3 Entering Commands

Problem

You've started R, and you've got a command prompt. Now what?

Solution

Simply enter expressions at the command prompt. R will evaluate them and print (display) the result. You can use command-line editing to facilitate typing.

Discussion

R prompts you with ">". To get started, just treat R like a big calculator: enter an expression, and R will evaluate the expression and print the result:

```
> 1+1
[1] 2
```

The computer adds one and one, giving two, and displays the result.

The `[1]` before the `2` might be confusing. To R, the result is a vector, even though it has only one element. R labels the value with `[1]` to signify that this is the first element of the vector...which is not surprising, since it's the *only* element of the vector.

R will prompt you for input until you type a complete expression. The expression `max(1,3,5)` is a complete expression, so R stops reading input and evaluates what it's got:

```
> max(1,3,5)
[1] 5
```

In contrast, "`max(1,3,`" is an incomplete expression, so R prompts you for more input. The prompt changes from greater-than (>) to plus (+), letting you know that R expects more:

```
> max(1,3,
+ 5)
[1] 5
```

It's easy to mistype commands, and retyping them is tedious and frustrating. So R includes command-line editing to make life easier. It defines single keystrokes that let you easily recall, correct, and reexecute your commands. My own typical command-line interaction goes like this:

1. I enter an R expression with a typo.
2. R complains about my mistake.
3. I press the up-arrow key to recall my mistaken line.
4. I use the left and right arrow keys to move the cursor back to the error.
5. I use the Delete key to delete the offending characters.

6. I type the corrected characters, which inserts them into the command line.
7. I press Enter to reexecute the corrected command.

That's just the basics. R supports the usual keystrokes for recalling and editing command lines, as listed in Table 1-1.

Table 1-1. Keystrokes for command-line editing

Labeled key	Ctrl-key combination	Effect
Up arrow	Ctrl-P	Recall previous command by moving backward through the history of commands.
Down arrow	Ctrl-N	Move forward through the history of commands.
Backspace	Ctrl-H	Delete the character to the left of cursor.
Delete (Del)	Ctrl-D	Delete the character to the right of cursor.
Home	Ctrl-A	Move cursor to the start of the line.
End	Ctrl-E	Move cursor to the end of the line.
Right arrow	Ctrl-F	Move cursor right (forward) one character.
Left arrow	Ctrl-B	Move cursor left (back) one character.
	Ctrl-K	Delete everything from the cursor position to the end of the line.
	Ctrl-U	Clear the whole darn line and start over.
Tab		Name completion (on some platforms).

On Windows and OS X, you can also use the mouse to highlight commands and then use the usual copy and paste commands to paste text into a new command line.

See Also

See Recipe 2.13. From the Windows main menu, follow Help → Console for a complete list of keystrokes useful for command-line editing.

1.4 Exiting from R

Problem

You want to exit from R.

Solution

Windows
: Select File → Exit from the main menu; or click on the red X in the upper-right corner of the window frame.

OS X
: Press CMD-q (apple-q); or click on the red X in the upper-left corner of the window frame.

Linux or Unix
: At the command prompt, press Ctrl-D.

On all platforms, you can also use the `q` function (as in *q*uit) to terminate the program.

```
> q()
```

Note the empty parentheses, which are necessary to call the function.

Discussion

Whenever you exit, R asks if you want to save your workspace. You have three choices:

- Save your workspace and exit.
- Don't save your workspace, but exit anyway.
- Cancel, returning to the command prompt rather than exiting.

If you save your workspace, then R writes it to a file called `.RData` in the current working directory. This will overwrite the previously saved workspace, if any, so don't save if you don't like the changes to your workspace (e.g., if you have accidentally erased critical data).

See Also

See Recipe 3.1 for more about the current working directory and Recipe 3.2 for more about saving your workspace. See Chapter 2 of *R in a Nutshell*.

1.5 Interrupting R

Problem

You want to interrupt a long-running computation and return to the command prompt without exiting R.

Solution

Windows or OS X
: Either press the Esc key or click on the Stop-sign icon.

Linux or Unix
: Press Ctrl-C. This will interrupt R without terminating it.

Discussion

Interrupting R can leave your variables in an indeterminate state, depending upon how far the computation had progressed. Check your workspace after interrupting.

See Also

See Recipe 1.4.

1.6 Viewing the Supplied Documentation

Problem

You want to read the documentation supplied with R.

Solution

Use the `help.start` function to see the documentation's table of contents:

```
> help.start()
```

From there, links are available to all the installed documentation.

Discussion

The base distribution of R includes a wealth of documentation—literally thousands of pages. When you install additional packages, those packages contain documentation that is also installed on your machine.

It is easy to browse this documentation via the `help.start` function, which opens a window on the top-level table of contents; see Figure 1-2.

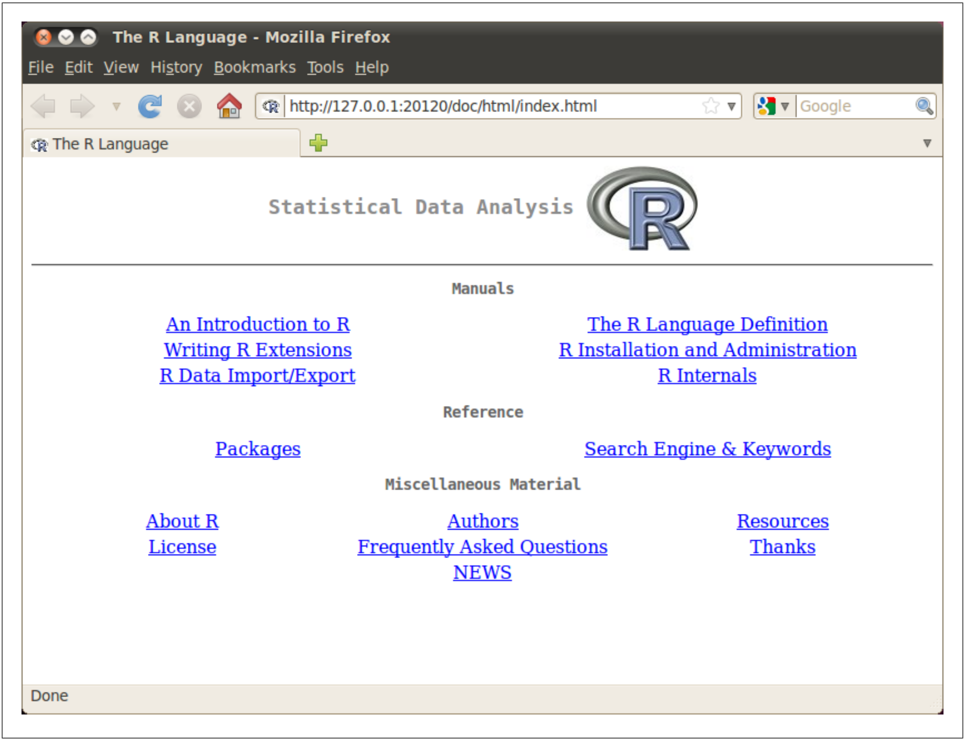
The two links in the Reference section are especially useful:

Packages
: Click here to see a list of all the installed packages, both in the base packages and the additional, installed packages. Click on a package name to see a list of its functions and datasets.

Search Engine & Keywords
: Click here to access a simple search engine, which allows you to search the documentation by keyword or phrase. There is also a list of common keywords, organized by topic; click one to see the associated pages.

See Also

The local documentation is copied from the R Project website, which may have updated documents.


The R Language - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://127.0.0.1:20120/doc/html/index.html
Google
The R Language
Done

Statistical Data Analysis

Manuals

An Introduction to R
Writing R Extensions
R Data Import/Export
The R Language Definition
R Installation and Administration
R Internals

Reference

Packages
Search Engine & Keywords

Miscellaneous Material

About R
License
Authors
Frequently Asked Questions
NEWS
Resources
Thanks

Figure 1-2. Documentation table of contents

1.7 Getting Help on a Function

Problem

You want to know more about a function that is installed on your machine.

Solution

Use `help` to display the documentation for the function:

```
> help(functionname)
```

Use `args` for a quick reminder of the function arguments:

```
> args(functionname)
```

Use `example` to see examples of using the function:

```
> example(functionname)
```

Discussion

I present many R functions in this book. Every R function has more bells and whistles than I can possibly describe. If a function catches your interest, I strongly suggest reading the help page for that function. One of its bells or whistles might be very useful to you.

Suppose you want to know more about the `mean` function. Use the `help` function like this:

```
> help(mean)
```

This will either open a window with function documentation or display the documentation on your console, depending upon your platform. A shortcut for the `help` command is to simply type `?` followed by the function name:

```
> ?mean
```

Sometimes you just want a quick reminder of the arguments to a function: What are they, and in what order do they occur? Use the `args` function:

```
> args(mean)
function (x, ...)
NULL
> args(sd)
function (x, na.rm = FALSE)
NULL
```

The first line of output from `args` is a synopsis of the function call. For `mean`, the synopsis shows one argument, `x`, which is a vector of numbers. For `sd`, the synopsis shows the same vector, `x`, and an optional argument called `na.rm`. (You can ignore the second line of output, which is often just `NULL`.)

Most documentation for functions includes examples near the end. A cool feature of R is that you can request that it execute the examples, giving you a little demonstration of the function's capabilities. The documentation for the `mean` function, for instance, contains examples, but you don't need to type them yourself. Just use the `example` function to watch them run:

```
> example(mean)

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

mean> xm <- mean(x)

mean> c(xm, mean(x, trim = 0.1))
[1] 8.75 5.50

mean> mean(USArrests, trim = 0.2)
  Murder  Assault UrbanPop     Rape
    7.42   167.60    66.20    20.16
```

The user typed `example(mean)`. Everything else was produced by R, which executed the examples from the help page and displayed the results.

See Also

See Recipe 1.8 for searching for functions and Recipe 3.5 for more about the search path.

1.8 Searching the Supplied Documentation

Problem

You want to know more about a function that is installed on your machine, but the `help` function reports that it cannot find documentation for any such function.

Alternatively, you want to search the installed documentation for a keyword.

Solution

Use `help.search` to search the R documentation on your computer:

```
> help.search("pattern")
```

A typical *`pattern`* is a function name or keyword. Notice that it must be enclosed in quotation marks.

For your convenience, you can also invoke a search by using two question marks (in which case the quotes are not required):

```
> ??pattern
```

Discussion

You may occasionally request help on a function only to be told R knows nothing about it:

```
> help(adf.test)
No documentation for 'adf.test' in specified packages and libraries:
you could try 'help.search("adf.test")'
```

This can be frustrating if you *know* the function is installed on your machine. Here the problem is that the function's package is not currently loaded, and you don't know which package contains the function. It's a kind of catch-22 (the error message indicates the package is not currently in your search path, so R cannot find the help file; see Recipe 3.5 for more details).

The solution is to search all your installed packages for the function. Just use the `help.search` function, as suggested in the error message:

```
> help.search("adf.test")
```

The search will produce a listing of all packages that contain the function:

```
Help files with alias or concept or title matching 'adf.test' using
regular expression matching:

tseries::adf.test       Augmented Dickey-Fuller Test

Type '?PKG::FOO' to inspect entry 'PKG::FOO TITLE'.
```

The following output, for example, indicates that the `tseries` package contains the `adf.test` function. You can see its documentation by explicitly telling `help` which package contains the function:

```
> help(adf.test, package="tseries")
```

Alternatively, you can insert the `tseries` package into your search list and repeat the original `help` command, which will then find the function and display the documentation.

You can broaden your search by using keywords. R will then find any installed documentation that contains the keywords. Suppose you want to find all functions that mention the Augmented Dickey–Fuller (ADF) test. You could search on a likely pattern:

```
> help.search("dickey-fuller")
```

On my machine, the result looks like this because I've installed two additional packages (`fUnitRoots` and `urca`) that implement the ADF test:

```
Help files with alias or concept or title matching 'dickey-fuller' using
fuzzy matching:

fUnitRoots::DickeyFullerPValues
                         Dickey-Fuller p Values
tseries::adf.test        Augmented Dickey-Fuller Test
urca::ur.df              Augmented-Dickey-Fuller Unit Root Test

Type '?PKG::FOO' to inspect entry 'PKG::FOO TITLE'.
```

See Also

You can also access the local search engine through the documentation browser; see Recipe 1.6 for how this is done. See Recipe 3.5 for more about the search path and Recipe 4.4 for getting help on functions.

1.9 Getting Help on a Package

Problem

You want to learn more about a package installed on your computer.

Solution

Use the `help` function and specify a package name (without a function name):

```
> help(package="packagename")
```

Discussion

Sometimes you want to know the contents of a package (the functions and datasets). This is especially true after you download and install a new package, for example. The help function can provide the contents plus other information once you specify the package name.

This call to help will display the information for the `tseries` package, a standard package in the base distribution:

```
> help(package="tseries")
```

The information begins with a description and continues with an index of functions and datasets. On my machine, the first few lines look like this:

```
                Information on package 'tseries'

Description:

Package:            tseries
Version:            0.10-22
Date:               2009-11-22
Title:              Time series analysis and computational finance
Author:             Compiled by Adrian Trapletti
                    <a.trapletti@swissonline.ch>
Maintainer:         Kurt Hornik <Kurt.Hornik@R-project.org>
Description:        Package for time series analysis and computational
                    finance
Depends:            R (>= 2.4.0), quadprog, stats, zoo
Suggests:           its
Imports:            graphics, stats, utils
License:            GPL-2
Packaged:           2009-11-22 19:03:45 UTC; hornik
Repository:         CRAN
Date/Publication:   2009-11-22 19:06:50
Built:              R 2.10.0; i386-pc-mingw32; 2009-12-01 19:32:47 UTC;
                    windows

Index:

NelPlo                  Nelson-Plosser Macroeconomic Time Series
USeconomic              U.S. Economic Variables
adf.test                Augmented Dickey-Fuller Test
arma                    Fit ARMA Models to Time Series

.
. (etc.)
.
```

Some packages also include vignettes, which are additional documents such as introductions, tutorials, or reference cards. They are installed on your computer as part of the package documentation when you install the package. The help page for a package includes a list of its vignettes near the bottom.

You can see a list of all vignettes on your computer by using the `vignette` function:

```
> vignette()
```

You can see the vignettes for a particular package by including its name:

```
> vignette(package="packagename")
```

Each vignette has a name, which you use to view the vignette:

```
> vignette("vignettename")
```

See Also

See Recipe 1.7 for getting help on a particular function in a package.

1.10 Searching the Web for Help

Problem

You want to search the Web for information and answers regarding R.

Solution

Inside R, use the `RSiteSearch` function to search by keyword or phrase:

```
> RSiteSearch("key phrase")
```

Inside your browser, try using these sites for searching:

http://rseek.org
: This is a Google custom search that is focused on R-specific websites.

http://stackoverflow.com/
: Stack Overflow is a searchable Q&A site oriented toward programming issues such as data structures, coding, and graphics.

http://stats.stackexchange.com/
: The Statistical Analysis area on Stack Exchange is also a searchable Q&A site, but it is oriented more toward statistics than programming.

Discussion

The `RSiteSearch` function will open a browser window and direct it to the search engine on the R Project website. There you will see an initial search that you can refine. For example, this call would start a search for "canonical correlation":

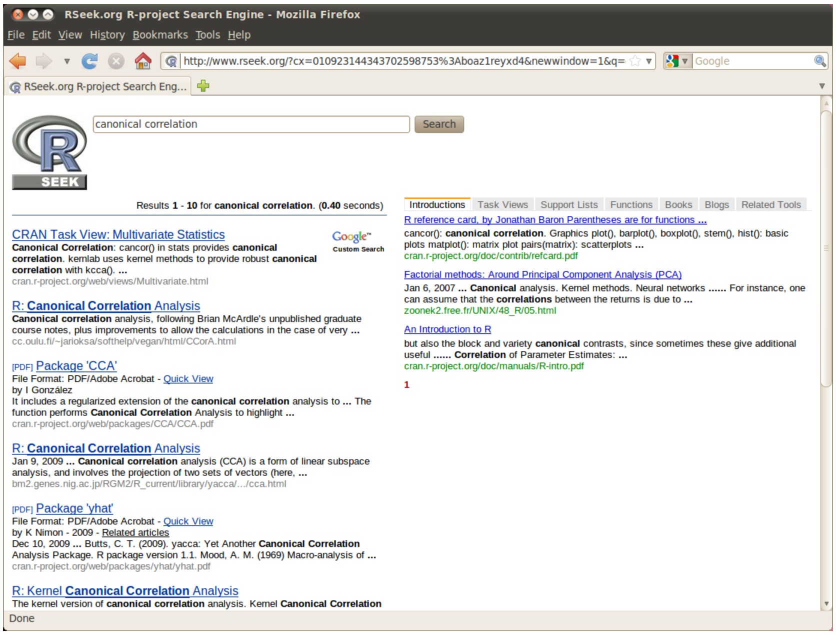
```
> RSiteSearch("canonical correlation")
```

This is quite handy for doing quick web searches without leaving R. However, the search scope is limited to R documentation and the mailing-list archives.

The *rseek.org* site provides a wider search. Its virtue is that it harnesses the power of the Google search engine while focusing on sites relevant to R. That eliminates the extraneous results of a generic Google search. The beauty of *rseek.org* is that it organizes the results in a useful way.

Figure 1-3 shows the results of visiting *rseek.org* and searching for "canonical correlation". The left side of the page shows general results for search R sites. The right side is a tabbed display that organizes the search results into several categories:

- Introductions
- Task Views
- Support Lists
- Functions
- Books
- Blogs
- Related Tools


RSeek.org R-project Search Engine - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.rseek.org/?cx=010923144343702598753%3Aboaz1reyxd4&newwindow=1&q=
Google
RSeek.org R-project Search Eng...
canonical correlation
Search
SEEK
Introductions
Task Views
Support Lists
Functions
Books
Blogs
Related Tools
Google Custom Search
Done

Results **1** - **10** for **canonical correlation**. (**0.40** seconds)

CRAN Task View: Multivariate Statistics
Canonical Correlation: cancor() in stats provides **canonical correlation**. kernlab uses kernel methods to provide robust **canonical correlation** with kcca(). ...
cran.r-project.org/web/views/Multivariate.html

R: **Canonical Correlation** Analysis
Canonical correlation analysis, following Brian McArdle's unpublished graduate course notes, plus improvements to allow the calculations in the case of very ...
cc.oulu.fi/~jarioksa/softhelp/vegan/html/CCorA.html

[PDF] Package 'CCA'
File Format: PDF/Adobe Acrobat - Quick View
by I González
It includes a regularized extension of the **canonical correlation** analysis to ... The function performs **Canonical Correlation** Analysis to highlight ...
cran.r-project.org/web/packages/CCA/CCA.pdf

R: **Canonical Correlation** Analysis
Jan 9, 2009 ... **Canonical correlation** analysis (CCA) is a form of linear subspace analysis, and involves the projection of two sets of vectors (here, ...
bm2.genes.nig.ac.jp/RGM2/R_current/library/yacca/.../cca.html

[PDF] Package 'yhat'
File Format: PDF/Adobe Acrobat - Quick View
by K Nimon - 2009 - Related articles
Dec 10, 2009 ... Butts, C. T. (2009). yacca: Yet Another **Canonical Correlation** Analysis Package. R package version 1.1. Mood, A. M. (1969) Macro-analysis of ...
cran.r-project.org/web/packages/yhat/yhat.pdf

R: Kernel **Canonical Correlation** Analysis
The kernel version of **canonical correlation** analysis. Kernel **Canonical Correlation**

R reference card, by Jonathan Baron Parentheses are for functions ...
cancor(): **canonical correlation**. Graphics plot(), barplot(), boxplot(), stem(), hist(): basic plots matplot(): matrix plot pairs(matrix): scatterplots ...
cran.r-project.org/doc/contrib/refcard.pdf

Factorial methods: Around Principal Component Analysis (PCA)
Jan 6, 2007 ... **Canonical** analysis. Kernel methods. Neural networks ...... For instance, one can assume that the **correlations** between the returns is due to ...
zoonek2.free.fr/UNIX/48_R/05.html

An Introduction to R
but also the block and variety **canonical** contrasts, since sometimes these give additional useful ...... **Correlation** of Parameter Estimates: ...
cran.r-project.org/doc/manuals/R-intro.pdf

1

Figure 1-3. Search results from rseek.org

If you click on the Introductions tab, for example, you'll find tutorial material. The Task Views tab will show any Task View that mentions your search term. Likewise, clicking on Functions will show links to relevant R functions. This is a good way to zero in on search results.

Stack Overflow is a so-called Q&A site, which means that anyone can submit a question and experienced users will supply answers—often there are multiple answers to each question. Readers vote on the answers, so good answers tend to rise to the top. This creates a rich database of Q&A dialogs, which you can search. Stack Overflow is strongly problem oriented, and the topics lean toward the programming side of R.

Stack Overflow hosts questions for many programming languages; therefore, when entering a term into their search box, prefix it with "[r]" to focus the search on questions tagged for R. For example, searching via "[r] standard error" will select only the questions tagged for R and will avoid the Python and C++ questions.

Stack Exchange (not Overflow) has a Q&A area for Statistical Analysis. The area is more focused on statistics than programming, so use this site when seeking answers that are more concerned with statistics in general and less with R in particular.

See Also

If your search reveals a useful package, use Recipe 3.9 to install it on your machine.

1.11 Finding Relevant Functions and Packages

Problem

Of the 2,000+ packages for R, you have no idea which ones would be useful to you.

Solution

- Visit the list of task views at *http://cran.r-project.org/web/views/*. Find and read the task view for your area, which will give you links to and descriptions of relevant packages. Or visit *http://rseek.org*, search by keyword, click on the Task Views tab, and select an applicable task view.
- Visit crantastic and search for packages by keyword.
- To find relevant functions, visit *http://rseek.org*, search by name or keyword, and click on the Functions tab.

Discussion

This problem is especially vexing for beginners. You think R can solve your problems, but you have no idea which packages and functions would be useful. A common question on the mailing lists is: "Is there a package to solve problem X?" That is the silent scream of someone drowning in R.

As of this writing, there are more than 2,000 packages available for free download from CRAN. Each package has a summary page with a short description and links to the package documentation. Once you've located a potentially interesting package, you would typically click on the "Reference manual" link to view the PDF documentation with full details. (The summary page also contains download links for installing the package, but you'll rarely install the package that way; see Recipe 3.9.)

Sometimes you simply have a generic interest—such as Bayesian analysis, econometrics, optimization, or graphics. CRAN contains a set of *task view* pages describing packages that may be useful. A task view is a great place to start since you get an overview of what's available. You can see the list of task view pages at *http://cran.r-project.org/web/views/* or search for them as described in the Solution.

Suppose you happen to know the name of a useful package—say, by seeing it mentioned online. A complete, alphabetical list of packages is available at *http://cran.r-project.org/web/packages/* with links to the package summary pages.

See Also

You can download and install an R package called `sos` that provides powerful other ways to search for packages; see the vignette at *http://cran.r-project.org/web/packages/sos/vignettes/sos.pdf*.

1.12 Searching the Mailing Lists

Problem

You have a question, and you want to search the archives of the mailing lists to see whether your question was answered previously.

Solution

- Open *http://rseek.org* in your browser. Search for a keyword or other search term from your question. When the search results appear, click on the "Support Lists" tab.
- You can perform a search within R itself. Use the `RSiteSearch` function to initiate a search:

  ```
  > RSiteSearch("keyphrase")
  ```

  The initial search results will appear in a browser. Under "Target", select the R-help sources, clear the other sources, and resubmit your query.

Discussion

This recipe is really just an application of Recipe 1.10. But it's an important application because you should search the mailing list archives before submitting a new question to the list. Your question has probably been answered before.

See Also

CRAN has a list of additional resources for searching the Web; see *http://cran.r-project .org/search.html*.

1.13 Submitting Questions to the Mailing Lists

Problem

You want to submit a question to the R community via the R-help mailing list.

Solution

The Mailing Lists page contains general information and instructions for using the R-help mailing list. Here is the general process:

1. Subscribe to the R-help list at the Main R Mailing List.
2. Read the Posting Guide for instructions on writing an effective submission.
3. Write your question carefully and correctly. If appropriate, include a minimal self-reproducing example so that others can reproduce your error or problem.
4. Mail your question to *r-help@r-project.org*.

Discussion

The R mailing list is a powerful resource, but please treat it as a last resort. Read the help pages, read the documentation, search the help list archives, and search the Web. It is most likely that your question has already been answered. Don't kid yourself: very few questions are unique.

After writing your question, submitting it is easy. Just mail it to *r-help@r-project.org*. You must be a list subscriber, however; otherwise your email submission may be rejected.

Your question might arise because your R code is causing an error or giving unexpected results. In that case, a critical element of your question is the *minimal self-contained example*:

Minimal
: Construct the smallest snippet of R code that displays your problem. Remove everything that is irrelevant.

Self-contained
: Include the data necessary to exactly reproduce the error. If the list readers can't reproduce it, they can't diagnose it. For complicated data structures, use the `dump` function to create an ASCII representation of your data and include it in your message.

Including an example clarifies your question and greatly increases the probability of getting a useful answer.

There are actually several mailing lists. R-help is the main list for general questions. There are also many special interest group (SIG) mailing lists dedicated to particular domains such as genetics, finance, R development, and even R jobs. You can see the full list at *https://stat.ethz.ch/mailman/listinfo*. If your question is specific to one such domain, you'll get a better answer by selecting the appropriate list. As with R-help, however, carefully search the SIG list archives before submitting your question.

See Also

An excellent essay by Eric Raymond and Rick Moen is entitled "How to Ask Questions the Smart Way". I suggest that you read it before submitting any question.

2.14 Avoiding Some Common Mistakes

Problem

You want to avoid some of the common mistakes made by beginning users—and also by experienced users, for that matter.

Discussion

Here are some easy ways to make trouble for yourself:

Forgetting the parentheses after a function invocation

You call an R function by putting parentheses after the name. For instance, this line invokes the `ls` function:

```
> ls()
[1] "x" "y" "z"
```

However, if you omit the parentheses then R does not execute the function. Instead, it shows the function definition, which is almost never what you want:

```
> ls
function (name, pos = -1, envir = as.environment(pos), all.names = FALSE,
    pattern)
{
    if (!missing(name)) {
        nameValue <- try(name)
        if (identical(class(nameValue), "try-error")) {
            name <- substitute(name)
.
. (etc.)
.
```

Forgetting to double up backslashes in Windows file paths

This function call appears to read a Windows file called *F:\research\bio\assay.csv*, but it does not:

```
> tbl <- read.csv("F:\research\bio\assay.csv")
```

Backslashes (\) inside character strings have a special meaning and therefore need to be doubled up. R will interpret this file name as *F:researchbioassay.csv*, for example, which is not what the user wanted. See Recipe 4.5 for possible solutions.

Mistyping "`<-`" as "`<` (blank) `-`"

The assignment operator is `<-`, with no space between the `<` and the `-`:

```
> x <- pi       # Set x to 3.1415926...
```

If you accidentally insert a space between `<` and `-`, the meaning changes completely:

```
> x < - pi      # Oops! We are comparing x instead of setting it!
```

This is now a comparison (<) between x and negative π (`- pi`). It does not change `x`. If you are lucky, `x` is undefined and R will complain, alerting you that something is fishy:

```
> x < - pi
Error: object "x" not found
```

If `x` is defined, R will perform the comparison and print a logical value, `TRUE` or `FALSE`. That should alert you that something is wrong: an assignment does not normally print anything:

```
> x <- 0       # Initialize x to zero
> x < - pi     # Oops!
[1] FALSE
```

Incorrectly continuing an expression across lines

R reads your typing until you finish a complete expression, no matter how many lines of input that requires. It prompts you for additional input using the + prompt until it is satisfied. This example splits an expression across two lines:

```
> total <- 1 + 2 + 3 +     # Continued on the next line
+    4 + 5
> print(total)
[1] 15
```

Problems begin when you accidentally finish the expression prematurely, which can easily happen:

```
> total <- 1 + 2 + 3       # Oops! R sees a complete expression
>    + 4 + 5               # This is a new expression; R prints its value
[1] 9
> print(total)
[1] 6
```

There are two clues that something is amiss: R prompted you with a normal prompt (>), not the continuation prompt (+); and it printed the value of 4 + 5.

This common mistake is a headache for the casual user. It is a nightmare for programmers, however, because it can introduce hard-to-find bugs into R scripts.

Using = instead of ==

Use the double-equal operator (==) for comparisons. If you accidentally use the single-equal operator (=), you will irreversibly overwrite your variable:

```
> v == 0     # Compare v against zero
> v = 0      # Assign 0 to v, overwriting previous contents
```

Writing `1:n+1` *when you mean* `1:(n+1)`

You might think that `1:n+1` is the sequence of numbers 1, 2, ..., n, n + 1. It's not. It is the sequence 1, 2, ..., n with 1 added to every element, giving 2, 3, ..., n, n + 1. This happens because R interprets `1:n+1` as `(1:n)+1`. Use parentheses to get exactly what you want:

```
> n <- 5
> 1:n+1
[1] 2 3 4 5 6
> 1:(n+1)
[1] 1 2 3 4 5 6
```

Getting bitten by the Recycling Rule

Vector arithmetic and vector comparisons work well when both vectors have the same length. However, the results can be baffling when the operands are vectors of differing lengths. Guard against this possibility by understanding and remembering the Recycling Rule, Recipe 5.3.

Installing a package but not loading it with `library()` *or* `require()`

Installing a package is the first step toward using it, but one more step is required. Use `library` or `require` to load the package into your search path. Until you do so, R will not recognize the functions or datasets in the package. See Recipe 3.6:

```
> truehist(x,n)
Error: could not find function "truehist"
> library(MASS)     # Load the MASS package into R
> truehist(x,n)
>
```

Writing `aList[i]` *when you mean* `aList[[i]]`*, or vice versa*

If the variable `lst` contains a list, it can be indexed in two ways: `lst[[n]]` is the *n*th element of the list; whereas `lst[n]` is a list whose only element is the *n*th element of `lst`. That's a big difference. See Recipe 5.7.

Using `&` *instead of* `&&`*, or vice versa; same for* `|` *and* `||`

Use `&` and `|` in logical expressions involving the logical values `TRUE` and `FALSE`. See Recipe 2.9.

Use `&&` and `||` for the flow-of-control expressions inside `if` and `while` statements.

Programmers accustomed to other programming languages may reflexively use `&&` and `||` everywhere because "they are faster." But those operators give peculiar results when applied to vectors of logical values, so avoid them unless that's really what you want.

Passing multiple arguments to a single-argument function

What do you think is the value of `mean(9,10,11)`? No, it's not 10. It's 9. The `mean` function computes the mean of the first argument. The second and third arguments are being interpreted as other, positional arguments.

Some functions, such as `mean`, take one argument. Other arguments, such as `max` and `min`, take multiple arguments and apply themselves across all arguments. Be sure you know which is which.

Thinking that `max` *behaves like* `pmax`*, or that* `min` *behaves like* `pmin`

The `max` and `min` functions have multiple arguments and return one value: the maximum or minimum of all their arguments.

The pmax and pmin functions have multiple arguments but return a vector with values taken element-wise from the arguments. See Recipe 12.9.

Misusing a function that does not understand data frames

Some functions are quite clever regarding data frames. They apply themselves to the individual columns of the data frame, computing their result for each individual column. The mean and sd functions are good examples. These functions can compute the mean or standard deviation of each column because they understand that each column is a separate variable and that mixing their data is not sensible.

Sadly, not all functions are that clever. This includes the median, max, and min functions. They will lump together every value from every column and compute their result from the lump, which might not be what you want. Be aware of which functions are savvy to data frames and which are not.

Posting a question to the mailing list before searching for the answer

Don't waste your time. Don't waste other people's time. Before you post a question to a mailing list or to Stack Overflow, do your homework and search the archives. Odds are, someone has already answered your question. If so, you'll see the answer in the discussion thread for the question. See Recipe 1.12.

See Also

See Recipes 1.12, 2.9, 5.3, and 5.7.

CHAPTER 3
Navigating the Software

Introduction

R is a big chunk of software, first and foremost. You will inevitably spend time doing what one does with any big piece of software: configuring it, customizing it, updating it, and fitting it into your computing environment. This chapter will help you perform those tasks. There is nothing here about numerics, statistics, or graphics. This is all about dealing with R as software.

3.1 Getting and Setting the Working Directory

Problem

You want to change your working directory. Or you just want to know what it is.

Solution

Command line

Use `getwd` to report the working directory, and use `setwd` to change it:

```
> getwd()
[1] "/home/paul/research"
> setwd("Bayes")
> getwd()
[1] "/home/paul/research/Bayes"
```

Windows

From the main menu, select File → Change dir... .

OS X

From the main menu, select Misc → Change Working Directory.

For both Windows and OS X, the menu selection opens the current working directory in a file browser. From there, you can navigate to a new working directory if desired.

Discussion

Your *working directory* is important because it is the default location for all file input and output—including reading and writing data files, opening and saving script files, and saving your workspace image. When you open a file and do not specify an absolute path, R will assume that the file is in your working directory.

The initial working directory depends upon how you started R. See Recipe 1.2.

See Also

See Recipe 4.5 for dealing with filenames in Windows.

3.2 Saving Your Workspace

Problem

You want to save your workspace without exiting from R.

Solution

Call the `save.image` function:

```
> save.image()
```

Discussion

Your workspace holds your R variables and functions, and it is created when R starts. The workspace is held in your computer's main memory and lasts until you exit from R, at which time you can save it.

However, you may want to save your workspace without exiting R. You might go to lunch, for example, and want to protect your work against an unexpected power outage or machine crash. Use the `save.image` function.

The workspace is written to a file called *.RData* in the working directory. When R starts, it looks for that file and, if found, initializes the workspace from it.

A sad fact is that the workspace does not include your open graphs: that cool graph on your screen disappears when you exit R, and there is no simple way to save and restore it. So before you exit, save the data and the R code that will re-create your graphs.

See Also

See Recipe 1.2 for how to save your workspace when exiting R and Recipe 3.1 for setting the working directory.

3.3 Viewing Your Command History

Problem

You want to see your recent sequence of commands.

Solution

Scroll backward by pressing the up arrow or Ctrl-P. Or use the `history` function to view your most recent input:

```
> history()
```

Discussion

The `history` function will display your most recent commands. By default it shows the most recent 25 lines, but you can request more:

```
> history(100)          # Show 100 most recent lines of history
> history(Inf)          # Show entire saved history
```

For very recent commands, simply scroll backward through your input using the command-line editing feature: pressing either the up arrow or Ctrl-P will cause your previous typing to reappear, one line at a time.

If you've exited from R then you can still see your command history. It saves the history in a file called *.Rhistory* in the working directory, if requested. Open the file with a text editor and then scroll to the bottom; you will see your most recent typing.

3.4 Saving the Result of the Previous Command

Problem

You typed an expression into R that calculated the value, but you forgot to save the result in a variable.

Solution

A special variable called `.Last.value` saves the value of the most recently evaluated expression. Save it to a variable before you type anything else.

Discussion

It is frustrating to type a long expression or call a long-running function but then forget to save the result. Fortunately, you needn't retype the expression nor invoke the function again—the result was saved in the `.Last.value` variable:

```
> aVeryLongRunningFunction()     # Oops! Forgot to save the result!
[1] 147.6549
> x <- .Last.value               # Capture the result now
> x
[1] 147.6549
```

A word of caution: the contents of `.Last.value` are overwritten every time you type another expression, so capture the value immediately. If you don't remember until another expression has been evaluated, it's too late.

See Also

See Recipe 3.3 to recall your command history.

3.5 Displaying the Search Path

Problem

You want to see the list of packages currently loaded into R.

Solution

Use the `search` function with no arguments:

```
> search()
```

Discussion

The *search path* is a list of packages that are currently loaded into memory and available for use. Although many packages may be installed on your computer, only a few of them are actually loaded into the R interpreter at any given moment. You might be wondering which packages are loaded right now.

With no arguments, the `search` function returns the list of loaded packages. It produces an output like this:

```
> search()
[1] ".GlobalEnv"        "package:stats"     "package:graphics"
[4] "package:grDevices" "package:utils"     "package:datasets"
[7] "package:methods"   "Autoloads"         "package:base"
```

Your machine may return a different result, depending on what's installed there. The return value of `search` is a vector of strings. The first string is `".GlobalEnv"`, which refers to your workspace. Most strings have the form `"package:`*`packagename`*`"`, which indicates that the package called *`packagename`* is currently loaded into R. In this example, the loaded packages include `stats`, `graphics`, `grDevices`, `utils`, and so forth.

R uses the search path to find functions. When you type a function name, R searches the path—in the order shown—until it finds the function in a loaded package. If the function is found, R executes it. Otherwise, it prints an error message and stops. (There is actually a bit more to it: the search path can contain environments, not just packages, and the search algorithm is different when initiated by an object within a package; see the *R Language Definition* for details.)

Since your workspace (`.GlobalEnv`) is first in the list, R looks for functions in your workspace before searching any packages. If your workspace and a package both contain a function with the same name, your workspace will "mask" the function; this means that R stops searching after it finds your function and so never sees the package function. This is a blessing if you want to override the package function...and a curse if you still want access to the package function.

R also uses the search path to find R datasets (not files) via a similar procedure.

Unix users: don't confuse the R search path with the Unix search path (the `PATH` environment variable). They are conceptually similar but two distinct things. The R search path is internal to R and is used by R only to locate functions and datasets, whereas the Unix search path is used by Unix to locate executable programs.

See Also

See Recipe 3.6 for loading packages into R, Recipe 3.8 for the list of installed packages (not just loaded packages), and Recipe 5.31 for inserting data frames into the search path.

3.6 Accessing the Functions in a Package

Problem

A package installed on your computer is either a standard package or a package downloaded by you. When you try using functions in the package, however, R cannot find them.

Solution

Use either the `library` function or the `require` function to load the package into R:

```
> library(packagename)
```

Discussion

R comes with several standard packages, but not all of them are automatically loaded when you start R. Likewise, you can download and install many useful packages from CRAN, but they are not automatically loaded when you run R. The `MASS` package comes standard with R, for example, but you could get this message when using the `lda` function in that package:

```
> lda(x)
Error: could not find function "lda"
```

R is complaining that it cannot find the `lda` function among the packages currently loaded into memory.

When you use the `library` function or the `require` function, R loads the package into memory and its contents become immediately available to you:

```
> lda(f ~ x + y)
Error: could not find function "lda"
> library(MASS)                         # Load the MASS library into memory
> lda(f ~ x + y)                        # Now R can find the function
Call:
lda(f ~ x + y)

Prior probabilities of groups:
.
. (etc.)
.
```

Before calling `library`, R does not recognize the function name. Afterward, the package contents are available and calling the `lda` function works.

Notice that you needn't enclose the package name in quotes.

The `require` function is nearly identical to `library`. It has two features that are useful for writing scripts. It returns `TRUE` if the package was successfully loaded and `FALSE` otherwise. It also generates a mere warning if the load fails—unlike `library`, which generates an error.

Both functions have a key feature: they do not reload packages that are already loaded, so calling twice for the same package is harmless. This is especially nice for writing scripts. The script can load needed packages while knowing that loaded packages will not be reloaded.

The `detach` function will unload a package that is currently loaded:

```
> detach(package:MASS)
```

Observe that the package name must be qualified, as in `package:MASS`.

One reason to unload a package is that it contains a function whose name conflicts with a same-named function lower on the search list. When such a conflict occurs, we say the higher function *masks* the lower function. You no longer "see" the lower function because R stops searching when it finds the higher function. Hence unloading the higher package unmasks the lower name.

See Also

See Recipe 3.5.

3.7 Accessing Built-in Datasets

Problem

You want to use one of R's built-in datasets.

Solution

The standard datasets distributed with R are already available to you, since the `datasets` package is in your search path.

To access datasets in other packages, use the `data` function while giving the dataset name and package name:

```
> data(dsname, package="pkgname")
```

Discussion

R comes with many built-in datasets. These datasets are useful when you are learning about R, since they provide data with which to experiment.

Many datasets are kept in a package called (naturally enough) `datasets`, which is distributed with R. That package is in your search path, so you have instant access to its contents. For example, you can use the built-in dataset called `pressure`:

```
> head(pressure)
  temperature pressure
1           0   0.0002
2          20   0.0012
3          40   0.0060
4          60   0.0300
5          80   0.0900
6         100   0.2700
```

If you want to know more about `pressure`, use the `help` function to learn about it and other datasets:

```
> help(pressure)          # Bring up help page for pressure dataset
```

You can see a table of contents for `datasets` by calling the `data` function with no arguments:

```
> data()                  # Bring up a list of datasets
```

Any R package can elect to include datasets that supplement those supplied in `datasets`. The `MASS` package, for example, includes many interesting datasets. Use the `data` function to access a dataset in a specific package by using the `package` argument. `MASS` includes a dataset called `Cars93`, which you can access in this way:

```
> data(Cars93, package="MASS")
```

After this call to `data`, the `Cars93` dataset is available to you; then you can execute `summary(Cars93)`, `head(Cars93)`, and so forth.

When attaching a package to your search list (e.g., via `library(MASS)`), you don't need to call `data`. Its datasets become available automatically when you attach it.

You can see a list of available datasets in `MASS`, or any other package, by using the data function with a `package` argument and no dataset name:

```
> data(package="pkgname")
```

See Also

See Recipe 3.5 for more about the search path and Recipe 3.6 for more about packages and the `library` function.

3.8 Viewing the List of Installed Packages

Problem

You want to know what packages are installed on your machine.

Solution

Use the `library` function with no arguments for a basic list. Use `installed.packages` to see more detailed information about the packages.

Discussion

The `library` function with no arguments prints a list of installed packages. The list can be quite long. On a Linux computer, these might be the first few lines of output:

```
> library()
Packages in library '/usr/local/lib/R/site-library':

boot                    Bootstrap R (S-Plus) Functions (Canty)
CGIwithR                CGI Programming in R
class                   Functions for Classification
cluster                 Cluster Analysis Extended Rousseeuw et al.
DBI                     R Database Interface
expsmooth               Data sets for "Forecasting with exponential
                        smoothing"
.
. (etc.)
.
```

On Windows and OS X, the list is displayed in a pop-up window.

You can get more details via the `installed.packages` function, which returns a matrix of information regarding the packages on your machine. Each matrix row corresponds to one installed package. The columns contain the information such as package name, library path, and version. The information is taken from R's internal database of installed packages.

To extract useful information from this matrix, use normal indexing methods. This Windows snippet calls `installed.packages` and extracts both the `Package` and `Version` columns, letting you see what version of each package is installed:

```
> installed.packages()[,c("Package","Version")]
              Package        Version
acepack       "acepack"      "1.3-2.2"
alr3          "alr3"         "1.0.9"
base          "base"         "2.4.1"
boot          "boot"         "1.2-27"
bootstrap     "bootstrap"    "1.0-20"
calibrate     "calibrate"    "0.0"
car           "car"          "1.2-1"
chron         "chron"        "2.3-12"
class         "class"        "7.2-30"
cluster       "cluster"      "1.11.4"
.
. (etc.)
.
```

See Also

See Recipe 3.6 for loading a package into memory.

3.9 Installing Packages from CRAN

Problem

You found a package on CRAN, and now you want to install it on your computer.

Solution

Command line
: Use the `install.packages` function, putting the name of the package in quotes:

```
    > install.packages("packagename")
```

Windows
: You can also download and install via Packages → Install package(s)... from the main menu.

OS X
: You can also download and install via Packages & Data → Package Installer from the main menu.

On all platforms, you will be asked to select a CRAN mirror.

On Windows and OS X, you will also be asked to select the packages for download.

System-wide Installation on Linux or Unix

On Linux or Unix systems, root privileges are required to install packages into the system-wide libraries because those directories are not usually writable by mere mortals. For that reason, installations are usually performed by a system administrator. If the administrator is unwilling or unable to do a system-wide installation, you can still install packages in your personal library.

If you have root privileges:

1. Run `su` or `sudo` to start a root shell.
2. Start an R session in the root shell.
3. From there, execute the `install.packages` function.

If you don't have root privileges, you'll know very quickly. The `install.packages` function will stop, warning you that it cannot write into the necessary directories. Then it will ask if you want to create a personal library instead. If you do, it will create the necessary directories in your home directory and install the package there.

Discussion

Installing a package locally is the first step toward using it. The installer will prompt you for a mirror site from which it can download the package files:

```
--- Please select a CRAN mirror for use in this session ---
```

It will then display a list of CRAN mirror sites. Select one close to you.

The official CRAN server is a relatively modest machine generously hosted by the Department of Statistics and Mathematics at WU Wien, Vienna, Austria. If every R user downloaded from the official server, it would buckle under the load, so there are numerous mirror sites around the globe. You are strongly encouraged to find and use a nearby mirror.

If the new package depends upon other packages that are not already installed locally, then the R installer will automatically download and install those required packages. This is a huge benefit that frees you from the tedious task of identifying and resolving those dependencies.

There is a special consideration when installing on Linux or Unix. You can install the package either in the system-wide library or in your personal library. Packages in the system-wide library are available to everyone; packages in your personal library are (normally) used only by you. So a popular, well-tested package would likely go in the system-wide library whereas an obscure or untested package would go into your personal library.

By default, `install.packages` assumes you are performing a system-wide install. To install into your personal library, first create a directory for the library—for example, *~/lib/R*: