2

The R Environment

This chapter provides an introduction to the R environment, including an overview of the environment, how to obtain and install R, and how to work with packages.

About R

R is three things: a project, a language, and a software environment. As a project, R is part of the GNU free software project (www.gnu.org), an international effort to share software on a free basis, without license restrictions. Therefore, R does not cost the user anything to use. The development and licensing of R are done under the philosophy that software should be free and not proprietary. This is good for the user, although there are some disadvantages. Mainly, that "R is free software and comes with ABSOLUTELY NO WARRANTY." This statement comes up on the screen every time you start R. There is no quality control team of a software company regulating R as a product.

The R project is largely an academic endeavor, and most of the contributors are statisticians. The R project started in 1995 by a group of statisticians at University of Auckland and has continued to grow ever since. Because statistics is a cross-disciplinary science, the use of R has appealed to academic researchers in various fields of applied statistics. There are a lot of niches in terms of R users, including: environmental statistics, econometrics, medical and public health applications, and bioinformatics, among others. This book is mainly concerned with the base R environment, basic statistical applications, and the growing number of R packages that are contributed by people in biomedical research.

The URL for the R project is http://www.r-project.org/. Rather than repeat its contents here, we encourage the reader to go ahead and spend some time reading the contents of this site to get familiar with the R project.

As a language R is a dialect of the S language, an object-oriented statistical programming language developed in the late 1980's by AT&T's Bell labs. The next chapter briefly discusses this language and introduces how to work with data objects using the S language.

The remainder of this chapter is concerned with working with R as a data analysis environment. R is an interactive software application designed specifically to perform calculations (a giant calculator of sorts), manipulate data (including importing data from other sources, discussed in Chapter 3), and produce graphical displays of data and results. Although it is a command line environment, it is not exclusively designed for programmers. It is not at all difficulty to use, but it does take a little getting used to, and this and the three subsequent chapters are geared mainly toward getting the user acquainted with working in R.

Obtaining and Installing R

The first thing to do in order to use R is to get a copy of it. This can be done on the Comprehensive R Archive Network, or CRAN, site, illustrated in Figure 2-1.

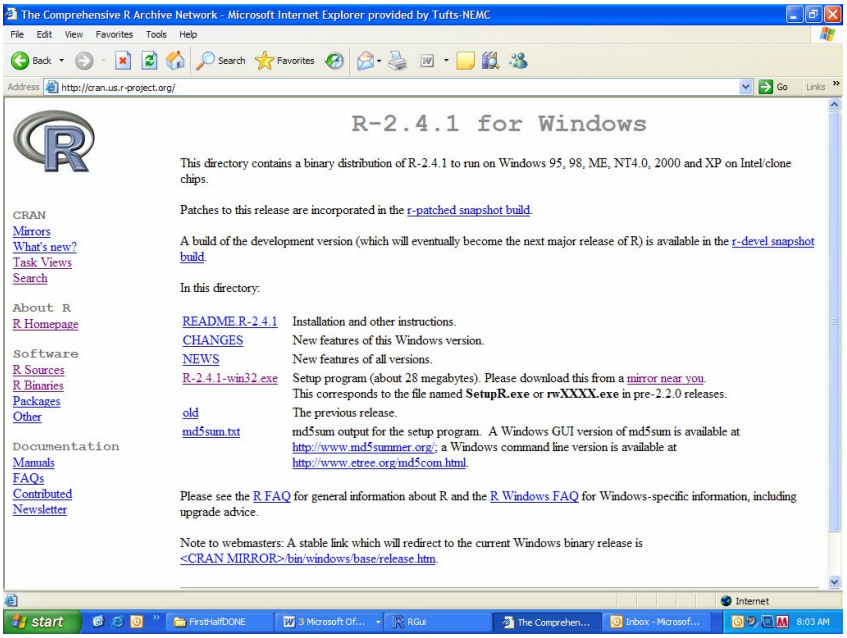


Figure 2-1

The URL for this site is *www.cran.r-project.org*. This site will be referred to many times (and links to the *www.r-project.org* site directly through the R homepage link on the left menu screen) and the user is advised to make a note of these URLs. The archive site is where you can download R and related packages, and the project site is source of information and links that provide help (including links to user groups).

On the top of the right side of the page shown in Figure 2-1 is a section entitled "Precompiled Binary Distributions", this means R versions you can download which are already compiled into a program package. For the technologically savvy you can also download R in a non-compiled version and compile it yourself (something we will not discuss here) by downloading source code.

In this sections are links to download R for various operating systems, if you click on the Windows link for example; you get the screen depicted in Figure 2-2.

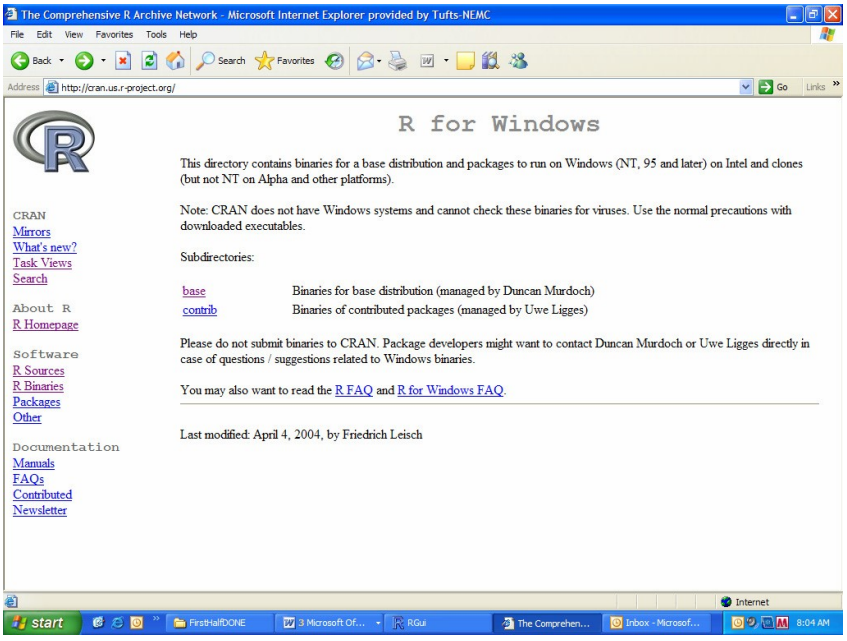


Figure 2-2

If you click on "base" (for base package, something discussed in the Packages section later in this chapter) you get the screen in Figure 2-3. The current version of R is available for download as the file with filename ending in *.exe (executable file, otherwise known as a program). R is constantly being updated and new versions are constantly released, although prior versions remain available for download.

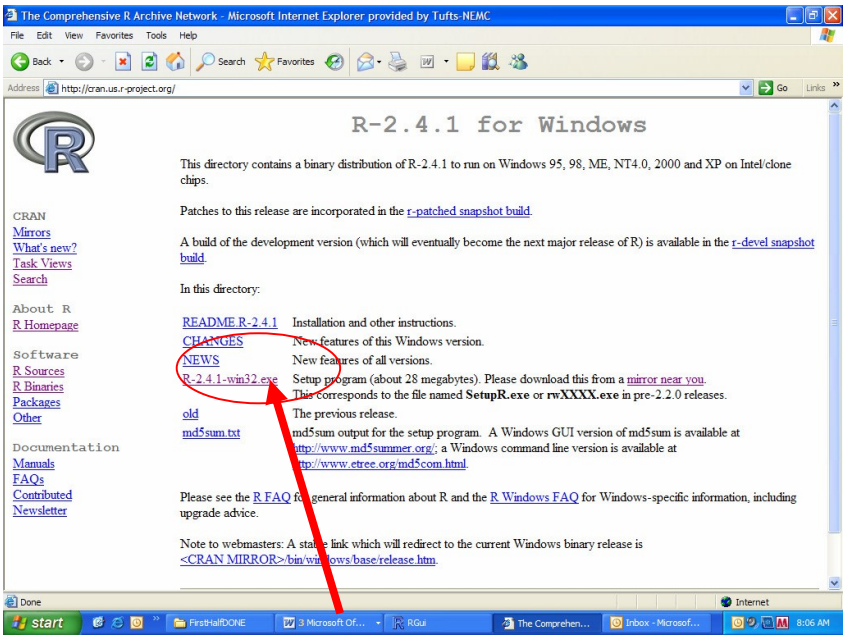


Figure 2-3

After downloading, the program needs to be installed. Installation is initiated by clicking on the "*.exe" icon (* is the filename of the current version) created and following the series of instructions presented in dialog boxes, which include accepting the user license and whether you want documentation installed. After installation the R program will be accessible from the Windows Start-Programs menu system, as well as an installed program in Program Files.

Installation for Mac and Linux systems follows similar steps. Although this book uses Windows in examples, the operating system used should not make a difference when using R and all examples should work under other operating systems.

Exploring the Environment

When you start up R the screen will look like Figure 2-4. The environment is actually quite plain and simple. There is a main application window and within it a console window. The main application contains a menu bar with six menus and toolbar with eight icons for basic tasks.

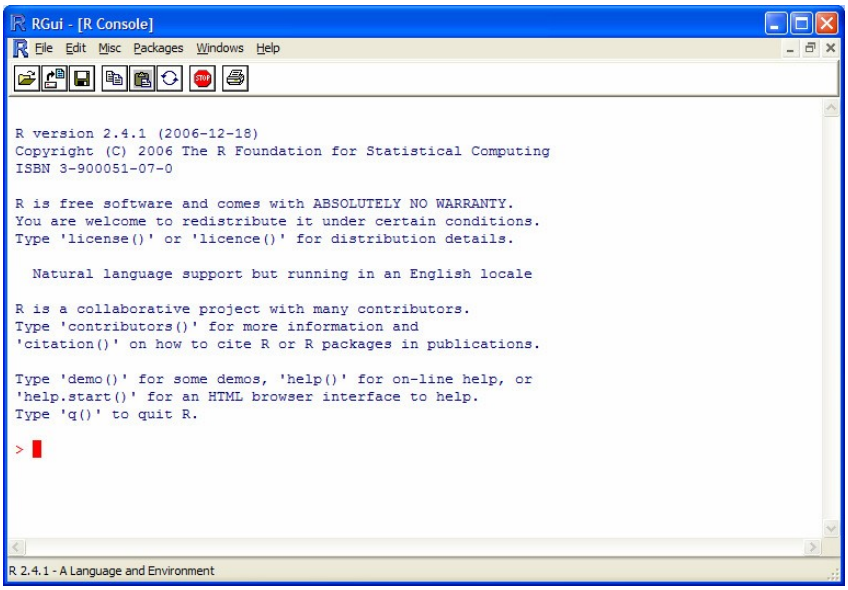


Figure 2-4

Let's explore some of the features of the R environment.

The Command Line

The command line is where you interact with R. This is designated in red and has a ">" symbol. At the command line you type in code telling R what to do. You can use R as calculator to perform basic mathematical operations. Try typing some basic arithmetic tasks at the command line. Hit enter and R will compute the requested result:

```
> 2+9
[1] 11
> 7*8
[1] 56
```

More than one line of code can be entered at the command line. Subsequent lines after the first line are designated by a "+" symbol. For example if you use an opening parenthesis and then hit enter you will get a "+" symbol and can continue writing code on the next line:

```
> 2*(
+ 4+6)
[1] 20
```

If you enter something incorrect, or that R does not understand, you will get an error message rather than a result:

```
> what
Error: Object "what" not found
```

The Menu Bar

The menu bar in R is very similar to that in most Windows based programs. It contains six pull down menus, which are briefly described below. Much of the functionality provided by the menus is redundant with those available using standard windows commands (CTRL+C to copy, for example) and with commands you can enter at the command line. Nevertheless, it is handy to have the menu system for quick access to functionality.

File

The file menu contains options for opening, saving, and printing R documents, as well as the option for exiting the program (which can also be done using the close button in the upper right hand corner of the main program window). The options that begin with "load" ("Load Workspace and "Load History") are options to open previously saved work. The next chapter discusses the different save options available in some detail as well as what a workspace and a history are in terms of R files. The option to print is standard and will print the information selected.

Edit

The edit menu contains the standard functionality of cut, copy and paste, and select all. In addition there is an option to "Clear console" which creates a blank workspace with only a command prompt (although objects are still in working memory), which can essentially clean a messy desk. The "Data editor" option allows you to access the data editor, a spreadsheet like interface for manually editing data discussed in depth in the next chapter. The last option on the edit menu is "GUI preferences" which pops up the Rgui configuration editor, allowing you to set options controlling the GUI, such as font size and background colors.

Misc

The Misc menu contains some functionality not categorized elsewhere. The most notable feature of this menu is the first option c which can also be accessed with the ESC key on your keyboard. This is your panic button should you have this misfortune of coding R to do something where it gets stuck, such as programming it in a loop which has no end or encountering some other unforeseeable snag. Selecting this option (or ESC) should get the situation under control and return the console to a new command line. Always try this before doing something more drastic as it will often work.

The other functionality provided by Misc is listing and removing objects. We will discuss working with objects in the next chapter.

Packages

The packages menu is very important, as it is the easiest way to load and install packages to the R system. Therefore the entire section following this is devoted to demonstrating how to use this menu.

Windows

The Windows menu provides options for cascading, and tiling windows. If there is more than one window open (for example, the console and a help window) you can use the open Windows list on the bottom of this menu to access the different open windows.

Help

The Help menu directs you to various sources of help and warrants some exploration. The first option, called "Console" pops up a dialog box listing a cheat sheet of "Information" listing various shortcut keystrokes to perform tasks for scrolling and editing in the main console window.

The next two options provide the FAQ (Frequently Asked Questions) HTML documents for R and R for the operating system you are using. These should work whether or not you are connected to the Internet since they are part of the program installation. The FAQ documents provide answers to technical questions and are worth browsing through.

The next section on the help menu contains the options "R language (standard)", "R language (HTML)", and "Manuals". "R language (standard) pops up the help dialog box in Figure 2-5. This will popup the help screen for the specified term, provided you enter a correct term (which can be hard if you don't know ahead of time what you're looking for). This can also be accomplished using the help () command, as we will see in the next chapter.

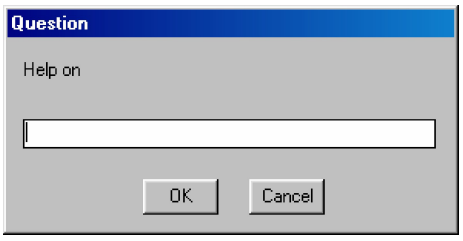


Figure 2-5

The menu option "R language (HTML)" will produce some HTML based documents containing information and links to more documentation. This should be available off-line as part of the R installation. The next option "Manuals" provides a secondary menu with several pdf files of R documents.

The remaining options on the Help menu are “Apropos” and “About”. “Apropos” pops up a dialog box similar to the help box depicted in Figure 2-5 but that you only need to enter a partial search term to search R documents. “About” pops up a little dialog box about R and the version you are using.

One of the most difficult tasks in R is finding documentation to help you. R is actually very extensively documented and only a fraction of this documentation is available directly using the help menu. However, much of the documentation is technical rather than tutorial, and geared more toward the programmer and developer rather than the applied user. More about getting help is discussed in the next chapter.

The Toolbar

Below the menu bar is the toolbar, depicted in Figure 2-5. This provides quick access icons to the main features of the menu bar. If you scroll over the icons with your mouse slowly you will get rollover messages about the feature of each icon. The stop icon can be useful as a panic button providing the same functionality as the Misc menu’s “Stop current computation” option.

Figure 2-5

Packages

The basic R installation contains the package base and several other packages considered essential enough to include in the main software installation. Exact packages included may vary with different versions of R. Installing and loading contributed packages adds additional specialized functionality. R is essentially a modular environment and you install and load the modules (packages) you need. You only need to install the packages once to you system, as they are saved locally, ready to be loaded whenever you want to use them. However

The easiest way to install and load packages is to use the Packages menu, although there are equivalent commands to use as well if you prefer the command line approach.

Installing Packages

In order to use an R package, it must be installed on your system. That is you must have a local copy of the package. Most packages are available from the CRAN site as contributed packages, and can be directly downloaded in R. In

order to do this, select “Install package from CRAN” from the Packages menu. You must be connected to tine Internet to use this option, and when you do so the connection will return a list of packages in a dialog box like that in Figure 2-6 listing available packages:

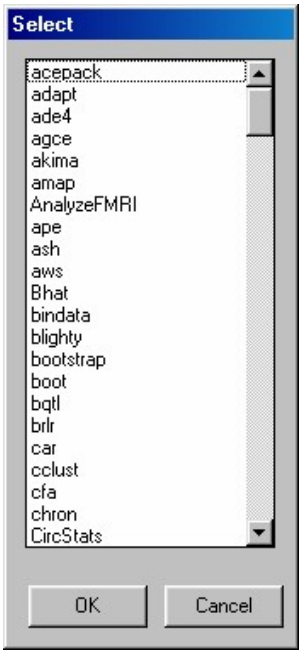


Figure 2-6

Select the package of interest and OK, and R will automatically download the package to your computer and put it in the appropriate directory to be loaded on command.

Some packages are not available directly from the CRAN site. For these packages download them to an appropriate folder on your computer from their source site. To install them select the “Install package from local zip file” option on the packages menu and R will put them in the appropriate directory.

Loading Packages

Whenever you want to use an R package you must not only have installed locally you must also load the package during the session you are using it. This makes R more efficient and uses less overhead than if all installed packages are loaded every time you use R, but makes the use do a little more work.

To load an installed package, select the “Load package” option from the packages menu. This produces another dialog box very similar to Figure 2-7, only this time the list of packages includes only those packages which are

installed locally. Select the package to load and you should be all set to use the features of that package during the current work session. Load packages become unloaded when you quit R and are NOT saved when you save your workspace or history (discussed in the next chapter). Even if you open previous work in R you will still need to reload packages you are using the features of.

R vs. S-Plus

The other major implementation of the S language, which is a popular statistical application, is a commercial package called S-Plus. This is a full commercial product, and is supported by the company that makes it, called Insightful (*www.insightful.com*). There is a demo version of S-Plus available for evaluation purposes.

R and S-Plus are almost identical in their implementation of the S language. Most code written with R can be used with S-Plus and vice versa. Where R and S-Plus differ is in the environment. S-Plus is a GUI based application environment that has many features that allow for data analysis to be more menu and pop-up dialog box assisted, requiring less coding by the user. On the other hand, the S-Plus environment has a heavier overhead, and many times code will run more efficiently in R as a consequence.

R and Other Technologies

Although not a topic that will be covered in this book, it is of interest to note that R is not an isolated technology, and a significant part of the R project involves implementing methods of using R in conjunction with other technologies. There are many packages available that contain functionality to R users in conjunction with other technologies available from the CRAN site. For example, the package ROracle provides functionality to interface R with Oracle databases, and package XML contains tools for parsing XML and related files. Interested users should explore these options on the R websites.

3

Basics of Working with R

This chapter introduces the foundational skills you need to work in R. These include the ability to create and work with data objects, controlling the workspace, importing and saving files, and how and where to get additional help.

Using the S Language

R is based on a programming language known as S. R is both an implementation of the S language that can be considered a language on its own, and a software system. There are some differences in language that are not noticeable by the applied user and are not discussed here.

The S language (and R language, if you consider them distinct languages, which is a debatable issue) was specifically designed for statistical programming. It is considered a high level object-oriented statistical programming language, but is not very similar to object-oriented languages such as C++ and Java. There is no need to know anything about object-oriented programming, other than the general idea of working with objects, in order to be an effective applied user of R.

The abstract concepts used in object-oriented languages can be confusing. However for our purposes, the concept of an object is very easy. Everything is an object in R and using R is all about creating and manipulating objects. We are concerned with two types of objects: function objects and data objects. Data objects are variable-named objects that you create to hold data in specified forms, which are described in detail in this chapter. Function objects are the objects that perform tasks on data objects, and are obtained as part of the R base system, part of an imported package, or can be written from scratch. We immediately start working with function objects in this chapter, but the next chapter covers them in more depth, including some basics of writing your own functions. Function objects perform tasks that manipulate data objects, usually some type of calculation, graphical presentation, or other data analysis. In essence, R is all about creating data objects and manipulating these data objects using function objects.

Structuring Data With Objects

In R the way that you work with data is to enter data (either directly or indirectly by importing a file) and in doing so, you are creating a data object. The form of the data object you create depends on your data analysis needs, but R has a set of standard data objects for your use. They are: scalars, vectors, factors, matrices and arrays, lists, and data frames. The different types of data objects handle different modes of data (character, numeric, and logical T/F) and format it differently. The first part of this section briefly explains what these different types of data objects are and when to use which object. The second part of this section deals with some general tasks of working with data objects that are of general use.

All data objects generally have three properties. First they have a type (scalar, vector, etc). Second, they have values (your data) that have a data mode. Finally, they generally are assigned a variable name (that you supply, using the assignment operator). Sometimes you will work only transiently with data objects, in which case you do not need to name them. But you should always provide a descriptive variable name of a data object you are going to perform further manipulations on. With few exceptions R will allow you to name your variables using any name you like.

Types of Data Objects in R

Scalars

The simplest type of object is a scalar. A scalar is an object with one value. To create a scalar data object, simply assign a value to a variable using the assignment operator "<-". Note the equals sign is not the assignment operator in R and serves other functionality.

For example to create scalar data objects x and y:

```
> #create scalar data object x with value 5
> x<-5
> #create scalar data object y with value 2
> y<-2
```

With scalar data objects of numeric mode, R is a big calculator. You can manipulate scalar objects in R and perform all sorts of algebraic calculations.

```
> #some manipulations on scalar objects x and y
> z<-x+y
> z
[1] 7
> x-y
[1] 3
> x*y+2
[1] 12
```

Of course data can also be logical or character mode. Logical data can be entered simply as T or F (no quotes).

```
> correctLogic<-T
> correctLogic
[1] TRUE
> incorrectLogic<-"T"
> incorrectLogic
[1] "T"
```

Character data should always be enclosed with quotations (either single or double quotes will do).

```
> single<-'singleQuote'
> double<-"doubleQuote"
> single
[1] "singleQuote"
> double
[1] "doubleQuote"
#You will get an error if you enter character data with no quotes at all
> tryThis<-HAHA
Error: Object "HAHA" not found
```

The function “mode (variable name)” will tell you the mode of a variable.

```
> mode(x)
[1] "numeric"
> mode(correctLogic)
[1] "logical"
> mode(incorrectLogic)
[1] "character"
```

Vectors

Of course the power of R lies not in its ability to work with simple scalar data but in its ability to work with large datasets. Vectors are the data objects probably most used in R and in this book are used literally everywhere. A vector can be defined as a set of scalars arranged in a one-dimensional array.

Essentially a scalar is a one-dimensional vector. Data values in a vector are all the same mode, but a vector can hold data of any mode.

Vectors may be entered using the c () function (or "combine values" in a vector) and the assignment operator like this:

```
> newVector<-c(2,5,5,3,3,6,2,3,5,6,3)
> newVector
 [1] 2 5 5 3 3 6 2 3 5 6 3
```

Vectors may also be entered using the scan () function and the assignment operator. This is a good way to enter data easily as you can past in unformatted data values from other documents.

```
> scannedVector<-scan()
1: 2
2: 3
3: 1
4: 3
5: 53
```

To stop the scan simply leave an entry blank and press enter.

```
6:
Read 5 items
```

Another way to make a vector is to make it out of other vectors:

```
> v1<-c(1,2,3)
> v2<-c(4,5,6)
```

You can perform all kinds of operations on vectors, a very powerful and useful feature of R, which will be used throughout this book.

```
> z<-v1+v2
> z
[1] 5 7 9
```

Note that if you perform operations on vectors with different lengths (not recommended) then the vector with the shorter length is recycled to the length of the longer vector so that the first element of the shorter vector is appended to the end of that vector (a way of faking that it is of equal length to the longer vector) and so forth. You will get a warning message, but it does let you perform the requested operation:

```
> x1<-c(1,2,3)
> x2<-c(3,4)
> x3<-x1+x2
Warning message:
longer object length
        is not a multiple of shorter object length in: x1 + x2
> x3
[1] 4 6 6
```

You can also create a vector by joining existing vectors with the c () function:

```
> q<-c(v1,v2)
> q
```

```
[1] 1 2 3 4 5 6
```

Vectors that have entries that are all the same can easily be created using the "rep" (repeat) function:

```
> x<-rep(3,7)
> x
[1] 3 3 3 3 3 3 3
> charvec<-rep("haha",4)
> charvec
[1] "haha" "haha" "haha" "haha"
```

Factors

A factor is a special type of character vector. In most cases character data is used to describe the other data, and is not used in calculations. However, for some computations qualitative variables are used. To store character data as qualitative variables, a factor data type is used. Although most coverage in this book is quantitative, we will use qualitative or categorical variables in some chapters in this book, notably in experimental design.

You may create a factor by first creating a character vector, and then converting it to a factor type using the factor () function:

```
> settings<-c("High","Medium","Low")
> settings<-factor(settings)
```

Notice that this creates "levels" based on the factor values (these are the values of categorical variables).

```
> settings
[1] High   Medium Low
Levels: High Low Medium
```

Matrices and Arrays

Matrices are collections of data values in two dimensions. In mathematics matrices have many applications, and a good course in linear algebra is required to fully appreciate the usefulness of matrices. An array is a matrix with more than two dimensions. Formatting data as matrices and arrays provides an efficient data structure to perform calculations in a computationally fast and efficient manner.

To declare a matrix in R, use the matrix () function, which takes as arguments a data vector and specification parameters for the number of rows and columns. Let's declare a simple 2 by 2 matrix.

```
> mat<-matrix(c(2,3,1,5),nrow=2,ncol=2)
> mat
     [,1] [,2]
[1,]    2    1
[2,]    3    5
```

This book makes no assumption of knowledge of matrix mathematics, and when matrices and arrays are used in applied examples, appropriate background information will be provided. Typically the data in an array or matrix is numerical.

Specially structured matrices can also be easily created. For example, creating a 2 by 3 matrix consisting of all ones can be done as follows:

```
> onemat<-matrix(1,nrow=2,ncol=3)
> onemat
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    1    1
```

If you create a matrix with a set of numbers, for example 7 numbers, and you specify it to have a set number of columns, for example 3 columns, R will cycle through the numbers until it fills all the space specified in the matrix, giving a warning about unequal replacement lengths:

```
> matrix(c(1,2,3,4,5,6,7),ncol=3)
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    1
[3,]    3    6    2
Warning message:
Replacement length not a multiple of the elements to replace in matrix(...)
```

Lists

Lists are the "everything" data objects. A list, unlike a vector, can contain data with different modes under the same variable name and encompass other data objects. Lists are useful for organizing information. Creating a list is very simple; just use the list () function to assign to a variable the list values. Note that list values are indexed with double bracket sets such as [[1]] rather than single bracket sets used by other data objects.

```
> myList<-list(5,6,"seven", mat)
> myList
[[1]]
[1] 5

[[2]]
[1] 6

[[3]]
[1] "seven"

[[4]]
     [,1] [,2]
[1,]    2    1
[2,]    3    5
```

Data Frames

Data frames are versatile data objects you can use in R. You can think of a data frame object as a being somewhat like a spreadsheet. Each column of the data frame is a vector. Within each vector, all data elements must be of the same mode. However, different vectors can be of different modes. All vectors in a data frame must be of the same length. We will use data frames frequently in this book.

To create a data frame object, let's first create the vectors that make up the data frame (genome size data from *www.ornl.gov*).:

```
> organism<-c("Human","Mouse","Fruit Fly", "Roundworm","Yeast")
> genomeSizeBP<-c(3000000000,3000000000,135600000,97000000,12100000)
> estGeneCount<-c(30000,30000,13061,19099,6034)
```

Now, with three vectors of equal length we can join these in a data frame using the function data.frame () with the vectors we want as the arguments of this function. Note that the format here is "column name"="vector to add" and the equals (not assignment) operator is used. We are naming columns not creating new variables here. Here, the variable names are used as column names, but you could rename the columns with names other than the variable names if you like.

```
> comparativeGenomeSize<-
data.frame(organism=organism,genomeSizeBP=genomeSizeBP,
+ estGeneCount=estGeneCount)


> comparativeGenomeSize
   organism genomeSizeBP estGeneCount
1     Human    3.000e+09        30000
2     Mouse    3.000e+09        30000
3 Fruit Fly    1.356e+08        13061
4 Roundworm    9.700e+07        19099
5     Yeast    1.210e+07         6034
```

Working with Data Objects

Once you have created a data object, you will often want to perform various tasks. This section discusses some common tasks to access and modify existing data objects. Mainly our focus here is on vectors and data frames, since these will be the data objects heavily utilized in this book, but similar techniques can be applied to other data objects.

Working with Vectors

In order to be able to work with a specific element in a data object, first you need to be able to identify that specific element. With vectors this is fairly easy. Every element in a vector is assigned an index value in the order in which elements were entered. This index starts with 1, not zero. To address a specific

element in a vector, enter the name of the vector and the element of interest in brackets:

```
> y<-c(9,2,4)
> y[2]
[1] 2
```

If you are not certain exactly where your value of interest is but have an idea of the range of indexes it may be in, you can look at selected index values of your vector using as set of numbers written in the form [start: finish]

```
> z<-c(1,2,3,4,12,31,2,51,23,1,23,2341,23,512,32,312,123,21,3)
> z[3:7]
[1]  3  4 12 31  2
```

You can overwrite a value by re-assigning a new value to that place in the vector:

```
> z[3]<-7
> z[3:7]
[1]  7  4 12 31  2
```

To organize your data, function sort will sort your data from smallest to largest (additional functional parameters possible to do other ordering) and function order will tell you which elements correspond to which order in your vector.

```
> sort(z)
 [1]    1    1    2    2    3    3    4   12   21   23   23   23   31   32
51
[16]  123  312  512 2341
> order(z)
 [1]  1 10  2  7  3 19  4  5 18  9 11 13  6 15  8 17 16 14 12
```

You may want to extract only certain data values from a vector. You can extract subsets of data from vectors in two ways. One is that you can directly identify specific elements and assign them to a new variable. The second way is that you can create a logical criterion to select certain elements for extraction. Illustrating both of these:

```
> #extracting specific elements
> z3<-z[c(2,3)]
> z3
[1] 2 7
> #logical extraction, note syntax
> z100<-z[z>100]
> z100
[1] 2341  512  312  123
```

Sometimes, if you are coding a loop for example, you may need to know the exact length of your vector. This is simple in R using the length () function:

```
> length(z)
[1] 19
```

Working with Data Frames

Because a data frame is a set of vectors, you can use vector tricks mentioned above to work with specific elements of each vector within the data frame. To address a specific vector (column) in a data frame, use the "$" operator with the specified column of the data frame after the "$".

```
> x<-c(1,3,2,1)
> y<-c(2,3,4,1)
> xy<-data.frame(x,y)
> xy
  x y
1 1 2
2 3 3
3 2 4
4 1 1
> #use q to create new vector extracting x column of dataframe xy
> q<-xy$x
> q
[1] 1 3 2 1
```

To address a specific element of a data frame, address that vector with the appropriate index:

```
> xy$x[2]
[1] 3
```

Commonly you will want to add a row or column to the data frame. Functions rbind, for rows, and cbind, for columns, easily perform these tasks. Note these functions work for matrices as well.

```
> #create and bind column z to
> z<-c(2,1,4,7)
> xyz<-cbind(xy,z)
> xyz
  x y z
1 1 2 2
2 3 3 1
3 2 4 4
4 1 1 7
> #create and bind new row w
> w<-c(3,4,7)
> xyz<-rbind(xyz,w)
> xyz
  x y z
1 1 2 2
2 3 3 1
3 2 4 4
4 1 1 7
5 3 4 7
```

There are many ways to work with data in data frames; only the basics have been touched on here. The best way to learn these techniques is to use them, and many examples of the use of data objects and possible manipulations will be presented in this book in the examples presented.

Checking and Changing Types

Sometimes you may forget or not know what type of data you are dealing with, so R provides functionality for you to check this. There is a set of "is.what" functions, which provide identification of data object types and modes. For example:

```
> x<-c(1,2,3,4)
> #checking data object type
> is.vector(x)
[1] TRUE
> is.data.frame(x)
[1] FALSE

> #checking data mode
> is.character(x)
[1] FALSE
> is.numeric(x)
[1] TRUE
```

Sometimes you may want to change the data object type or mode. To do this R provides a series of "as.what" functions where you convert your existing data object into a different type or mode. Don't forget to assign the new data object to a variable (either overwriting the existing variable or creating a new one) because otherwise the data object conversion will only be transient.

To change data object types, you may want to convert a vector into a matrix:

```
> y<-as.matrix(x)
> y
     [,1]
[1,]    1
[2,]    2
[3,]    3
[4,]    4
```

You can also use the "as.what" functionality to change the mode of your data. For example, you may want to change a numerical vector to a character mode vector.

```
> z<-as.character(x)
> z
[1] "1" "2" "3" "4"
```

R is smart enough to try catching you if you try to do an illogical conversion, such as convert character data to numeric mode. It does do the conversion but the data is converted to NA values.

```
> words<-c("Hello", "Hi")
> words
[1] "Hello" "Hi"
> as.numeric(words)
[1] NA NA
Warning message:
NAs introduced by coercion
```

Missing Data

Anyone working with empirical data sooner or later deals with a data set that has missing values. R treats missing values by using a special NA value. You should encode missing data in R as NA and convert any data imports with missing data in other forms to NA as well, assuming you are not using a numerical convention (such as entering 0's).

```
> missingData<-c(1,3,1,NA,2,1)
> missingData
[1]  1  3  1 NA  2  1
```

If computations are performed on data objects with NA values the NA value is carried through to the result.

```
> missingData2<-missingData*2
> missingData2
[1]  2  6  2 NA  4  2
```

If you have a computation problem with an element of a data object and are not sure whether that is a missing value, the function is.na can be used to determine if the element in question is a NA value.

```
> is.na(missingData[1])
[1] FALSE

> is.na(missingData[4])
[1] TRUE
```

Controlling the Workspace

This section describes some basic housekeeping tasks of listing, deleting, and editing existing objects. Then there is discussion of the different ways of saving your workspace.

Listing and Deleting Objects in Memory

When working in R and using many data objects, you may lose track of the names of the objects you have already created. Two different functions ls() and objects() have redundant functionality in R to list the current objects in current workspace memory.

```
> ls()
[1] "q"  "v1" "v2" "x1" "x2" "x3" "z"
> objects()
[1] "q"  "v1" "v2" "x1" "x2" "x3" "z"
```

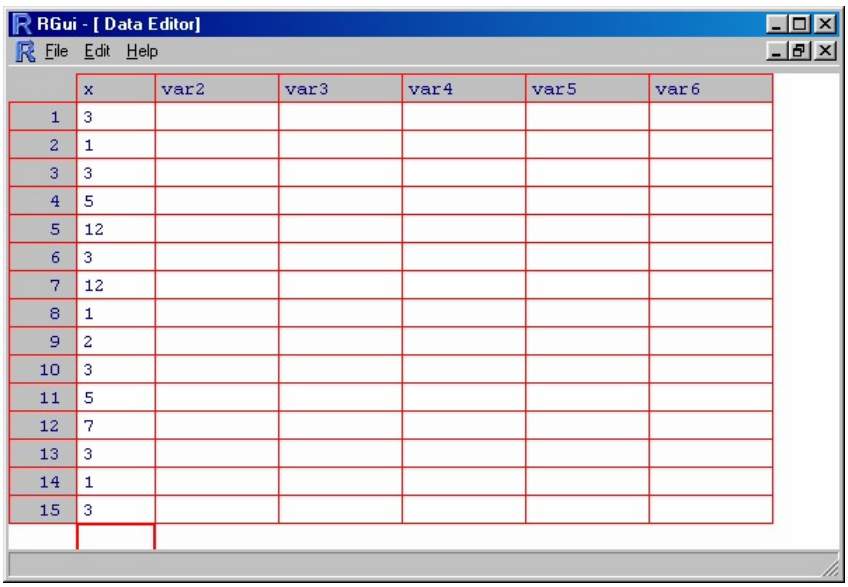
Sometimes you will want to remove specific objects from the workspace. This is easily accomplished with the remove function, rm(object) with the object name as the argument.

```
> rm(q)
```

```
> ls()
[1] "v1" "v2" "x1" "x2" "x3" "z"
> rm(v1, z)
> ls()
[1] "v2" "x1" "x2" "x3"
```

Editing data objects

R has a built in data editor which you can use to edit existing data objects. This can be particularly helpful to edit imported files easily to correct entries or if you have multiple data entries to edit beyond just simple editing of a particular entry. The data editor has a spreadsheet like interface as depicted in Figure 3-1, but has no spreadsheet functionality.



	x	var2	var3	var4	var5	var6
1	3					
2	1					
3	3					
4	5					
5	12					
6	3					
7	12					
8	1					
9	2					
10	3					
11	5					
12	7					
13	3					
14	1					
15	3					

Figure 3-1

To use the data editor, use the data.entry function with the variable being edited as the argument:

```
> x<-c(3,1,3,5,12,3,12,1,2,3,5,7,3,1,3)
> data.entry(x)
```

All changes made using the data editor are automatically saved when you close the data editor. Using the Edit menu option “Data editor”, which brings up a dialog box asking which object to edit, is an alternative way to access the data editor.

Saving your work

R has a few different options for saving your work: save to file, savehistory, and save workspace. Save to file saves everything, savehistory saves commands and objects and save image just saves the objects in the workspace. Let's explain a little more about these and how to use them.

Save to file

Save to file is an option available under the file menu. This option saves everything – commands and output – to a file and is the most comprehensive method of saving your work. This option produces a save as dialog box, making saving the file in a specific directory easy. It produces a text file format viewable in a text editor or word processor or custom specified file type using the all files option and typed in file type. This method of saving is most familiar and simplest, but sometimes you may not want to save everything, particularly when you have large amounts of output and only want to save commands or objects.

Savehistory

This history of what you did in a session can be saved in a *.Rhistory file using the savehistory function. This will save everything typed into the command line prior to the savehistory() function call in the session without R formatting or specific output (versus Save to file which includes all output and formatting).

```
> x<-c(1,2,3,4,5)
> x
[1] 1 2 3 4 5
> savehistory(file="shortSession.Rhistory")
```

This creates a *.Rhistory file in the main R directory (C:\Program Files\R* where * is the current version of R) unless otherwise specified. This file should be readable by a text editor, such as notepad, as in Figure 3-2.

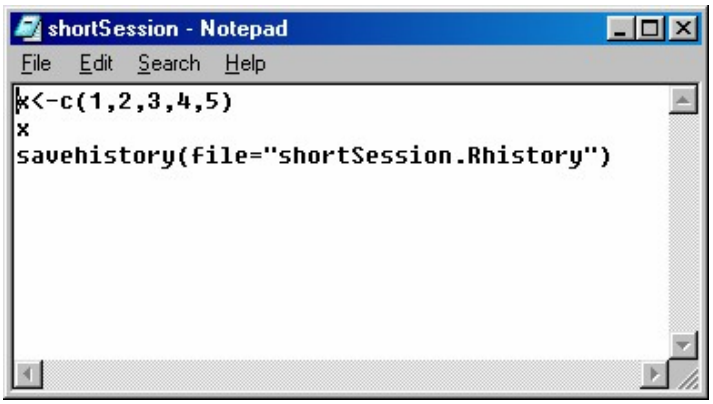


Figure 3-2

Saving workspace image

The option to save the workspace saves only the objects you have created, not any output you have produced using them. The option to save the workspace can be performed at any time using the save.image () command (also available as "Save Workspace" under the file menu) or at the end of a session, when R will ask you if you want to save the workspace.

```
> x<-c(1,2,3,4,5)
> x
[1] 1 2 3 4 5
> save.image()
```

This creates an R workspace file. It defaults to having no specific name (and will be overwritten the next time you save a workspace with no specific name), but you can do save.image(filename) if you want to save different workspaces and name them.

Note that R will automatically restore the latest saved workspace when you restart R with the following message

```
[Previously saved workspace restored]
```

To intentionally load a previously saved workspace use the load command (also available under the file menu as "Load Workspace").

```
> load("C:/Program Files/R/rw1062/.RData")
> x
[1] 1 2 3 4 5
```

Importing Files

We have seen that entering data can be done from within R by using the scan function, by directly entering data when creating a data object, and by using the data editor. What about when you have a data file that you want to import into R, which was made in another program? This section touches on the basics of answering these questions.

It is of note here that there is a manual available for free on the R site and on the R help menu (if manuals were installed as part of the installation) called "R Data Import/Export" which covers in detail the functionality R has to import and export data. Reading this is highly recommended to the user working extensively with importing or exporting data files. This manual covers importing data from spreadsheets, data, and networks.

The first thing to do before importing any file is to tell R what directory your file is in. Do this by going under the File menu and choosing the "Change dir" option", which produces the dialog box illustrated in Figure 3-3. Type in or browse to the directory of your data file and press the OK button.

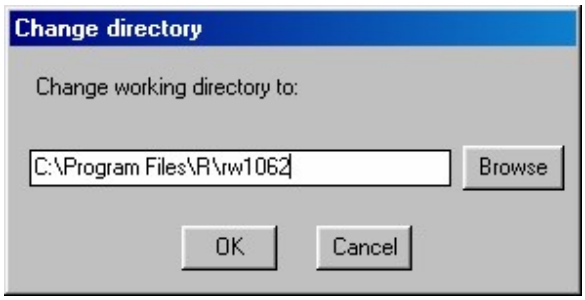


Figure 3-3

Importing using the function read. *()

The most convenient form to import data into R is to use the read functions, notably read.table(). This function will read in a flat file data file, created in ASCII text format. In Notepad you can simply save such a file as a regular text file (extension *.txt). Many spreadsheet programs can save data in this format. Figure 3-4 gives an example of what this format should look like in Notepad:

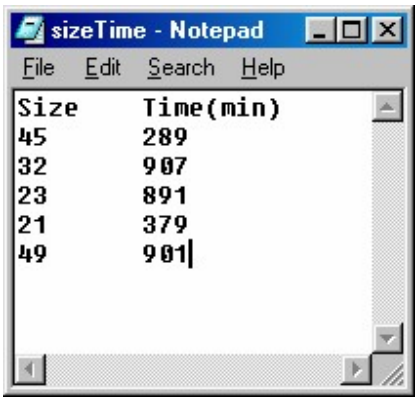


Figure 3-4

Using read.table with arguments of file name and header=T (to capture column headings), such a file can easily be read in R as a data frame object:

```
> sizeTime<-read.table("sizeTime.txt",header=T)
> sizeTime
  Size Time.min.
1   45       289
2   32       907
3   23       891
4   21       379
5   49       901
```

There are some additional read function variants. Notably read.csv() which will read comma delineated spreadsheet file data, which most spreadsheets can save files as.

Importing with scan ()

The scan function can be used with an argument of a file name to import files of different types, notably text files (extension *.txt) or comma separated data files (extension *.csv). Data from most spreadsheet programs can be saved in one or both of these file types. Note that scan() tends to produce formatting problems when reading files much more often than read and is not recommended for importing files.

Package foreign

Also of note is an R package called foreign. This package contains functionality for importing data into R that is formatted by most other statistical software packages, including SAS, SPSS, STRATA and others. Package foreign is available for download and installation from the CRAN site.

Troubleshooting Importing

Finally, sometimes you may have data in a format R will not understand. Sometimes for trouble imports with formatting that R cannot read, using scan () or the data editor to enter your data may be a simple and easy solution. Another trick is to try importing the data into another program, such as a spreadsheet program, and saving it as a different file type. In particular saving spreadsheet data in a text (comma or tab delineated) format is simple and useful. Caution should be used as some spreadsheet programs may restrict the number of data values to be stored.

Getting Help

Virtually everything in R has some type of accessible help documentation. The challenge is finding the documentation that answers your question. This section gives some suggestions for where to look for help.

Program Help Files

The quickest and most accessible source of help when using R is to use the on-line help system that is part of R. This includes on-line documentation for the R base packages, as well as on-line documentation for any loaded packages.

Finding help when you know the name of what it is your asking for help on is easy, just use the help function with the topic of interest as the argument of the help function. For example to get help on function sum:

```
> help(sum)
```

This produces a help file as depicted in Figure 3-5.

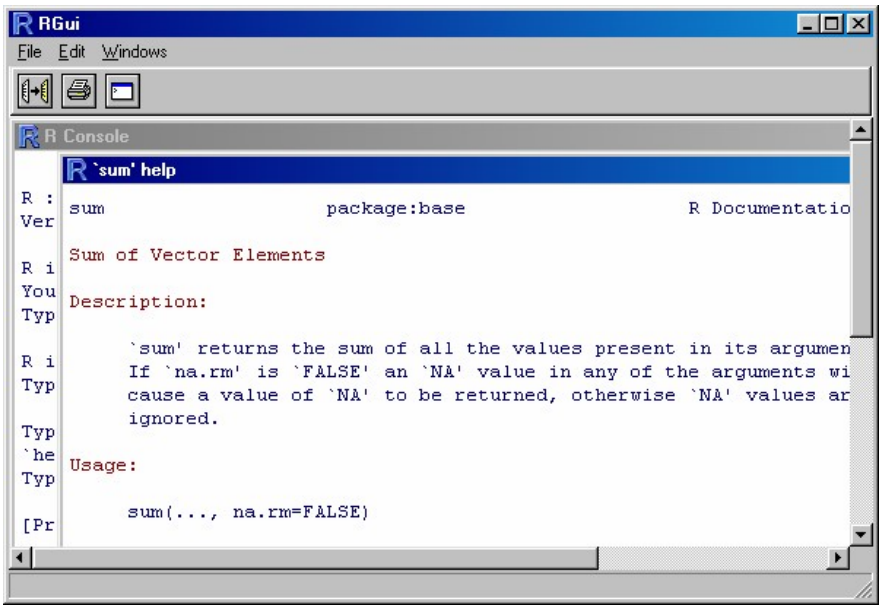


Figure 3-5

As an alternative to calling the help function you can just put a question mark in front of the topic of interest:

```
> ?sum
```

If you don't know the exact name of what you're looking for, the apropos() function can help. To use this function type in a part of the word you are looking for using quotes as the function argument. The result of calling apropos is that you will get a list of everything that matches that clue, as in Figure 3-6 for apropos("su"), and you can then do a help search on your term of interest.

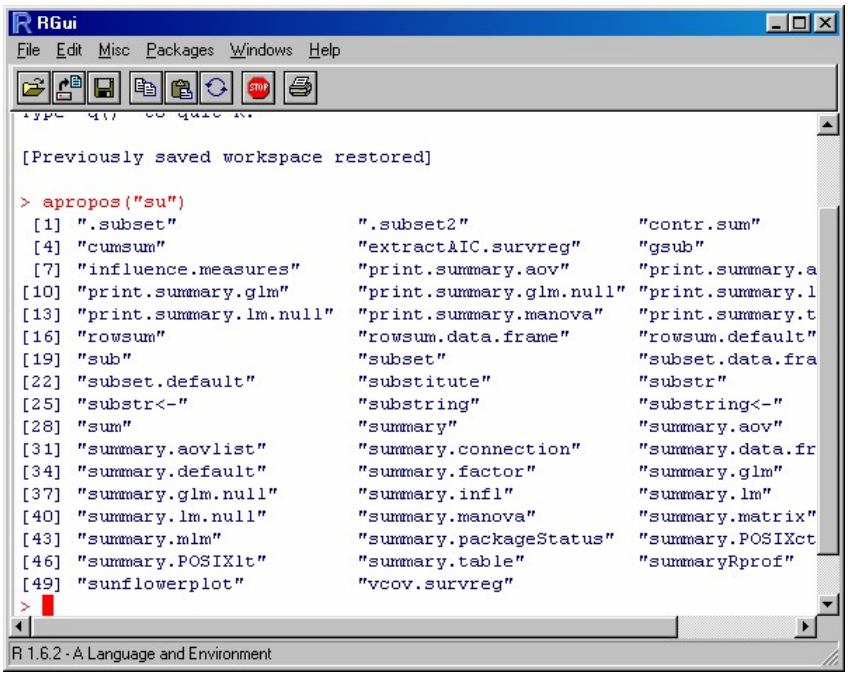


Figure 3-6

Note that on-line help is especially useful for describing function parameters, which this book will not discuss in great detail because many functions in R have lots of optional parameters (read.table for example has about 12 different optional parameters). The reader should be comfortable with looking up functions using help to get detailed parameter information to use R functions to suit their needs.

Documents

The R system and most packages are fairly well documented, although the documentation tends to be technical rather than tutorial. The R system itself comes with several documents that are installed as part of the system (recommended option). These are under the "Manuals" option on the "Help" menu.

In addition to manuals that cover R in general, most packages have their own documents. R has a system where package contributors create pdf files in standard formats with explain the technical details of the package, including a description of the package and its functionality. These documents also generally list the name and contact information for the author(s) of the package. These documents are available from the "Package Sources" section of the CRAN site (cran.r-project.org) where the relevant package is listed and are always listed as Reference Manual, although they save with a file name corresponding to the

package name. They can be downloaded and saved or viewed on-line using a web browser that reads pdf files.

Books

There are several books written that are about or utilize R. In addition, there are several books that cover the related topics of S and S-Plus. These are included in the resource section at the end of the book.

On-line Discussion Forums

One of the best sources of help if you have a specific question is the on-line discussion forums where people talk about R. These forums serve as "technical support" since R is open source and has no formal support system. There are usually many well-informed users who regularly read these discussion lists. A guide to such lists is found at *www.r-project.org* as depicted in Figure 3-7. In addition many other forums exist on the web where questions about R may be posted.

Figure 3-7