

Running R

There are several ways you can run R:

- Interactively using its programming language. You can see the result of each function call immediately after you submit it.
- Interactively using one of several graphical user interfaces (GUIs) that you can add on to R. Some of these use programming and some use menus much like Stata.
- Noninteractively in batch mode using its programming language. You enter your program into a file and run it all at once.

You can ease your way into R by continuing to use Stata or your favorite spreadsheet program to enter and manage your data and then use one of the methods below to import and analyze it. As you find errors in your data (and you know you will), you can go back to your other software, correct them, and then import it again. It is not an ideal way to work, but it does get you into R quickly.

3.1 Running R Interactively on Windows

You can run R programs interactively in several steps:

1. Start R by choosing *Start>All Programs>R>R x.x.x* (where x.x.x is the version of R you are using). The main R Console window will appear looking like the left window in Fig. 3.1. Then enter your program choosing one of the methods described in steps 2 and 3 below.
2. Enter R functions into the R console. You can enter function calls into the console one line at a time at the “>” prompt. R will execute each line when you press the Enter key. If you enter them into the console, you can retrieve them with the up arrow key and edit them to run again. We find it much easier to use the program editor described in the next step.

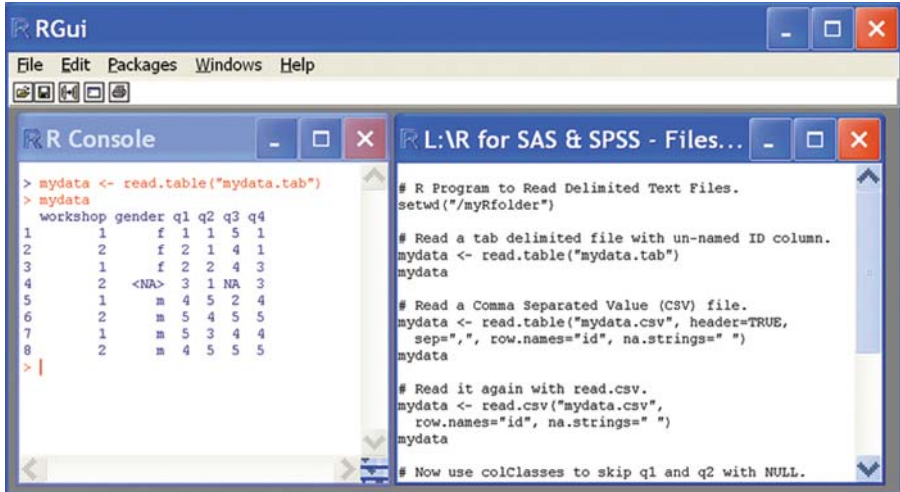


Fig. 3.1. R graphical user interface on Microsoft Windows.

If you type the beginning of an R function, such as “me” and press Tab, R will show you all of the R functions that begin with those letters, such as `mean` or `median`. If you enter the name of a function and an open parenthesis, such as “`mean(,`” R will show you the arguments that you can use with that function.

3. Enter R functions into the R Editor. Open the R Editor by choosing *File>New Script*. You can see it in the bottom right corner of Fig. 6.1. You can enter programs as you would in the Stata commandline.
4. Submit your program from the R Editor. To submit just the current line, you can hold the Ctrl key down and press “r,” for run, or choose *Edit>Run line or selection*. To run a block of lines, select them first and then submit them the same way. To run the whole program, choose *Edit>Run All*.
5. As you submit program statements, they will appear in the R Console along with results and/or error messages. Make any changes you need and submit the program again until finished. You can clear the console results by choosing *Edit>Clear console* or by holding the Ctrl key down and pressing “l” (i.e. CTRL-l). See *Help>Console* for more keyboard shortcuts.
6. Save your program and output. Click on either the console window or the R Editor window to make it active and choose *File>Save to file*. Unlike in Stata, the console output will contain the commands and their output blended together.
7. Save your data and any functions you may have written. The data and/or function(s) you created are stored in an area called your workspace. You can save that with the menu selection, *File>Save Workspace...* In a later R session you can retrieve it with *File>Load Workspace...* You can also save your workspace using the `save.image` function:

```
save.image(file="myWorkspace.RData")
```

Later, you can read the workspace back in with the function call:

```
load("myWorkspace.RData")
```

For details, see Chapter 13, “Managing Your Files and Workspace.”

- Optionally save your history. R has a history file that saves all of the function calls you submit in a given session. This is just like the Stata log menu. Unlike Stata, however, the history of the past function calls is not cumulative on Windows computers. You can save the session history to a file using *File>Save History...* and you can load it into future session with *File>Load History...* There are also various R functions to do these tasks.

```
savehistory(file="myHistory.Rhistory")
loadhistory(file="myHistory.Rhistory")
```

Note that the filename can be anything you like, but the extension should be “.Rhistory.” In fact the entire filename will be simply “.Rhistory” if you do not provide one. We prefer to always save a cumulative history file automatically. For details, see Appendix C.

- To quit R, choose *File>Exit* or submit the function `quit()` or just `q()`. R offers to save your workspace automatically on exit. If you are using the `save.image` and `load` functions to tell R where to save and retrieve your workspace in step 4 above, you can answer *No*. If you answer *Yes*, it will save your work in the file “.RData” in your default working directory. Next time you start R, it will load the contents of the .RData file automatically. Creating a .RData file this way is a convenient way to work. However, we prefer naming each project ourselves as described in step 4 above.

3.2 Running R Interactively on Macintosh

Like Stata, R programs run interactively on the Macintosh. R does so in several steps.

- Start R by choosing R in the Applications folder. The R console window will appear (see left window in Fig. 3.2). Then enter your program choosing one of the methods described in steps 2 and 3 below.
- Enter R functions in the console window. You can enter function calls into the console one line at a time at the “>” prompt. R will execute each line when you press the Enter key. If you enter them into the console, you can retrieve them with the up arrow key and edit them to run again. We find it much easier to use the program editor described in the next step. If you type “me” at the command prompt and press Tab or hold the Command key down and press “.” (i.e., CTRL-period), R will show you all of the R functions that begin with those letters, such as `mean` or `median`. When you type a whole function name, the functions arguments will appear below it in the console window.

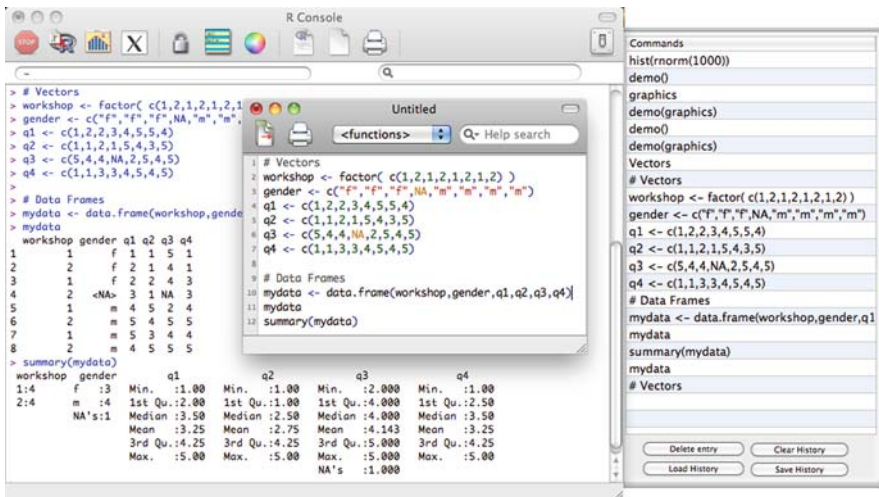


Fig. 3.2. R graphical user interface on Macintosh.

- Enter R functions into the R Editor. Open the R Editor by choosing *File>New Document*. The R Editor will start with an empty window. You can see it in the center of Fig. 3.2. You can enter R programs as you would on the Stata command line or write *do* or *ado* files in the Stata editor.
- Submit your program from the R Editor. To submit one or more lines, highlight them, then hold the Command key, and press Return, or choose *Edit>Execute*. To run the whole program, select it by holding down the Command key and pressing “a,” and then choose *Edit>Execute*.
- As you submit program statements, they will appear in the R Console along with results and/or error messages. Make any changes you need and submit the program again until finished.
- Save your program and output. Click on a window to make it the active window and choose *File>Save to file*. The function calls and their output are blended together, unlike Stata.
- Save your data and any functions you may have written. The data and/or function(s) you created are stored in an area called your *workspace*. You can save your workspace with *Workspace>Save Workspace File...* In a later R session you can retrieve it with *Workspace>Load Workspace File...* You can also perform these functions using the R functions `save.image` and `load`.

```
save.image(file="myWorkspace.RData")
```

```
load("myWorkspace.RData")
```

For details, see Chapter 13, “Managing Your Files and Workspace.”

8. Optionally save your history. R has a history file that saves all of the functions you submit in a given session (and not the output). This is just like the Stata log menu. The history file is not cumulative on Macintosh computers. You can view your history by clicking on the *Show/Hide R command history* icon in the console window (to the right of the lock icon). You can see the command history window on the right side of Fig. 3.2. Notice that it has alternating stripes, matching its icon. Clicking the icon once makes the history window slide out to the right of the console. Clicking it again causes it to slide back and disappear. You can see the various buttons at the bottom of the history, such as *Save History* or *Load History*. You can use them to save your history or load it from a previous session. You can also use R functions to do these tasks.

```
savehistory(file="myHistory.Rhistory")
```

```
loadhistory(file="myHistory.Rhistory")
```

The filename can be anything you like, but the extension should be “.Rhistory.” In fact the entire filename will be simply “.Rhistory” if you do not provide one. We prefer to always save a cumulative history file automatically. For details, see Appendix C.

9. Exit R by choosing *R>Quit R*. Users of any operating system can quit by submitting the function `quit()` or just `q()`. R will offer to save your workspace automatically on exit. If you are using the `save.image` and `load` functions to tell R where to save/retrieve your workspace in step 4 above, you can answer *No*. If you answer *Yes*, it will save your work in the file “.RData” in your default working directory. Next time when you start R, it will load the contents of the .RData file automatically. Creating a .RData file this way is a convenient way to work. However, we recommend naming each project yourself, as described in step 4 above.

3.3 Running R Interactively on Linux or UNIX

You can run R programs interactively in several steps.

1. Start R by entering the command “R,” which will bring up the “>” prompt, where you enter commands. For a wide range of options, refer to Appendix B , An Introduction to R [52], available at <http://www.r-project.org/> under *Manuals*, or in your R *Help* menu. You can enter R functions using either of the methods described in steps 2 and 3 below.
2. Enter R functions into the console one line at a time at the “>” prompt. R will execute each line when you press the Enter key. You can retrieve a function call with the up arrow key and edit it, and then press Enter to run again. You can include whole R programs from files with the `source` function. For details, see Section 3.4, “Running Programs That Include Other Programs.” If you type the beginning of an R function name like “me” and press the Tab key, R will show you all of the R functions that

begin with those letters, such as `mean` or `median`. If you type the function name and an open parenthesis like `“mean(”` and press Tab, R will show you the arguments you can use to control that function.

3. Enter R functions into a text editor. Although R for Linux or UNIX does not come with its own GUI or program editor, a popular alternative is to use the Emacs editor in ESS mode. It color-codes your programs to help find syntax errors. You can submit your programs directly from Emacs to R. See the R FAQ at <http://www.r-project.org/> under R for Emacs for details.
4. Save your program and output. Linux or UNIX users can route input and output to a file with the `sink` function. You must specify it in advance of any output you wish to save.

```
sink("myTranscript.txt", split=TRUE)
```

The argument `split=TRUE` tells R to display the text on the screen as well as route it to the file. The file will contain a transcript of your work. The function calls and their output are blended together, unlike Stata.

5. Save your data and any functions you may have written. The data and/or function(s) you created are stored in an area called your workspace. Users of any operating system can save it by calling the `save.image` function.

```
save.image(file="myWorkspace.RData")
```

Later, you can read the workspace back in with the function call

```
load("myWorkspace.RData")
```

For details, see Chapter 13, “Managing Your Files and Workspace.”

6. R has a history file that saves all of the functions you submit in a given session. This is just like the Stata log menu. The Linux/UNIX version of R saves a cumulative set of function calls across sessions. You can also save or load your history at any time with the `savehistory` and `loadhistory` functions.

```
savehistory(file="myHistory.Rhistory")
```

```
loadhistory(file="myHistory.Rhistory")
```

Note that the filename can be anything you like, but the extension should be `“.Rhistory.”` In fact the entire filename will be simply `“.Rhistory”` if you do not provide one.

7. Quit R by submitting the function `quit()` or just `q()`. R offers to save your workspace automatically on exit. If you are using the `save.image` and `load` functions to tell R where to save/retrieve your workspace in step 4 above, you can answer *No*. If you answer *Yes*, it will save your work in the file `“.RData”` in your default working directory. Next time you start R, it will load the contents of the `.RData` file automatically. Creating an `.RData` file this way is a convenient way to work. However, we prefer naming each project ourselves as described in step 4 above.

3.4 Running Programs That Include Other Programs

When you find yourself using the same block of code repeatedly in different programs, it makes sense to save it to a file and include it into the other programs where it is needed. Stata does this using a global constant, where the program code is assigned a unique name and recalled into the program code by giving the constant with a leading dollar sign. To include a program in R, use the `source` function

```
source("myprog.R")
```

One catch to keep in mind is that by default R will not display any results that sourced files may have created. Of course, any objects they create — data, functions, and so forth — will be available to the program code that follows. If the program you source creates actual output that you want to see, you can source the program in the following manner:

```
source("myprog.R", echo=TRUE)
```

This will show you all of the output created by the program. If you prefer to see only some results, you can wrap the `print` function around only those functions whose output you do want displayed. For example, if you sourced the following R program, it would display the standard deviation, but not the mean:

```
x <- c(1, 2, 3, 4, 5)
mean(x)          # This result will not display.
print( sd(x) )  # This one will.
```

An alternative to using the `source` function is to create your own R package. However, that is beyond the scope of this book.

3.5 Running R in Batch Mode

You can write a program to a file and run it all at once, routing its results to another file (or files). This is called batch processing. If you had a program named `myprog.do`, you would run it with the following command:

```
"C:\Program Files\Stata10\wstata" /b do myprog.do
```

Stata would run the program and place the results into a log file called `myprog.log`.

In R, you can find the details of running batch on your operating system by starting R and entering the following command. Note that the letters of `BATCH` must be all uppercase.

```
help(BATCH)
```

The following operating system command is an example of running an R batch job on Microsoft Windows. You will need to change the path of Rterm.exe to reflect its location on your computer and fill in your version in place of x.x.x.

```
"C:\Program Files\R\R-x.x.x\bin\Rterm.exe"  
--no-restore --no-save < myprog.r > myprog.out
```

The command wraps to two lines in this book, but enter it as a single line. It is too long to fit in a standard cmd.exe window, so you will need to change its default width from 80 to something wider, like 132. R will execute myprog.r and write the results to myprog.out.

It is easier to write a small batch file like myR.bat.

```
"G:\Program Files\R\R-x.x.x\bin\Rterm.exe"  
--no-restore --no-save < %1 > %1.Rout 2>&1
```

Once you have saved that in the file myR.bat, you can then submit batch programs with the following command. It will route your results to myprog.Rout. You can also download this batch file with this book's example programs and data sets at <http://r4stats.com>.

```
myR myprog.R
```

UNIX users can run a batch program with the following command. It, too, will write your output to myprog.Rout.

```
R CMD BATCH myprog.R
```

There are, of course, many options to give you more control over how your batch programs run. See the help file for details.

3.6 Graphical User Interfaces

The main R installation provides an interface to help you enter programs. It does not include a point-and-click graphical user interface (GUI) for running analyses. There are, however, several GUIs written by R users. You can learn about several at the main R web site, <http://www.r-project.org/> under *Related Projects* and then *R GUIs*.

3.6.1 R Commander

Our favorite GUI for general statistical analysis is John Fox's R Commander [13], which is similar to the Stata GUI. It provides menus for many analytic and graphical methods and shows you the R function calls that it enters, making it easy to learn to program in R as you use it. Since it does not come with the main R installation, you have to install it one time with the `install.packages` function.


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

R Commander uses *many* other packages, and R will download and install them for you if you use the `dependencies=TRUE` argument.

Let us review the steps of a basic R Commander session. Below are the steps we followed to create the screen image you see in Fig. 3.3 .

1. We started R. For details see the section “Running R Interactively on Windows,” or similarly named sections for other operating systems previously covered in this chapter.
2. Then, from within R itself we started R Commander by loading its package from the library. That brought up the window that looks something like Fig. 3.3.

```
library("Rcmdr")
```

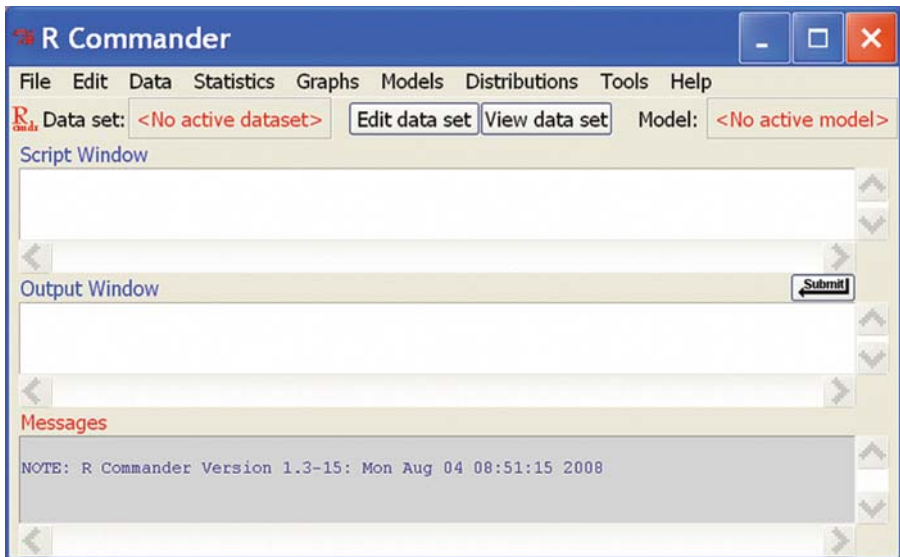
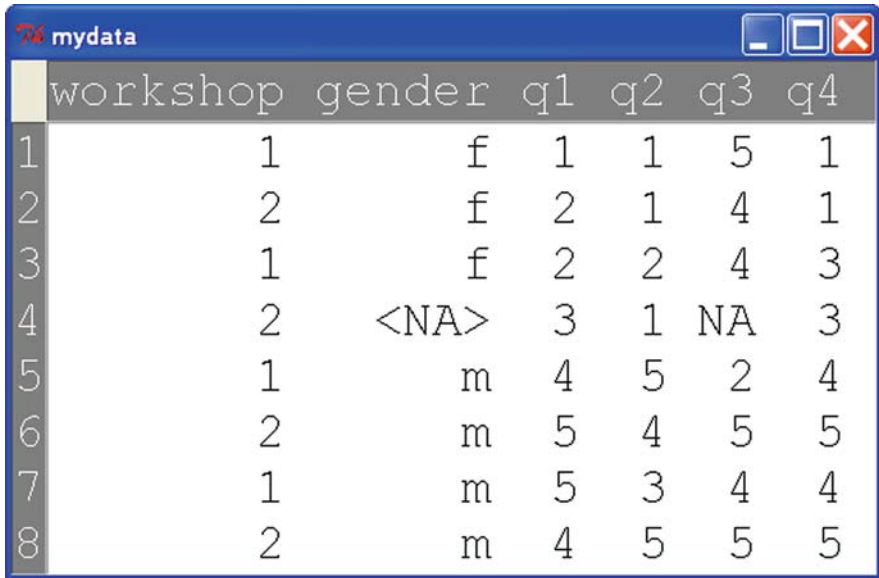


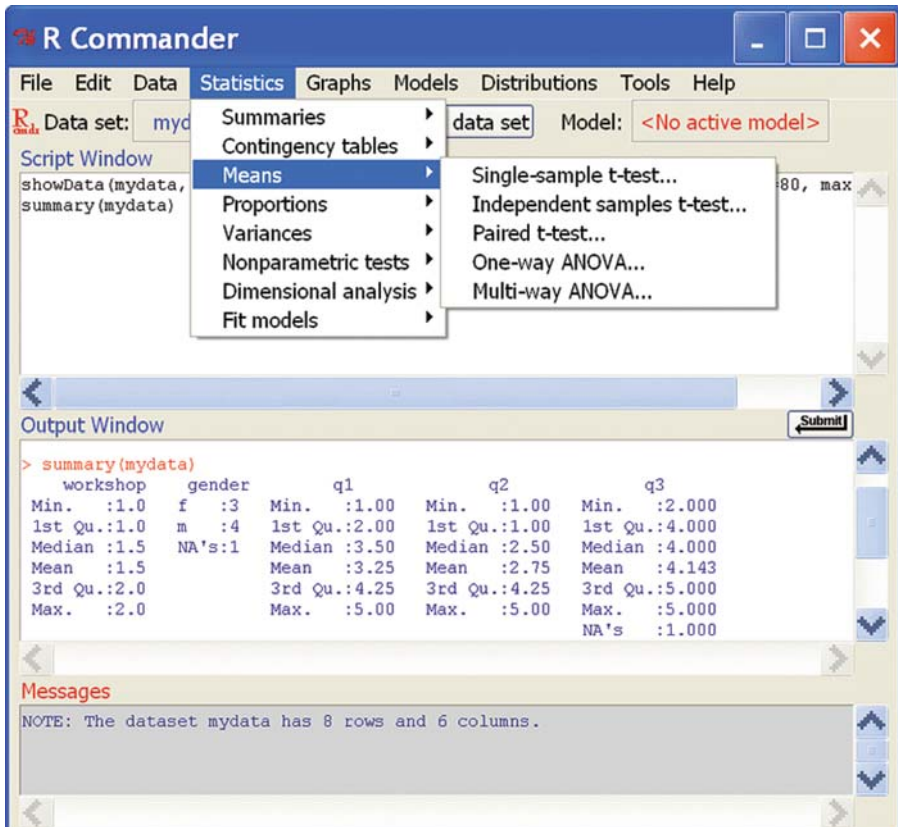
Fig. 3.3. The R Commander GUI, before any work is done.

3. We then chose *Data>Load a data set* and browsed to myRfolder, where our practice data sets are stored. We had to tell it to look for *All Files* because by default it looks for *.RDA* file types and ours are *.RData*. We then chose the file *mydata.RData*. R Commander uses a different file extension because it is unable to deal with more than one data set at a time. Since *mydata.RData* contains only one data set, it works fine.
4. We next click on the *View data set* button in order to view the data. Then the data appeared, as shown in Fig. 3.4.
5. We then chose *Statistics>Summaries>Active Data Set*. The output you see on the bottom of the screen in Fig. 3.5.



	workshop	gender	q1	q2	q3	q4
1	1	f	1	1	5	1
2	2	f	2	1	4	1
3	1	f	2	2	4	3
4	2	<NA>	3	1	NA	3
5	1	m	4	5	2	4
6	2	m	5	4	5	5
7	1	m	5	3	4	4
8	2	m	4	5	5	5

Fig. 3.4. R Commander's Data Viewer window that appears when you click the *View data set* button.



R Commander

File Edit Data **Statistics** Graphs Models Distributions Tools Help

Data set: myd Model: <No active model>

Script Window

```
showData(mydata,
summary(mydata))
```

Output Window

```
> summary(mydata)
  workshop gender      q1      q2      q3
Min.   :1.0   f   :3   Min.   :1.00  Min.   :1.00  Min.   :2.000
1st Qu.:1.0   m   :4   1st Qu.:2.00  1st Qu.:1.00  1st Qu.:4.000
Median :1.5   NA's:1   Median :3.50  Median :2.50  Median :4.000
Mean   :1.5           Mean   :3.25  Mean   :2.75  Mean   :4.143
3rd Qu.:2.0           3rd Qu.:4.25  3rd Qu.:4.25  3rd Qu.:5.000
Max.   :2.0           Max.   :5.00  Max.   :5.00  Max.   :5.000
                        NA's   :1.000
```

Messages

```
NOTE: The dataset mydata has 8 rows and 6 columns.
```

Fig. 3.5. The R Commander GUI with work in progress.

6. Finally, we chose *Statistics>Means...* You see that the menu is still open, showing that we can choose various t-tests and analysis of variance (ANOVA) procedures.

You can learn more about R Commander at <http://cran.r-project.org>, under *Packages*.

3.6.2 Rattle for Data Mining

Graham William's Rattle package [57] provides a tabbed-dialog box style of user interface. Although its emphasis is on data mining, the interface is useful for standard statistical analyses as well. Its name stands for the *R analytical tool to learn easily*. That name fits it well, as it is very easy to learn. Its point-and-click interface writes and executes R programs for you.

Before you install the `rattle` package, you must install some other tools. See the web site for directions <http://rattle.togaware.com>. Once it is installed, you load it from your library in the usual way.

```
> library("rattle")
```

```
Rattle, Graphical interface for data mining
  using R, Version 2.2.64.
```

```
Copyright (C) 2007 Graham.Williams@togaware.com, GPL
```

```
Type "rattle()" to shake, rattle, and roll your data.
```

As the instructions tell you, simply enter the call to the `rattle` function to bring up its interface.

```
> rattle()
```

The main Rattle interface shown in Fig. 3.6 will then appear. It shows the steps it uses to do an analysis on the tabs at the top of its window. You move from left to right, clicking on each tab to do the following steps:

1. Data. Choose your data type from a Comma Separated Value (CSV) file, Attribute Relation File Format (ARFF), Open DataBase Connectivity (ODBC), .RData file, R data object already loaded or created before starting Rattle, or even manual data entry.
2. Select. Choose your variables and the roles they play in the analysis. In Fig. 3.6 we have chosen gender as the target variable (dependent variable) and the other variables as inputs (independent variables or predictors).
3. Explore. Examine the variables using summary statistics, distributions, interactive visualization via GGobi [47], correlation, hierarchical cluster analysis of variables, and principal components. A very interesting feature in distribution analysis is the application of Benford's law, an examination of the initial digits of data values that people use to detect fraudulent data (e.g., faked expense account values)

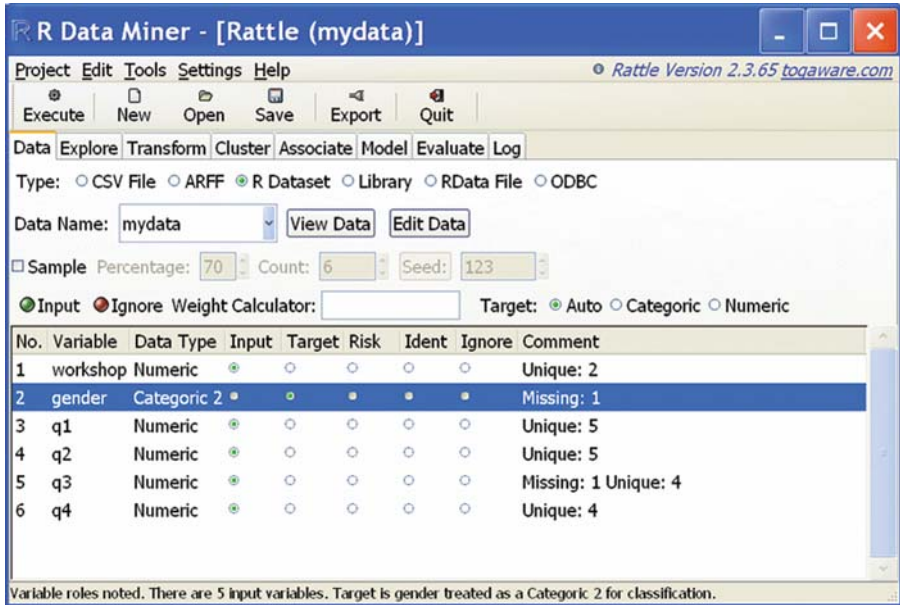


Fig. 3.6. Rattle GUI for data mining.

4. Transform. Replace missing values with reasonable estimates (imputation), convert variables to factors, or look for outliers.
5. Model. Apply models from tree, boost, forest, SVM, regression, or all.
6. Evaluate. Assess model quality and compare different models using confusion tables, lift charts, ROC curves, and so forth.
7. Log. See the R program that Rattle wrote for you to do all of the steps.

Figure 3.7 shows an R program that Rattle wrote when asked for box plots of my data (box plots not shown).

3.6.3 JGR Java GUI for R

The Java GUI for R, JGR [20] (pronounced “jaguar”), is very similar to R’s own simple interface, making it very easy to learn. Written by Markus Helbig, Simon Urbanek, and Martin Theus, JGR provides some very helpful additions to R, like syntax checking in its program editor. It also provides the help files in a way that lets you execute any part of an example you select. That is very helpful when trying to understand a complicated example.

JGR is installed differently than most R packages. On Microsoft Windows or Apple Macintosh, you download two programs: an installer and a launcher. Running the installer installs JGR and double-clicking the launcher starts it up. The JGR web site that contains both programs is <http://rosuda.org/JGR/>.

Linux users follow slightly different steps that are described at the site.

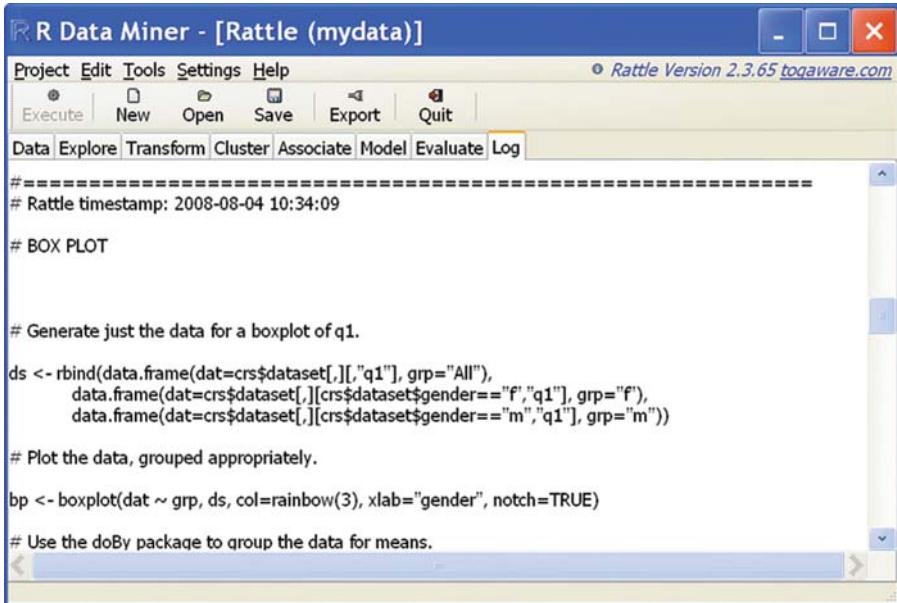


Fig. 3.7. An R program written by Rattle to do a box plot.

We started JGR by double-clicking on its launcher and opened an R program using *File>Open Document*. You can see the program in Fig. 3.8. Note that the JGR program editor has automatically color-coded my comments, function names, and arguments, making it much easier to spot errors. In the printed version of this book those colors are displayed as shades of gray.

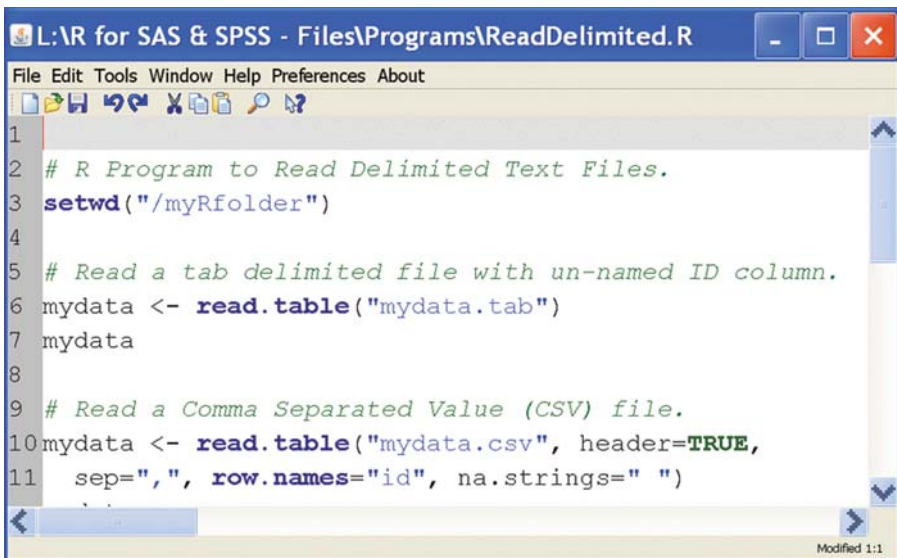


Fig. 3.8. Color-coded editor in JGR helps prevent typing errors.

In the next example, we typed "cor(" into the bottom of the console area shown in Fig. 3.9. JGR then displayed a box showing the various arguments that control the cor function for doing correlations. That is very helpful when you are learning!

The screenshot shows the JGR console window with the following content:

```
> mydata <- read.table("mydata.tab")
> mydata
  workshop gender q1 q2 q3 q4
1         1     f  1  1  5  1
2         2     f  2  1  4  1
3         1     f  2  2  4  3
4         2  <NA>  3  1 NA  3
5         1     m  4  5  2  4
6         2     m  5  4  5  5
7         1     m  5  3  4  4
8         2     m  4  5  5  5
```

Below the console output, a yellow tooltip box displays the help text for the `cor` function:

```
cor (x, y = NULL, use = "all.obs", method = c("pearson",
      "kendall", "spearman"))
```

Fig. 3.9. JGR showing arguments that you might choose for the cor function.

JGR's *Package Manager* makes it easier to control which packages you are using (Fig. 3.10). Simply checking the boxes under "loaded" will load those packages from your library. If you also check it under "default," JGR will load them every time you start JGR. Without JGR's help, automatically loading packages would require editing your `.Rprofile`.

JGR's *Object Browser* makes it easy to manage your workspace; see Fig. 3.11. Selecting different tabs across the top enable you to see the different types of objects in your workspace. We right-clicked on gender, which brought up the box listing the number of males, females, and missing values (NAs). If you have a list of models, you can sort them easily by various measures, like their R-squared values.

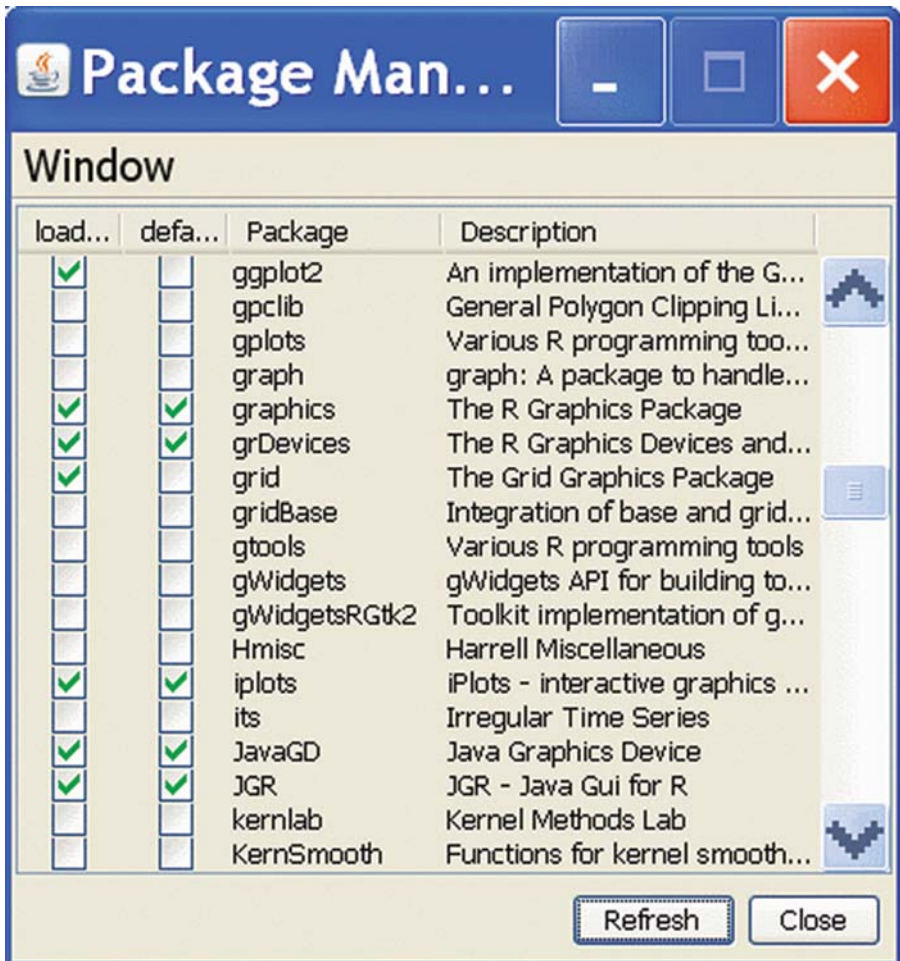


Fig. 3.10. JGR's Package Manager, which allows you to load packages from the library on demand or at startup.

Double-clicking on a data frame in the Object Browser starts the *Data Table* editor (Fig. 3.12), which is much nicer than the one built into R. It lets you rename variables, search for values, sort by clicking on variable names, cut and paste values, and add or delete rows or columns.

There are many more useful features in JGR that are described on its web site.

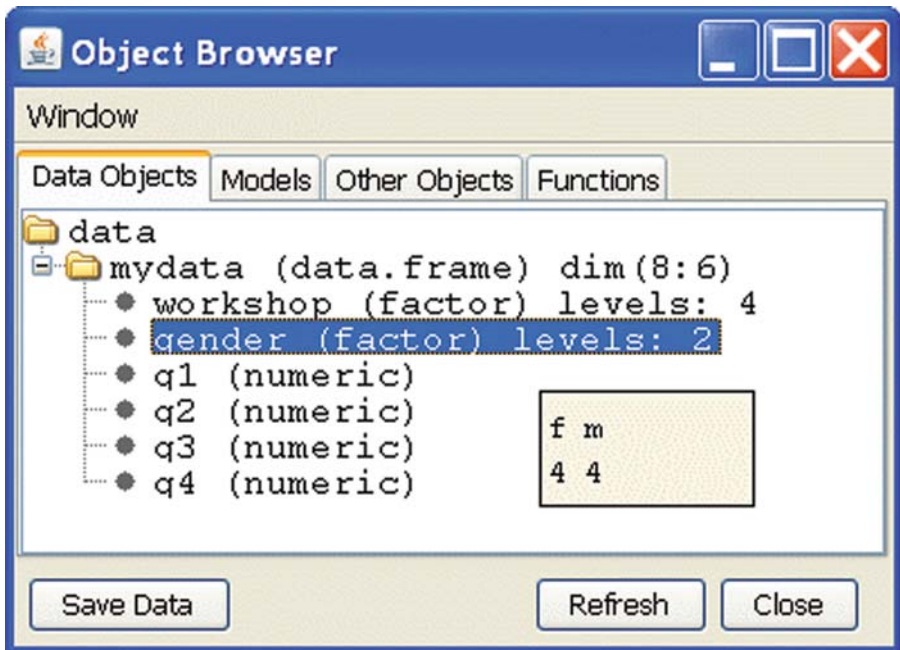


Fig. 3.11. JGR's Object Browser shows information about each object in your workspace.

row.names	workshop	gender	q1	q2	q3	q4
1	1	f	1.0	1.0	5.0	1.0
2	2	f	2.0	1.0	4.0	1.0
3	1	f	2.0	2.0	4.0	3.0
4	2	NA	3.0	1.0	NA	3.0
5	1	m	4.0	5.0	2.0	4.0
6	2	m	5.0	4.0	5.0	5.0
7	1	m	5.0	3.0	4.0	4.0
8	2	m	4.0	5.0	5.0	5.0

Fig. 3.12. JGR's Data Table editor, an improvement over R's primitive one.