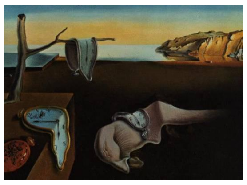


Journal of Statistical Software

September 2005, Volume 14, Issue 9. *http://www.jstatsoft.org/*

The R Commander: A Basic-Statistics Graphical User Interface to R

John Fox
McMaster University

Abstract

Unlike S-PLUS, R does not incorporate a statistical graphical user interface (GUI), but it does include tools for building GUIs. Based on the **tcltk** package (which furnishes an interface to the Tcl/Tk GUI toolkit), the **Rcmdr** package provides a basic-statistics graphical user interface to R called the "R Commander." The design objectives of the R Commander were as follows: to support, through an easy-to-use, extensible, cross-platform GUI, the statistical functionality required for a basic-statistics course (though its current functionality has grown to include support for linear and generalized-linear models, and other more advanced features); to make it relatively difficult to do unreasonable things; and to render visible the relationship between choices made in the GUI and the R commands that they generate. The R Commander uses a simple and familiar menu/dialog-box interface. Top-level menus include *File*, *Edit*, *Data*, *Statistics*, *Graphs*, *Models*, *Distributions*, *Tools*, and *Help*, with the complete menu tree given in the paper. Each dialog box includes a *Help* button, which leads to a relevant help page. Menu and dialog-box selections generate R commands, which are recorded in a script window and are echoed, along with output, to an output window. The script window also provides the ability to edit, enter, and re-execute commands. Error messages, warnings, and some other information appear in a separate messages window. Data sets in the R Commander are simply R data frames, and can be read from attached packages or imported from files. Although several data frames may reside in memory, only one is "active" at any given time. There may also be an active statistical model (e.g., an R `lm` or `glm` object). The purpose of this paper is to introduce and describe the use of the R Commander GUI; to describe the design and development of the R Commander; and to explain how the R Commander GUI can be extended. The second part of the paper (following a brief introduction) can serve as an introductory guide for students who will use the R Commander

Keywords: statistical GUI, statistical software, statistical education, R language.

1. Background and motivation

R (Ihaka and Gentleman 1996; R Core Development Team 2004) is a free, open-source implementation of the S statistical computing language and programming environment. R is a command-driven system: One normally specifies a statistical analysis in R by typing commands—that is, statements in the S language that are executed by the R interpreter. S-PLUS (a commercial implementation of the S language), also incorporates a graphical user interface (GUI) to much of its statistical functionality.

In my opinion, a GUI for statistical software is a mixed blessing: On the one hand, a GUI does not require that the user remember the names and arguments of commands, and decreases the chances of syntax and typing errors. These characteristics make GUIs particularly attractive for introductory, casual, or infrequent use of software.

On the other hand, having to drill one's way through successive layers of menus and dialog boxes can be tedious and can make it difficult to reproduce a statistical analysis, perhaps with variations. Moreover, providing a GUI for a statistical system that includes hundreds (or even thousands) of commands, many incorporating extensive options, can produce a labyrinth. The R Commander GUI described in this paper is not immune to these problems, but I have tried to keep things relatively simple, and to render visible, in a reusable form, the R commands that the GUI generates.

Unlike S-PLUS, R does not include a statistical GUI, but it does furnish tools for building GUIs.[1] The **Rcmdr** package provides a basic-statistics GUI for R, which I call the "R Commander." The design objectives of the R Commander were as follows:

- Most importantly, to provide, through an easy-to-use, cross-platform, extensible GUI, the statistical functionality required for a basic-statistics course.[2] The original target text was David Moore's *The Basic Practice of Statistics, Second Edition* (Moore 2000). With the help of a research assistant (Tony Christensen), I have since examined several other texts, including the third edition of Moore (2004), collected suggestions from a number of individuals, and slightly expanded the horizons of the R Commander—for example, to include linear and generalized-linear models.

- To make it relatively difficult to do unreasonable things (such as calculating the mean of a categorical variable).

- To render visible the relationship between choices made in the GUI and the R commands that they generate. Commands are both pasted into a script window in the R Commander and echoed to an output window (see below). The script window is editable, commands in the window can be executed or re-executed, and new commands can be entered by typing directly in the window. Scripts can also be saved to, and loaded from, files.

[1] The R Commander, described in this paper, is based on the **tcltk** package (Dalgaard 2001, 2002), which provides an interface to Tcl/Tk (Welch 2000).

[2] The examples in this document use the Windows version of R, and parts of the document are specific to the Windows version. R, however, is available on other computing platforms as well (Macintosh computers and Unix/Linux systems), and the use of R and the R Commander on these other systems is very similar to their use under Windows. I focus here on the Windows version of the software because I believe that the large majority of students in basic-statistics classes are Windows users.

One purpose of this paper is to introduce and describe the basic use of the R Commander GUI. In particular, Section 2 of the paper can serve as an introductory guide for students who will use the R Commander. Section 3 describes the design and development of the R Commander; informally assesses the extent to which it has met its goals; and suggests future directions for the project. Section 4 explains how the R Commander can be extended. The final section provides some information for instructors. In addition, the help files for the current version of the **Rcmdr** package are available on the Comprehensive R Archive Network (CRAN) website at `http://CRAN.R-project.org/doc/packages/Rcmdr.pdf`.

2. Using the R Commander

2.1. Starting the R Commander

Once R is running, simply loading the **Rcmdr** package by typing the command `library("Rcmdr")` into the *R Console* starts the R Commander GUI. To function properly under Windows, the R Commander requires the single-document interface (SDI) to R.[3] After loading the package, the *R Commander* window should appear more or less as in Figure 1. This and other screen images in this document were created under Windows XP; if you use another version of Windows (or, of course, another computing platform), then the appearance of the screen may differ.[4]

The *R Commander* and *R Console* windows float freely on the desktop. You will normally use the menus and dialog boxes of the R Commander to read, manipulate, and analyze data.

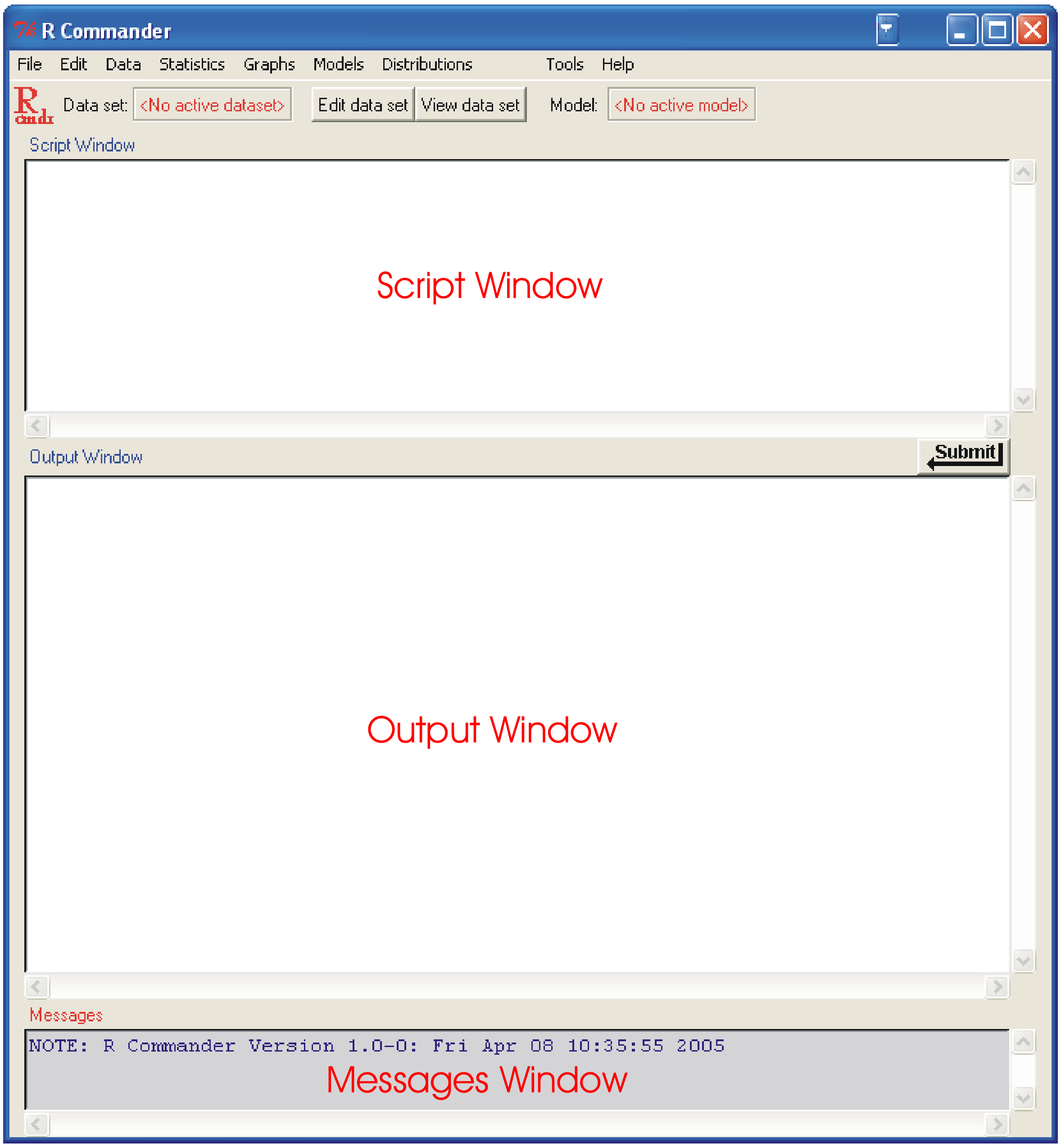
- R commands generated by the R Commander GUI appear in the upper text window (labelled *Script Window*) within the main *R Commander* window. You can also type R commands directly into the script window or at the > (greater-than) prompt in the *R Console*; the main purpose of the R Commander, however, is to avoid having to type commands.

[3]The Windows version of R is normally run from a multiple-document interface (MDI), which contains the *R Console* window, *Graphical Device* windows created during the session, and any other windows related to the R process. In contrast, under the single-document interface (SDI), the *R Console* and *Graphical Device* windows are not contained within a master window. There are several ways to run R in SDI mode—for example, by editing the `Rconsole` file in R's `etc` subdirectory, or by adding `--sdi` to the *Target* field in the *Shortcut* tab of the R desktop icon's *Properties*. This limitation of the **Rcmdr** package is inherited from the **tcltk** package, on which **Rcmdr** depends.

[4]The **Rcmdr** requires some packages in addition to several of the "recommended" packages that are normally distributed with R, and loads these packages at startup. **Rcmdr**, the required packages, and many other contributed packages are available for download from CRAN at `http://CRAN.R-project.org/`.

If these packages are not installed, the **Rcmdr** will offer to install them from the Internet or from local files (e.g., on a CD/ROM). If you install the **Rcmdr** package via the Windows "R GUI," the packages on which the **Rcmdr** depends should be installed automatically. More generally, you can install the **Rcmdr** package and all of the packages on which it depends via the `install.packages` function, setting the argument `dependencies = TRUE`.

Thanks to Dirk Eddelbuettel, Debian Linux users need only issue the command `$ apt-get install r-cran-rcmdr` to install the **Rcmdr** package along with all of the packages that it requires. In any event, building and installing the **Rcmdr** package on Linux systems is typically straightforward. The task can be more formidible under OS/X on Macintosh systems, since the **tcltk** package on which the **Rcmdr** depends requires that Tcl/Tk be installed and that R is running under X-Windows. Detailed installation instructions for Macintosh (and other) users are available at `http://socserv.socsci.mcmaster.ca/jfox/Misc/Rcmdr/installation-notes.html`.



```
NOTE: R Commander Version 1.0-0: Fri Apr 08 10:35:55 2005
```

Figure 1: The *R Commander* window at start-up, showing the *Script*, *Output*, and *Messages* sub-windows.

- Printed output appears by default in the second text window (labelled *Output Window*).
- The lower, gray window (labelled *Messages Window*) displays error messages, warnings, and some other information ("notes"), such as the start-up message in Figure 1.
- When you create graphs, these will appear in a separate *Graphics Device* window, outside of the main *R Commander* window.

There are several menus along the top of the *R Commander* window:

File Menu items for loading and saving script files; for saving output and the R workspace; and for exiting.

Edit Menu items (*Cut*, *Copy*, *Paste*, etc.) for editing the contents of the script and output windows. Right clicking in the script or output window also brings up an edit "context" menu.

Data Submenus containing menu items for reading and manipulating data.

Statistics Submenus containing menu items for a variety of basic statistical analyses.

Graphs Menu items for creating simple statistical graphs.

Models Menu items and submenus for obtaining numerical summaries, confidence intervals, hypothesis tests, diagnostics, and graphs for a statistical model, and for adding diagnostic quantities, such as residuals, to the data set.

Distributions Probabilities, quantiles, and graphs of standard statistical distributions (to be used, for example, as a substitute for statistical tables).

Tools Menu items for loading R packages unrelated to the **Rcmdr** package (e.g., to access data saved in another package), and for setting some options.

Help Menu items to obtain information about the R Commander (including an introductory manual derived from this paper). As well, each R Commander dialog box has a *Help* button (see below).

The complete menu "tree" for the R Commander (version 1.0-0) is shown below. Most menu items lead to dialog boxes, as illustrated later in this paper. Menu items are inactive ("grayed out") if they are inapplicable to the current context.

- File
  - Open script file
  - Save script
  - Save script as
  - Save output
  - Save output as
  - Save R workspace
  - Save R workspace as
  - Exit
    - from Commander
    - from Commander and R

- Edit
  - Clear Window
  - Cut
  - Copy
  - Paste
  - Delete
  - Find
  - Select all

- Data
  - New data set
  - Import data
    - from text file
    - from SPSS data set
    - from Minitab data set
    - from STATA data set
  - Data in packages
    - List data sets in packages
    - Read data set from attached package
    - Active data set
      - Select active data set
      - Help on active data set (if available)
      - Variables in active data set
      - Set case names
      - Subset active data set
      - Remove cases with missing data
      - Export active data set
  - Manage variables in active data set
    - Recode variable
    - Compute new variable
    - Standardize variables
    - Convert numeric variable to factor
    - Bin numeric variable
    - Reorder factor levels
    - Define contrasts for a factor
    - Rename variables
    - Delete variables from data set

- Statistics
  - Summaries
    - Active data set
    - Numerical summaries
    - Frequency distribution
    - Table of statistics
    - Correlation matrix
  - Contingency Tables
    - Two-way table
    - Multi-way table
    - Enter and analyze two-way table
  - Means
    - Single sample t-test
    - Independent-samples t-test
    - Paired t-test
    - One-way ANOVA
    - Multi-way ANOVA
  - Proportions
    - Single-sample proportion test
    - Two-sample proportions test
  - Variances
    - Two-variances F-test
    - Bartlett's test
    - Levene's test
  - Nonparametric tests
    - Two-sample Wilcoxon test
    - Paired-samples Wilcoxon test
    - Kruskal-Wallis test
  - Dimensional analysis
    - Scale reliability
    - Principal-components analysis
    - Factor analysis
    - Cluster analysis
      - k-means cluster analysis
      - Hierarchical cluster analysis
      - Summarize hierarchical clustering
      - Add hierarchical clustering to data set

- Fit models
  - Linear regression
  - Linear model
  - Generalized linear model
  - Multinomial logit model
  - Proportional-odds logit model

- Graphs
  - Index plot
  - Histogram
  - Stem-and-leaf display
  - Boxplot
  - Quantile-comparison plot
  - Scatterplot
  - Scatterplot matrix
  - 3D scatterplot
  - Line graph
  - Plot of means
  - Bar graph
  - Pie chart
  - Save graph to file
    - as bitmap
    - as PDF/Postscript/EPS
    - 3D RGL graph

- Models
  - Select active model
  - Summarize model
  - Add observation statistics to data
  - Confidence intervals
  - Hypothesis tests
    - ANOVA table
    - Compare two models
    - Linear hypothesis
  - Numerical diagnostics
    - Variance-inflation factors
    - Breusch-Pagan test for heteroscedasticity
    - Durbin-Watson test for autocorrelation
    - RESET test for nonlinearity
    - Bonferroni outlier test
  - Graphs
    - Basic diagnostic plots
    - Residual quantile-comparison plot
    - Component+residual plots
    - Added-variable plots
    - Influence plot
    - Effect plots

- Distributions
  - Normal distribution
    - Normal quantiles
    - Normal probabilities
    - Plot normal distribution
  - t distribution
    - t quantiles
    - t probabilities
    - Plot t distribution
  - Chi-squared distribution
    - Chi-squared quantiles
    - Chi-squared probabilities
    - Plot chi-squared distribution
  - F distribution
    - F quantiles
    - F probabilities
    - Plot F distribution
  - Binomial distribution
    - Binomial quantiles
    - Binomial tail probabilities
    - Binomial probabilities
    - Plot binomial distribution
  - Poisson distribution
    - Poisson probabilities
    - Plot Poisson distribution

- Tools
  - Load package(s)
  - Options

- Help
  - Commander help
  - About Rcmdr
  - Introduction to the R Commander
  - Help on active data set (if available)

The R Commander interface includes a few elements in addition to the menus and dialogs:

- Below the menus is a "toolbar" with a row of buttons.
  - The left-most (flat) button shows the name of the active data set. Initially there is no active data set. If you press this button, you will be able to choose among data sets currently in memory (if there is more than one). Most of the menus and dialogs in the R Commander reference the active data set. (The *File*, *Edit*, and *Distributions* menus are exceptions.)
  - Two buttons allow you to open the R data editor to modify the active data set or a viewer to examine it. The data-set viewer can remain open while other operations are performed.[5]
  - A flat button indicates the name of the active statistical model—a linear model (such as a linear-regression model), a generalized linear model, a multinomial logit model, or a proportional-odds model.[6] Initially there is no active model. If there is more than one model in memory, you can choose among them by pressing the button.
- Immediately below the toolbar is the script window (so labelled), a large scrollable text window. As mentioned, commands generated by the GUI are automatically copied into this window. You can edit the text in the script window or even type your own R commands into the window. Pressing the *Submit* button, which is at the right below the script window (or, alternatively, the key combination *Ctrl-r*, for "run"), causes the line containing the cursor to be submitted (or resubmitted) for execution. If several lines are selected (e.g., by left-clicking and dragging the mouse over them), then pressing *Submit* will cause all of them to be executed. Commands entered into the script window can extend over more than one line, but if they do, lines after the first must be indented with one or more spaces or tabs.
- Below the script window is a large scrollable and editable text window for output. Commands echoed to this window appear in red, output in dark blue (as in the *R Console*).
- At the bottom is a small gray text window for messages. Error messages are displayed in red text, warnings in green, and other messages in dark blue. Errors and warnings also provide an audible cue by ringing a bell. Messages are cleared at the next operation, but a 'note' does not clear an error message or a warning.

Once you have loaded the **Rcmdr** package, you can minimize the *R Console*. The *R Commander* window can also be resized or maximized in the normal manner. If you resize the

[5] The data viewer, provided by the `showData` function from David Firth's **relimp** package, can be slow for data sets with large numbers of variables. When the number of variables exceeds a threshold (initially set to 100), the R data editor is used instead to display the data set. To use the data editor regardless of the number of variables, set the threshold to 0. See the R Commander help file for details. A disadvantage of using the data editor to display the current data set is that the editor window cannot continue to be displayed while other operations are performed.

[6] Users can provide additional classes of statistical models by adding the necessary dialog boxes and menu items, and editing the `model-classes.txt` file in R's `etc` directory.

R Commander, the width of subsequent R output is automatically adjusted to fit the output window.

The R Commander is highly configurable: I have described the default configuration here. Changes to the configuration can be made via the *Tools* ⟶ *Options...* menu, or—much more extensively—by setting options in R. [A menu item that terminates in ellipses (i.e., three dots, ...) leads to a dialog box, which is a standard GUI convention. In this document, ⟶ represents selecting a menu item or submenu from a menu.] See the **Rcmdr** help files for details.

2.2. Data input

Most of the procedures in the R Commander assume that there is an active data set.[7] If there are several data sets in memory, you can choose among them, but only one is active. When the R Commander starts up, there is no active data set.

The R Commander provides several ways to get data into R:

- You can enter data directly via *Data* ⟶ *New data set...*. This is a reasonable choice for a very small data set.
- You can import data from a plain-text (ASCII) file or from another statistical package (Minitab, SPSS, or Stata).
- You can read a data set that is included in an R package, either typing the name of the data set (if you know it), or selecting the data set in a dialog box.

Reading data from a text file

For example, consider the data file `Nations.txt`.[8] The first few lines of the file are as follows:

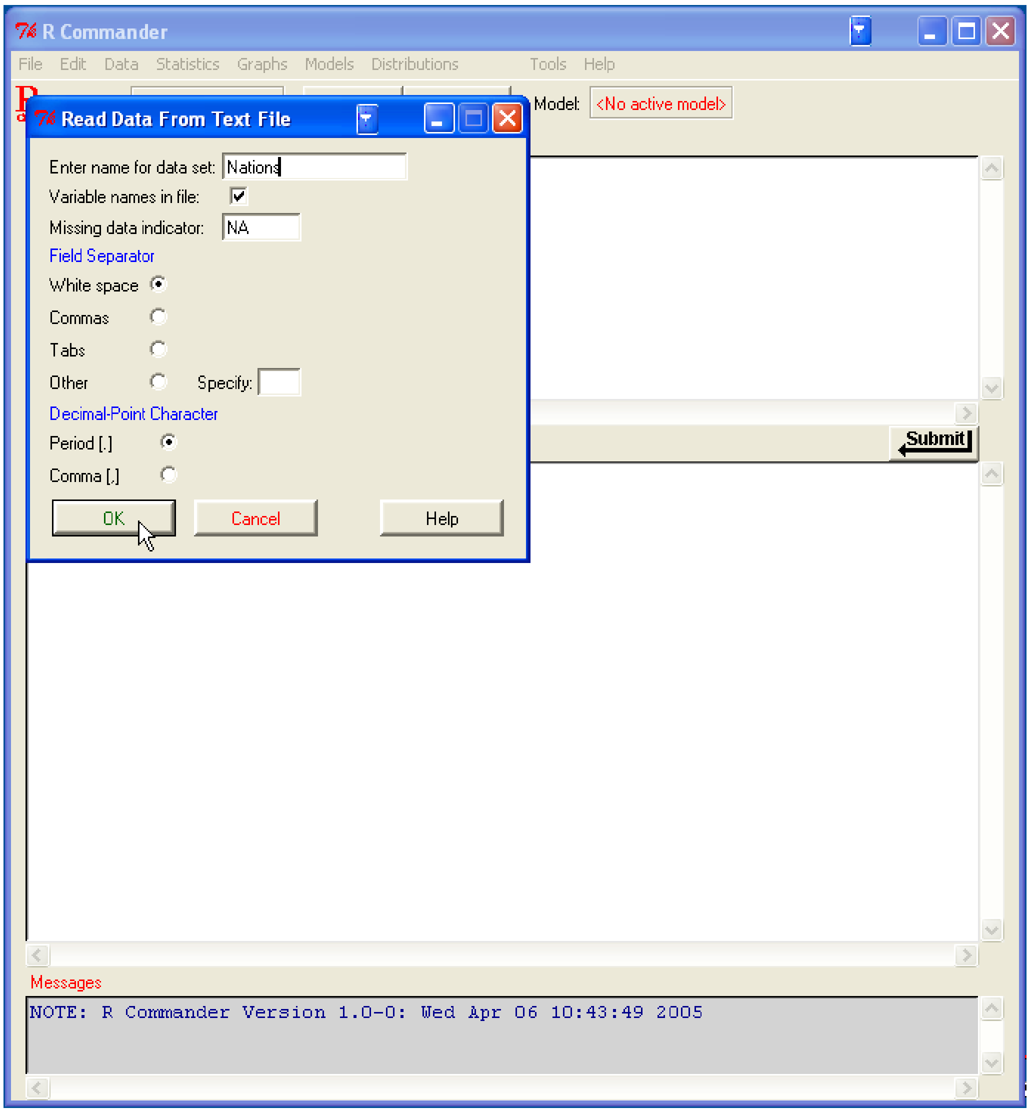
```
  TFR contraception  infant.mortality  GDP region
Afghanistan                 6.90    NA  154  2848   Asia
Albania                     2.60    NA   32   863   Europe
Algeria                     3.81    52   44  1531   Africa
American-Samoa              NA      NA   11  NA     Oceania
Andorra                     NA      NA  NA   NA     Europe
Angola                      6.69    NA  124   355   Africa
Antigua                     NA      53   24  6966   Americas
Argentina                   2.62    NA   22  8055   Americas
Armenia                     1.70    22   25   354   Europe
Australia                   1.89    76    6 20046   Oceania
. . .
```

[7]Procedures selected under via the *Distributions* menu are exceptions, as is *Enter and analyze two-way table...* under the *Statistics* ⟶ *Contingency tables* menu.

[8]This file resides in the `etc` subdirectory of the **Rcmdr** package.

- The first line of the file contains variable names: `TFR` (the total fertility rate, expressed as number of children per woman), `contraception` (the rate of contraceptive use among married women, in percent), `infant.mortality` (the infant-mortality rate per 1000 live births), `GDP` (gross domestic product per capita, in U.S. dollars), and `region`.

- Subsequent lines contain the data values themselves, one line per country. The data values are separated by "white space"—one or more blanks or tabs. Although it is helpful to make the data values line up vertically, it is not necessary to do so. Notice that the data lines begin with the country names. Because we want these to be the "row names" for the data set, there is no corresponding variable name: That is, there are five variable names but six data values on each line. When this happens, R will interpret the first value on each line as the row name.



Enter name for data set: Nations
Variable names in file: ✓
Missing data indicator: NA
White space [✓]
Commas
Tabs
Other Specify:
Period [.] [✓]
Comma [,]

```
NOTE: R Commander Version 1.0-0: Wed Apr 06 10:43:49 2005
```

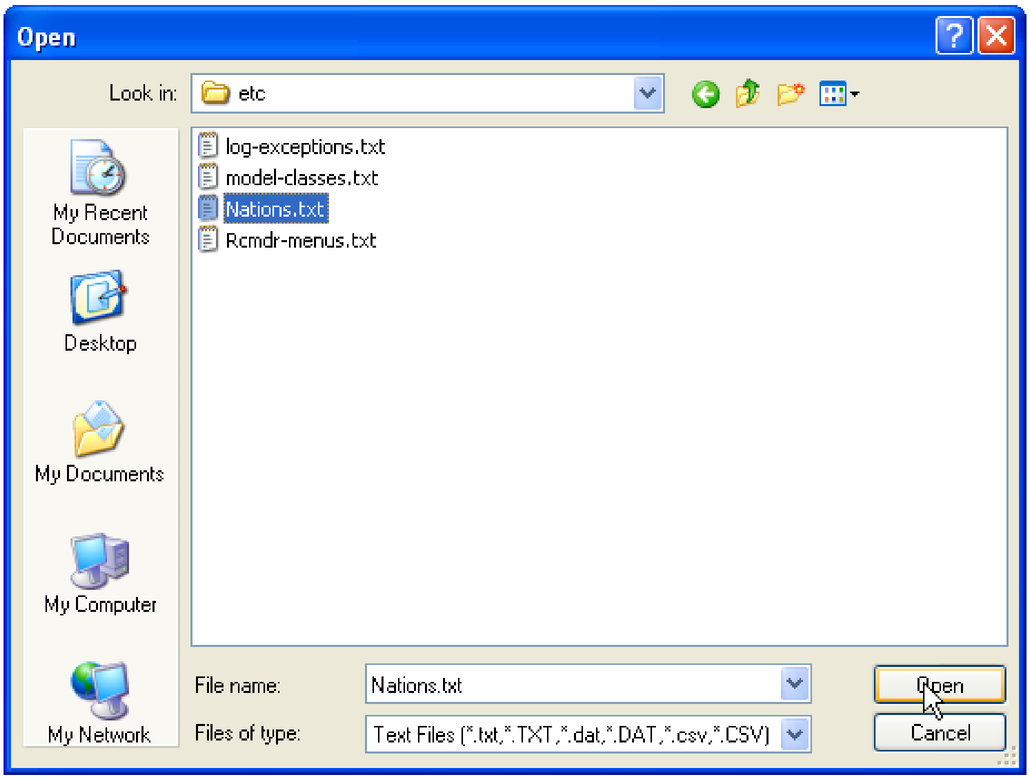
Figure 2: Reading data from a text file.

- Some of the data values are missing. In R, it is most convenient to use `NA` (representing "not available") to encode missing data, as I have done here.
- The variables `TFR`, `contraception`, `infant.mortality`, and `GDP` are numeric (quantitative) variables; in contrast, `region` contains region names. When the data are read, R will treat `region` as a "factor"—that is, as a categorical variable. In most contexts, the R Commander distinguishes between numerical variables and factors. The categories of a factor are termed its "levels."

To read the data file into R, select *Data* ⟶ *Import data* ⟶ *from text file...* from the *R Commander* menus. This operation brings up a *Read Data >From Text File* dialog, as shown in Figure 2. The default name of the data set is `Dataset`. I have changed the name to `Nations`.

Valid R names begin with an upper- or lower-case letter (or a period, `.`) and consist entirely of letters, periods, underscores (`_`), and numerals (i.e., 0–9); in particular, do not include any embedded blanks in a data-set name. You should also know that R is case-sensitive, and so, for example, `nations`, `Nations`, and `NATIONS` are distinguished, and could be used to represent different data sets.

Clicking the *OK* button in the *Read Data From Text File* dialog brings up an *Open* file dialog, shown in Figure 3. Here I navigated to the file `Nations.txt`. Clicking the *Open* button in the dialog will cause the data file to be read. Once the data file is read, it becomes the active data



log-exceptions.txt
model-classes.txt
Nations.txt
Rcmdr-menus.txt

File name: Nations.txt

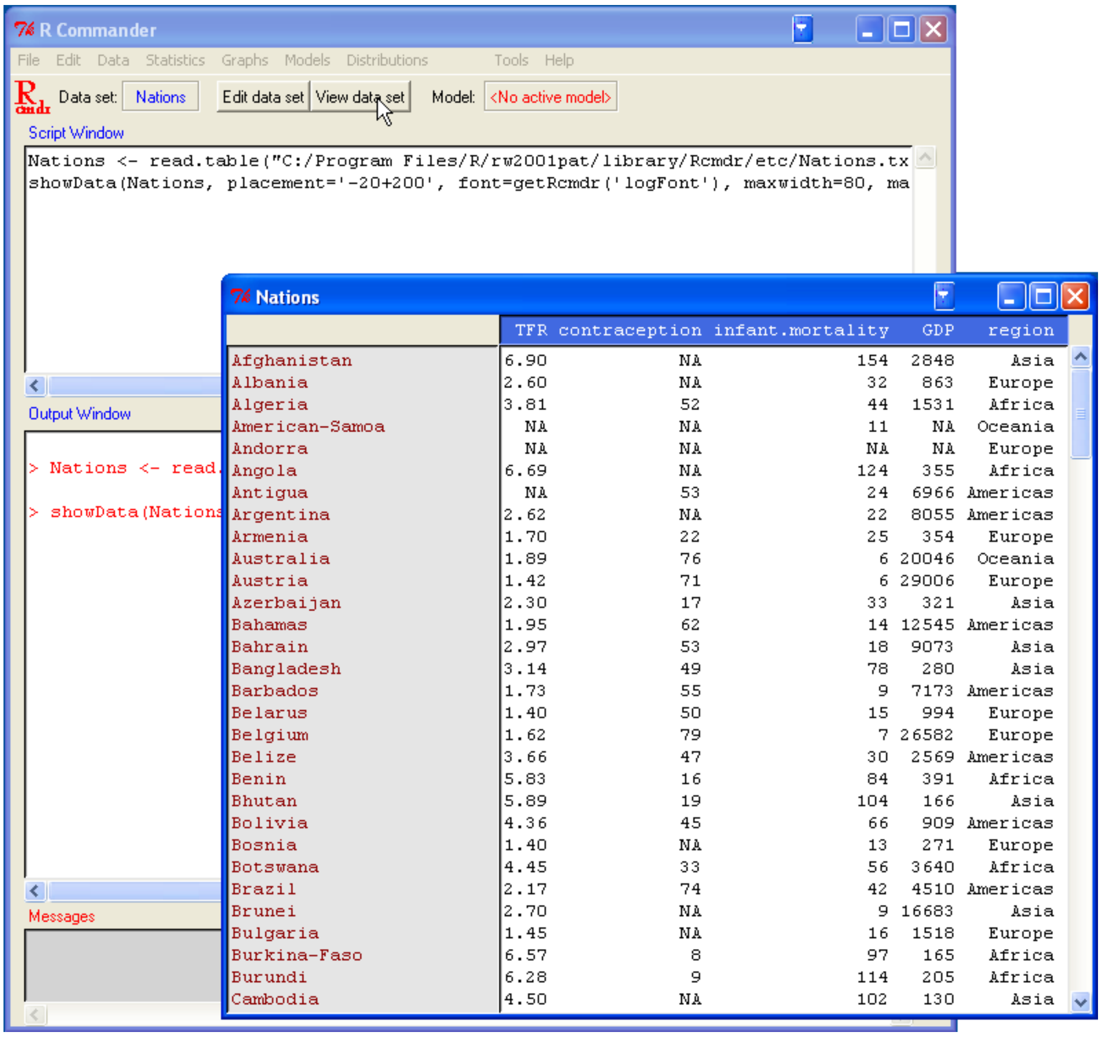
Files of type: Text Files (*.txt,*.TXT,*.dat,*.DAT,*.csv,*.CSV)

Figure 3: Open-file dialog for reading a text data file.

set in the R Commander. As a consequence, in Figure 4, the name of the data set appears in the data set button near the top left of the *R Commander* window.

I clicked the *View data set* button to bring up the data viewer window, also shown in Figure 4. Notice that the commands to read and view the `Nations` data set (the R `read.table` and `showData` commands) appear, partially obscured by the display of the data set, in the script and output windows. When the data set is read and becomes the active data set, a note appears in the messages window (and this is erased when the subsequent `showData` command is executed).

The `read.table` command creates an R "data frame," which is an object containing a rectangular cases-by-variables data set: The rows of the data set represent cases or observations and the columns represent variables. Data sets in the R Commander are R data frames.



```
Nations <- read.table("C:/Program Files/R/rw2001pat/library/Rcmdr/etc/Nations.tx
showData(Nations, placement='-20+200', font=getRcmdr('logFont'), maxwidth=80, ma
```

```
> Nations <- read.
> showData(Nations
```

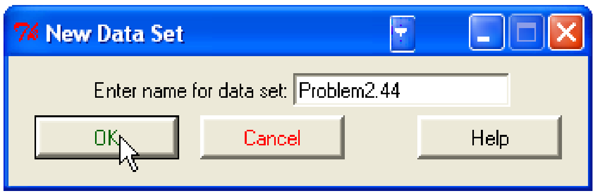
	TFR	contraception	infant.mortality	GDP	region
Afghanistan	6.90	NA	154	2848	Asia
Albania	2.60	NA	32	863	Europe
Algeria	3.81	52	44	1531	Africa
American-Samoa	NA	NA	11	NA	Oceania
Andorra	NA	NA	NA	NA	Europe
Angola	6.69	NA	124	355	Africa
Antigua	NA	53	24	6966	Americas
Argentina	2.62	NA	22	8055	Americas
Armenia	1.70	22	25	354	Europe
Australia	1.89	76	6	20046	Oceania
Austria	1.42	71	6	29006	Europe
Azerbaijan	2.30	17	33	321	Asia
Bahamas	1.95	62	14	12545	Americas
Bahrain	2.97	53	18	9073	Asia
Bangladesh	3.14	49	78	280	Asia
Barbados	1.73	55	9	7173	Americas
Belarus	1.40	50	15	994	Europe
Belgium	1.62	79	7	26582	Europe
Belize	3.66	47	30	2569	Americas
Benin	5.83	16	84	391	Africa
Bhutan	5.89	19	104	166	Asia
Bolivia	4.36	45	66	909	Americas
Bosnia	1.40	NA	13	271	Europe
Botswana	4.45	33	56	3640	Africa
Brazil	2.17	74	42	4510	Americas
Brunei	2.70	NA	9	16683	Asia
Bulgaria	1.45	NA	16	1518	Europe
Burkina-Faso	6.57	8	97	165	Africa
Burundi	6.28	9	114	205	Africa
Cambodia	4.50	NA	102	130	Asia

Figure 4: Displaying the active data set.

Entering data directly

To enter data directly into the R spreadsheet-like data editor you can proceed as follows. As an example, I use a very small data set from Problem 2.44 in Moore (2000):

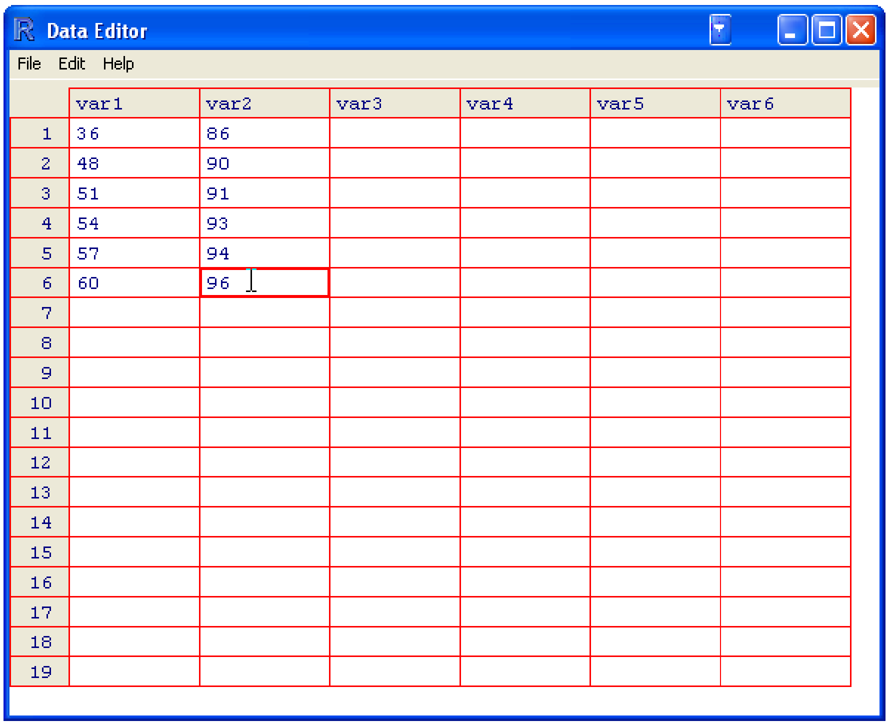
- Select *Data* ⟶ *New data set...* from the *R Commander* menus. Optionally enter a name for the data set (such as `Problem2.44`) in the resulting dialog box, as shown in Figure 5, and click the *OK* button. (Remember that R names cannot include intervening blanks.) This will bring up a *Data Editor* window with an empty data set.



Enter name for data set: Problem2.44

Figure 5: Defining a new data set.

- Enter the data from the problem into the first two columns of the data editor. You can move from one cell to another by using the arrow keys on your keyboard, by tabbing, by pressing the *Enter* key, or by pointing with the mouse and left-clicking. When you are finished entering the data, the window should look like Figure 6.



	var1	var2	var3	var4	var5	var6
1	36	86				
2	48	90				
3	51	91				
4	54	93				
5	57	94				
6	60	96				
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						

Figure 6: Data editor after the data are entered.

- Next, click on the name `var1` above the first column. This will bring up a *Variable editor* dialog box, as in Figure 7.

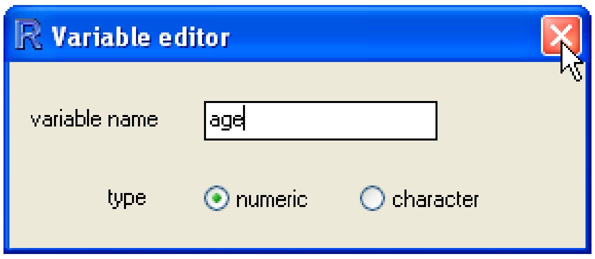
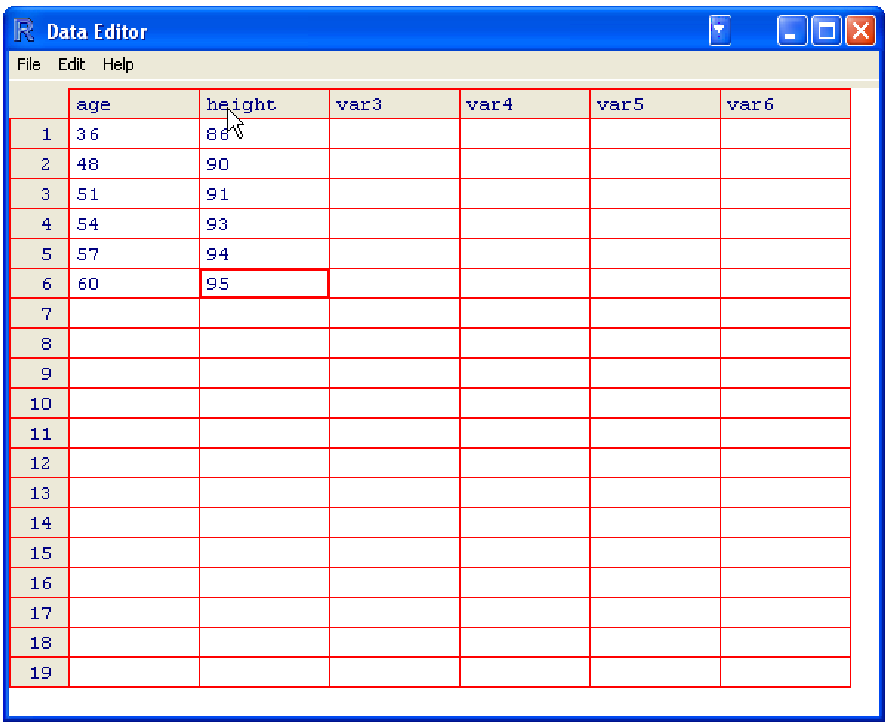


Figure 7: Dialog box for changing the name of a variable in the data editor.

- Type the variable name `age` in the box, just as I have, and click the X button at the upper-right corner of the *Variable editor* window, or press the *Enter* key, to close the window. Repeat this procedure to name the second column `height`. The *Data Editor* should now look like Figure 8.



	age	height	var3	var4	var5	var6
1	36	86				
2	48	90				
3	51	91				
4	54	93				
5	57	94				
6	60	95				
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						

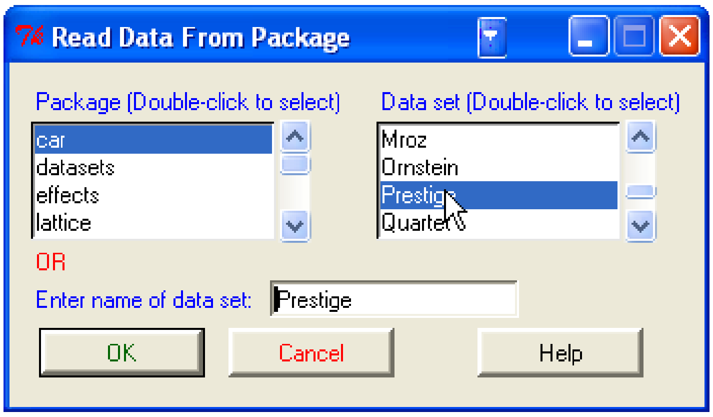
Figure 8: The *Data Editor* window after both variable names have been changed.

- Select *File* ⟶ *Close* from the *Data Editor* menus or click the X at the upper-right of the *Data Editor* window.[9] The data set that you entered is now the active data set in the R Commander.

[9]Saving the data by selecting *File* → *Close* or by simply closing the editor window is not a standard GUI convention, but this is how the R data editor behaves.

Reading data from a package

Many R packages include data. Data sets in packages can be listed in a pop-up window via *Data* ⟶ *Data in packages* ⟶ *List data sets in packages*, and can be read into the R Commander via *Data* ⟶ *Data in packages* ⟶ *Read data set from an attached package...* .[10] The resulting dialog box is shown in Figure 9, where I have selected the data set `Prestige` in the **car** package. If you know the name of a data set in a package then you can enter its name directly; otherwise double-clicking on the name of a package displays its data sets in the right list box; and double-clicking on a data set name copies the name to the data-set entry field in the dialog.[11] You can attach additional R packages by *Tools* ⟶ *Load packages*.



Enter name of data set: Prestige

Figure 9: Reading data from an attached package.

2.3. Creating numerical summaries and graphs

Once there is an active data set, you can use the *R Commander* menus to produce a variety of numerical summaries and graphs. I will describe just a few basic examples here. A good GUI should be largely self-explanatory: I hope that once you see how the R Commander works, you will have little trouble using it, assisted perhaps by the on-line help files.

In the examples below, I assume that the active data set is the `Nations` data set, read from a text file in the previous section. If you typed in the five-observation data set from Moore (2000), or read in the `Prestige` data set from the **car** package, following the procedures described in the previous section, then one of these is the active data set. Recall that you can change the active data set by clicking on the flat button with the active data set's name near the top left of the *R Commander* window, selecting from among a list of data sets currently resident in memory.

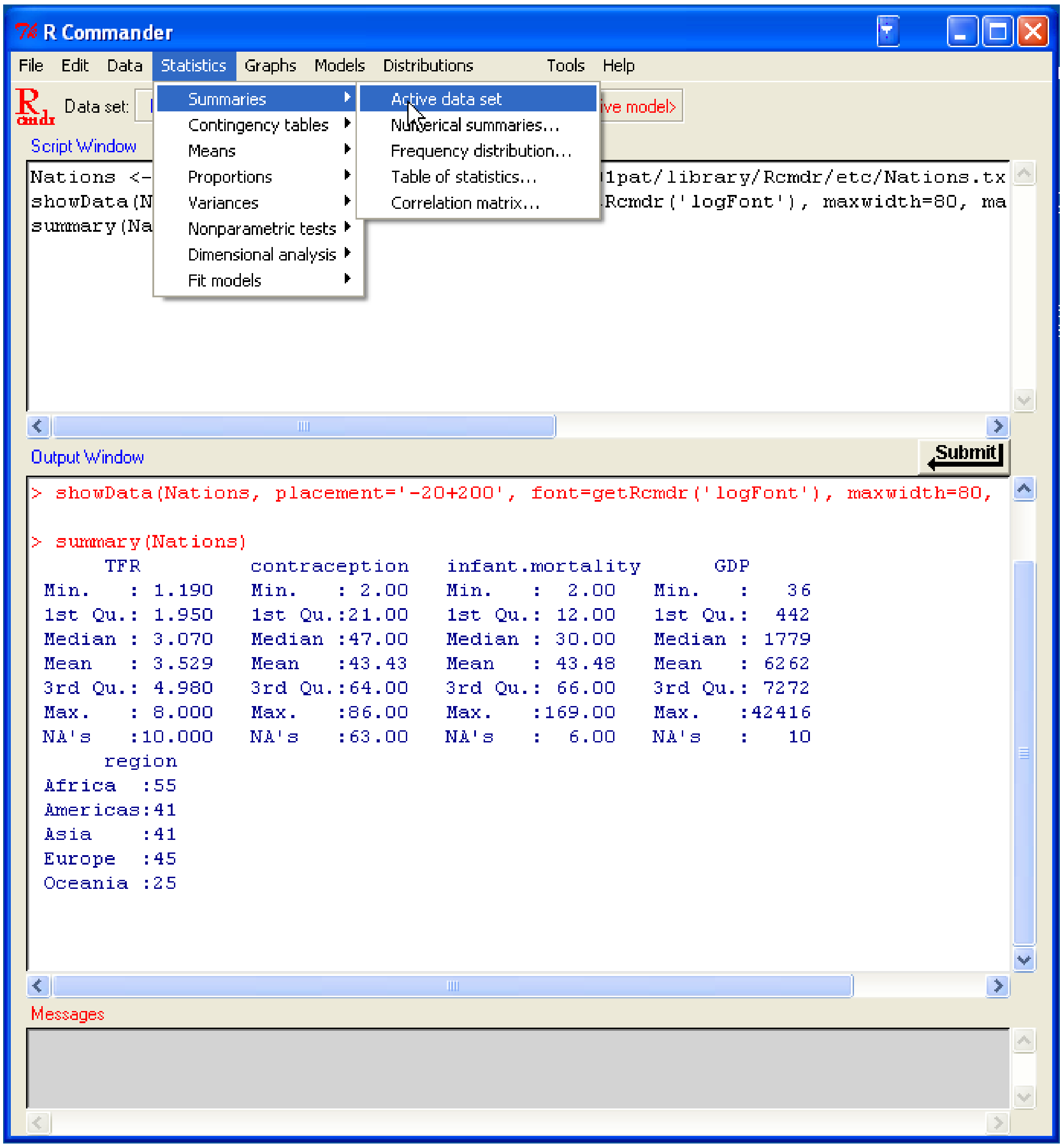
Selecting *Statistics* ⟶ *Summaries* ⟶ *Active data set* produces the results shown in Figure 10. For each numerical variable in the data set (`TFR`, `contraception`, `infant.mortality`, and `GDP`), R reports the minimum and maximum values, the first and third quartiles, the median, and the mean, along with the number of missing values. For the categorical variable

[10]Not all data in packages are data frames, but only data frames are suitable for use in the R Commander. If you try to read data that are not a data frame, an error message will appear in the messages window.

[11]In general in the R Commander, when it is necessary to copy an item from a list box to another location in a dialog, a double-click is required.

`region`, we get the number of observations at each level of the factor. Had the data set included more than ten variables, the R Commander would have asked us whether we really want to proceed—potentially protecting us from producing unwanted voluminous output.

Similarly, selecting *Statistics* ⟶ *Summaries* ⟶ *Numerical summaries...* brings up the dialog box shown in Figure 11. Only numerical variables are shown in the variable list in this dialog; the factor `region` is missing, because it is not sensible to compute numerical summaries for a factor. Clicking on `infant.mortality`, and then clicking *OK*, produces the



```
Nations <-                          1pat/library/Rcmdr/etc/Nations.tx
showData(N                          Rcmdr('logFont'), maxwidth=80, ma
summary(Na
```

```
> showData(Nations, placement='-20+200', font=getRcmdr('logFont'), maxwidth=80,

> summary(Nations)
      TFR          contraception   infant.mortality      GDP
 Min.   : 1.190   Min.   : 2.00   Min.   :  2.00   Min.   :   36
 1st Qu.: 1.950   1st Qu.:21.00   1st Qu.: 12.00   1st Qu.:  442
 Median : 3.070   Median :47.00   Median : 30.00   Median : 1779
 Mean   : 3.529   Mean   :43.43   Mean   : 43.48   Mean   : 6262
 3rd Qu.: 4.980   3rd Qu.:64.00   3rd Qu.: 66.00   3rd Qu.: 7272
 Max.   : 8.000   Max.   :86.00   Max.   :169.00   Max.   :42416
 NA's   :10.000   NA's   :63.00   NA's   :  6.00   NA's   :   10
      region
 Africa  :55
 Americas:41
 Asia    :41
 Europe  :45
 Oceania :25
```

Figure 10: Getting variable summaries for the active data set.

following output (in the output window):[12]

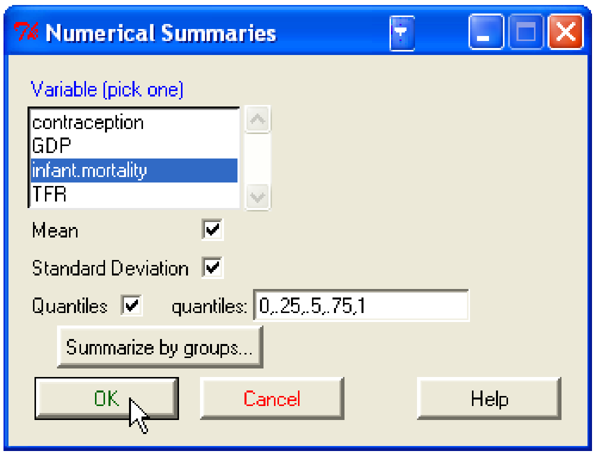
```
> mean(Nations$infant.mortality, na.rm=TRUE)
[1] 43.47761

> sd(Nations$infant.mortality, na.rm=TRUE)
[1] 38.75604

> quantile(Nations$infant.mortality, c( 0,.25,.5,.75,1 ), na.rm=TRUE)
  0%  25%  50%  75% 100% 
   2   12   30   66  169 
```

By default, the R commands that are executed print out the mean and standard deviation of the variable, along with quantiles (percentiles) corresponding to the minimum, the first quartile, the median, the third quartile, and the maximum.

As is typical of R Commander dialogs, the *Numerical Summaries* dialog box in Figure 11 includes *OK*, *Cancel*, and *Help* buttons. The *Help* button leads to a help page either for the dialog itself or (as here) for an R function that the dialog invokes.


Numerical Summaries

Variable (pick one)
contraception
GDP
infant.mortality
TFR

Mean ✓
Standard Deviation ✓
Quantiles ✓ quantiles: 0,.25,.5,.75,1

Figure 11: The *Numerical Summaries* dialog box.

The *Numerical Summaries* dialog box also makes provision for computing summaries within groups defined by the levels of a factor. Clicking on the *Summarize by groups...* button brings up the *Groups* dialog, as shown in Figure 12. Because there is only one factor in the `Nations` data set, only the variable `region` appears in the variable list; selecting this variable and clicking *OK* changes the *Summarize by groups...* button to *Summarize by region* (see Figure 13); clicking *OK* produces the following results (with most of the output suppressed for brevity):

[12]To select a single variable in a variable-list box, simply left-click on its name. In some contexts, you will have to select more than one variable. In these cases, the usual Windows conventions apply: Left-clicking on a variable selects it and de-selects any variables that have previously been selected; *Shift-left-click* extends the selection; and *Ctrl-left-click* toggles the selection for an individual variable.

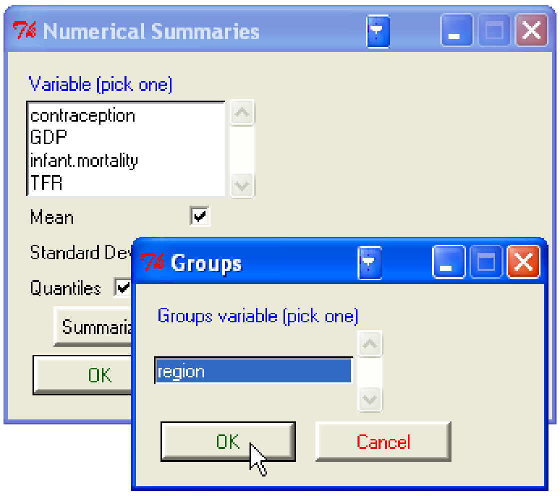


Figure 12: Selecting a grouping variable in the *Groups* dialog box.

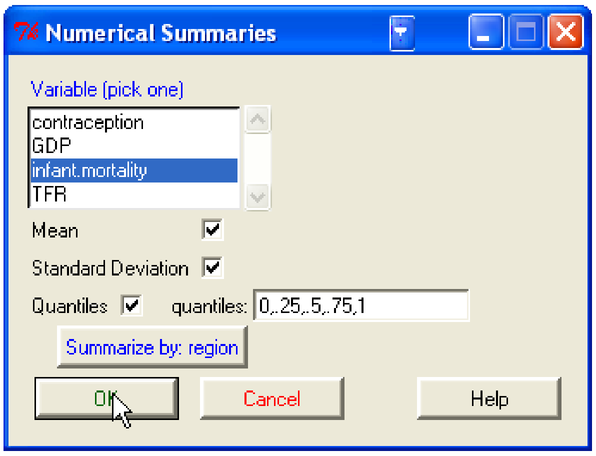


Figure 13: The *Numerical Summaries* dialog box after a grouping variable has been selected.

```
> by(Nations$infant.mortality, Nations$region, mean, na.rm=TRUE)
INDICES: Africa
[1] 85.27273
------------------------------------------------------------
INDICES: Americas
[1] 25.6
------------------------------------------------------------
INDICES: Asia
[1] 45.65854
------------------------------------------------------------
INDICES: Europe
[1] 11.85366
------------------------------------------------------------
INDICES: Oceania
[1] 27.79167

. . .
```

Several other R Commander dialogs allow you to select a grouping variable in this manner.

Making graphs with the R Commander is also straightforward. For example, selecting *Graphs* ⟶ *Histogram...* from the *R Commander* menus brings up the *Histogram* dialog box in Figure 14; and clicking on `infant.mortality` followed by *OK*, opens a *Graphics Device* window with the histogram shown in Figure 15.

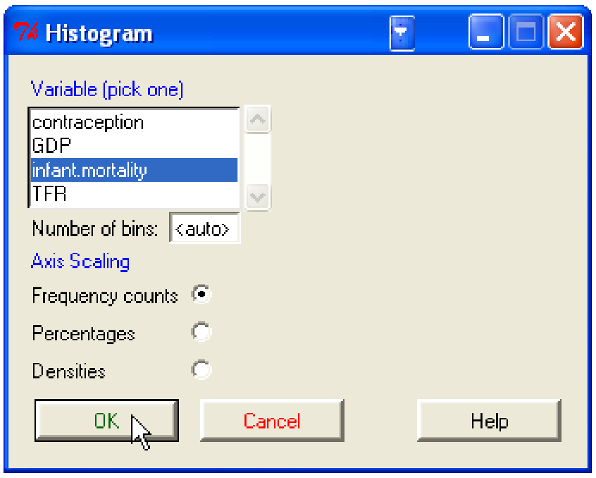


Figure 14: The *Histogram* dialog.

If you make several graphs in a session, then only the most recent normally appears in the *Graphics Device* window. You can recall previous graphs using the *Page Up* and *Page Down* keys on your keyboard.[13]

[13]At start-up, the R Commander turns on the graph history mechanism; this feature is available only in Windows systems. Dynamic three-dimensional scatterplots created by *Graphs* ⟶ *3D scatterplot...* appear in a special *RGL device* window; likewise, effect displays created for statistical models (Fox 2003) via *Models* ⟶ *Graphs* ⟶ *Effect plots* appear in individual graphics-device windows.

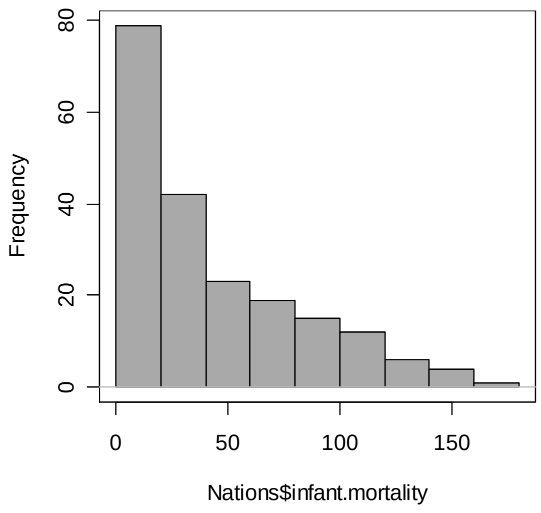


Figure 15: A graphics window containing the histogram for infant mortality.

2.4. Statistical models

Several kinds of statistical models can be fit in the R Commander using menu items under *Statistics* $\longrightarrow$ *Fit models*: linear models (by both *Linear regression* and *Linear model*), generalized linear models, multinomial logit models, and proportional-odds models, the latter two from Venables and Ripley's **nnet** and **MASS** packages, respectively (Venables and Ripley 2002). Although the resulting dialog boxes differ in certain details (for example, the generalized linear model dialog makes provision for selecting a distributional family and corresponding link function), they share a common general structure, as illustrated in the *Linear Model* dialog in Figure 16.[14]

In R (and in S generally), linear and linear-like statistical models are specified using a version of Wilkinson and Rogers's model-formula notation (Wilkinson and Rogers 1973). It is beyond the scope of this paper to describe model formulas in detail, but the following basic information may prove useful:[15] The left and right-hand sides of the model are separated by a tilde (`~`). The left hand side may be the name of the response variable (e.g., `prestige`) or an expression that evaluates to the response variable [e.g., `log(prestige)`]. On the right-hand side of the model, operators such as $+$ and $*$ have special meaning. For example, `+` adds a term to the model, while `*` can be used to include an interaction in the model along with all terms (such as main effects) that are marginal to the interaction. Parentheses may be used to group terms.

[14]An exception is the *Linear Regression* dialog in which the response variable and explanatory variables are simply selected by name from list boxes containing the numeric variables in the current data set. Although linear regression models may also be specified in the *Linear Model* dialog, the *Linear Regression* dialog avoids the explicit specification of a model formula and thus is more suited to a basic-statistics course.

[15]For more information on specifying models, see the *Introduction to R* manual that comes with R, which may be accessed from the *Help* menu in the *R Console*, or a general treatment of statistical modeling in S, such as Chambers and Hastie (1992), Fox (2002), or Venables and Ripley (2002).

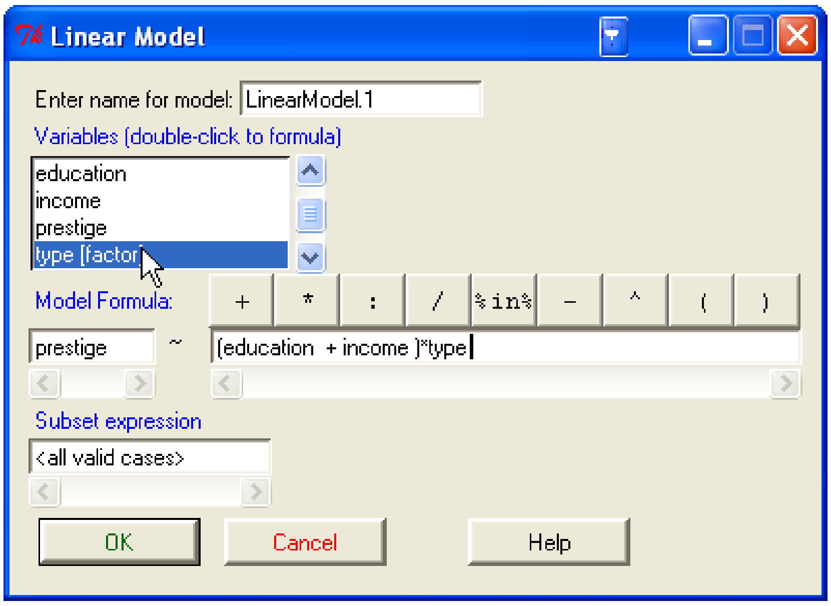


Figure 16: The *Linear Model* dialog box.

Thus, `(education + income)*type` specifies terms for `education`, `income`, `type`, and the interactions between `education` and `type` and between `income` and `type`. Contrasts (such as dummy regressors) are automatically created when a factor (such as `type` in Figure 16) is included on the right-hand side of a model formula.

- Double-clicking on a variable in the variable-list box copies it to the model formula—to the left-hand side of the formula, if it is empty, otherwise to the right-hand side (with a preceding + sign if the context requires it). Note that factors (categorical variables) are parenthetically labelled as such in the variable list.

- The row of buttons above the formula can be used to enter operators and parentheses into the right-hand size of the formula.

- You can also type directly into the formula fields, and indeed have to do so, for example, to put a term such as log(income) into the formula.

- The name of the model, here `LinearModel.1`, is automatically generated, but you can substitute any valid R name.

- You can type an R expression into the box labelled *Subset expression*; if supplied, this is passed to the `subset` argument of the `lm` function, and is used to fit the model to a subset of the observations in the data set. One form of subset expression is a logical expression that evaluates to `TRUE` or `FALSE` for each observation, such as `type != "prof"` (which would select all non-professional occupations from the `Prestige` data set).

Clicking the *OK* button produces the following output (in the output window), and makes `LinearModel.1` the active model, with its name displayed in the *Model* button:

```
> LinearModel.1 <- lm(prestige ~ (education  + income )*type , data=Prestige)

> summary(LinearModel.1)

Call:
lm(formula = prestige ~ (education + income) * type, data = Prestige)

Residuals:
    Min      1Q  Median      3Q     Max 
-13.462  -4.225   1.346   3.826  19.631 

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)    
(Intercept)              2.276e+00  7.057e+00   0.323   0.7478    
education                1.713e+00  9.572e-01   1.790   0.0769 .  
income                   3.522e-03  5.563e-04   6.332 9.62e-09 ***
type[T.prof]             1.535e+01  1.372e+01   1.119   0.2660    
type[T.wc]              -3.354e+01  1.765e+01  -1.900   0.0607 .  
education:type[T.prof]  1.388e+00  1.289e+00   1.077   0.2844    
education:type[T.wc]    4.291e+00  1.757e+00   2.442   0.0166 *  
income:type[T.prof]    -2.903e-03  5.989e-04  -4.847 5.28e-06 ***
income:type[T.wc]      -2.072e-03  8.940e-04  -2.318   0.0228 *  
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1 

Residual standard error: 6.318 on 89 degrees of freedom
Multiple R-Squared: 0.8747, Adjusted R-squared: 0.8634 
F-statistic: 77.64 on 8 and 89 DF,  p-value: < 2.2e-16 
```

Operations on the active model may be selected from the *Models* menu. For example, *Models* ⟶ *Hypothesis tests* ⟶ *Anova table* produces the following output:

```
> Anova(LinearModel.1)
Anova Table (Type II tests)

Response: prestige
               Sum Sq Df F value    Pr(>F)    
education      1068.0  1 26.7532 1.413e-06 ***
income         1131.9  1 28.3544 7.511e-07 ***
type            591.2  2  7.4044  0.001060 ** 
education:type  238.4  2  2.9859  0.055574 .  
income:type     951.8  2 11.9210 2.588e-05 ***
Residuals      3552.9 89                      
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1 
```

2.5. Odds and ends

Saving and printing output

You can save text output directly from the *File* menu in the *R Commander*; likewise you can save or print a graph from the *File* menu in an R *Graphics Device* window.[16] It is generally more convenient, however, to collect the text output and graphs that you want to keep in a word-processor document. In this manner, you can intersperse R output with your typed notes and explanations.

Open a word processor such as Word, or even Windows WordPad. To copy text from the output window, block the text with the mouse, select *Copy* from the *Edit* menu (or press the key combination *Ctrl-c*, or right-click in the window and select *Copy* from the context menu), and then paste the text into the word-processor window via *Edit* $\longrightarrow$ *Paste* (or *Ctrl-v*), as you would for any Windows application. One point worth mentioning is that you should use a mono-spaced ("`typewriter`") font, such as *Courier New*, for text output from R; otherwise the output will not line up neatly.

Likewise, to copy a graph, select *File* $\longrightarrow$ *Copy to the clipboard* $\longrightarrow$ *as a Metafile* from the R *Graphics Device* menus; then paste the graph into the word-processor document via *Edit* $\longrightarrow$ *Paste* (or *Ctrl-v*). Alternatively, you can use *Ctrl-w* to copy the graph from the R *Graphics Device*, or right-click on the graph to bring up a context menu, from which you can select *Copy as metafile*.[17] At the end of your R session, you can save or print the document that you have created, providing an annotated record of your work.

Alternative routes to saving text and graphical output may be found respectively under the R Commander *File* and *Graphs* $\longrightarrow$ *Save graph to file* menus.

Terminating the R Session

There are several ways to terminate your session. For example, you can select *File* $\longrightarrow$ *Exit* $\longrightarrow$ *From Commander and R* from the *R Commander* menus. You will be asked to confirm, and then asked whether you want to save the contents of the script and output windows. Likewise, you can select *File* $\longrightarrow$ *Exit* from the *R Console*; in this case, you will be asked whether you want to save the R workspace (i.e., the data that R keeps in memory); you would normally answer *No*: In my experience beginning students can be confused by objects carried over from one session to another in a saved workspace. The ability to save the workspace, and to maintain different saved workspaces for different projects, can, however, be helpful to more advanced users.

Entering commands in the script window

The script window provides a simple facility for editing, entering, and executing commands. Commands generated by the R Commander automatically appear in the script window, and you can type and edit commands in the window more or less as in any editor. The R Com-

[16]Most of the information in this subsection on saving and printing output is specific to the Windows operating system.

[17]As you will see when you examine these menus, you can save graphs in a variety of formats, and to files as well as to the clipboard. The procedure suggested here is straightforward, however, and generally results in high-quality graphs. Once again, this description applies to Windows systems.

mander does not provide a true "console" for R, however, and the script window has some limitations:

- Commands that extend over more than one line should have the second and subsequent lines indented by one or more spaces or tabs; all lines of a multiline command must be submitted simultaneously for execution.
- Commands that include an assignment arrow (`<-`) will not generate printed output, even if such output would normally appear had the command been entered in the *R Console* [the command `print(x <- 10)`, for example]. On the other hand, assignments made with the equals sign (`=`) produce printed output even when they normally would not (e.g., `x = 10`).
- Commands that produce normally invisible output will occasionally cause output to be printed in the output window. This behavior can be modified by editing the entries of the `log-exceptions.txt` file in the R Commander's `etc` directory.
- Blocks of commands enclosed by braces, i.e., `{}`, are not handled properly unless each command is terminated with a semicolon (`;`). This is poor R style, and implies that the script window is of limited use as a programming editor. For serious R programming, it would be preferable to use the script editor provided by the Windows version of R itself, or—even better—a programming editor.

3. Design and development of the R Commander

Prior to developing the R Commander, I had for several years wanted to use R in teaching basic statistics to social-science undergraduates, but from past experience I felt that the command-line interface to R would present an obstacle to many students. The software that I used in this course over the previous decade or so—first Minitab and then SPSS—was not software that I used in my own work. Moreover I did not feel that I could ask my students to purchase software for the class, which already requires them to buy a relatively expensive textbook and some other materials. Consequently statistical computing in the course was relegated to university computer labs. I expect that this is not an uncommon scenario, at least at universities that do not offer attractive site-licensing of statistical software to students.

I expected someone else with more experience in GUI development to produce a suitable GUI for R, but when nothing that I could use in my course materialized by the Spring of 2003, I decided to explore creating one myself. I looked initially at the facilities provided by the Windows version of R—for example, the `winMenu*` and `winDialog` functions—but quickly determined that these were inadequate for developing a broadly useful *statistical* GUI.[18] I experimented next with Visual Basic, and although this route to a statistical GUI for R

[18]The standard "RGui" console to R for Windows furnishes many useful "housekeeping" operations and amenities (such as installing and loading packages), but does not provide access to the statistical capabilities of R. The same can be said of the R consoles developed for other computing platforms. The `windlgs` package—a source package distributed with the Windows version of R—demonstrates the use of C-code for constructing statistical menus and dialogs under Windows. This route to a statistical GUI for R, which employs the GraphApp toolkit on which the RGui console is based, would be feasible—and, in principle, capable of building a cross-platform GUI—but it would also be relatively difficult.

appeared to be feasible, I decided against it for several reasons, the most important of which were the propriety nature of Visual Basic and my desire to produce a cross-platform solution.

I quickly gravitated towards Peter Dalgaard's **tcltk** package: The package is available for all of the major R platforms; it provides a serviceable, if not rich, set of widgets; and most importantly, the standard Windows version of R installs a basic Tcl/Tk system. The last point was key, in my view, because the principal target audience for a basic-statistics GUI consists in large majority of Windows users, many of whom have difficulty installing and configuring software. By using Tcl/Tk through the **tcltk** package, I was also able to provide a GUI as a standard R package, which developed into the **Rcmdr**. Installing the **Rcmdr** (and its dependencies) is simple, especially on Windows systems, and loading the package starts up the GUI.

Other, arguably more capable, GUI toolkits—such as GTK via the **RGtk** package (see `http://www.omegahat.org/RGtk/`)—appeared to create obstacles for Windows users. I believe that use of the **tcltk** package still provides the most convenient route to a GUI for Windows users, though I am also aware of several other R GUI projects in addition to the R Commander.[19] I look forward to these producing a better statistical GUI than the R Commander that is usable by relatively naive Windows users.

Using Tcl/Tk entailed several compromises, however: The standard widget set is limited; in particular, I was unable to employ drop-down lists, tabbed dialogs, and table widgets, which I would have preferred to use in certain contexts. For example, the data set viewer in the **Rcmdr** package—the `showData` function from the **relimp** package—would have been more naturally programmed using a table widget, as would the **Rcmdr** *Enter Two-Way Table* and *Test Linear Hypothesis* dialogs. Similarly, providing options on an *Options* tab would produce cleaner and more uniform dialog boxes. There are extended widget sets available for Tcl/Tk, but because these are not part of the standard installation of R for Windows, I reluctantly ruled out their use.[20]

Another limitation of Tcl/Tk is that while it is available on all of the major platforms that run R, its look and feel is non-standard on all of these platforms. Nevertheless, I have been able to tune the behavior of the R Commander GUI to be very similar to that of a standard Windows application.

Some problems remain: On the Macintosh (as mentioned), applications such as the R Commander that use the **tcltk** package must run under X-Windows and require software that is not installed on out-of-the-box OS/X systems; the appearance of the R Commander GUI is not as attractive on Linux systems as it is on Windows systems, although the cosmetics can be improved by carefully selecting fonts and font sizes (as supported by R Commander options); and there are some (if now greatly reduced) stability problems on Windows systems, stemming from the integration of the Tcl/Tk and R event loops.

The initial version of the **Rcmdr** package (numbered 0.5-0)[21], with perhaps half the content

[19]To elaborate slightly, at the time of writing, the web page for the **RGtk** package (downloaded on 31 August 2005, and dated 4 September 2003) states: "There is currently no version [of] this package for Windows available. I have compiled one and it works. However, I need to enhance the event loop integration." I understand from a reviewer of this paper that **RGtk** does work with Windows, but as far as I have been able to ascertain, Windows binaries are not available, and building the package for Windows is relatively complicated.

[20]This situation may change, however: Philippe Grosjean is working on an extension to the **tcltk** package that provides additional widgets (see `http://www.sciviews.org/SciViews-R/`).

[21]Early development of the project was done prior to creating a package.

of the current version, was completed in about a month, and somewhat later, in the Summer of 2003, was contributed to CRAN. The range of features supported by the R Commander grew gradually over the following two years, but a number of conventions established in this early version of the package persist:

- The interface uses standard menus, most of which lead to simple dialog boxes. As mentioned, the limited range of R-Commander dialog-box elements is the product of the restricted standard Tk widget set, but the simplicity and familiarity of the interface is deliberate. The object was to produce an interface that students would be able to learn and negotiate with little trouble. Though it is less extensive and less polished, the R Commander GUI is similar in many respects to other GUIs to command-oriented statistical software, such as SPSS (`http://www.spss.com/`) and Minitab (`http://www.minitab.com/`): The basic model of work-flow is procedural. This contrasts with statistical packages [such as JMP (`http://www.jmp.com/`) or Vista (`http://www.visualstats.org/`)] that are meant to be pedagogically innovative.

- The set of top-level menus in Version 0.5-0 was the same as the current one, except that a *Tools* menu was introduced much later. The R Commander menus were initially "hard-wired" in the package code, but were later made configurable via a text file. In other instances as well, features in the package were made more flexible and configurable. For example, the **Rcmdr** originally supported only linear and generalized linear models; now, the range of supported models has expanded and can be augmented by the user.[22]

- Typical R Commander dialog boxes have one or more scrollable variable-list boxes at the top; check boxes and radio buttons for selecting options below that; and *OK*, *Cancel*, and *Help* buttons at the bottom. Some dialog boxes have buttons that produce sub-dialogs displayed over the main dialog. I have tried to use this arrangement sparingly, and could have avoided it altogether were tabbed dialogs available in the Tk widget set supported under Windows by the **tcltk** package.

- Menus and dialog boxes generate R commands (whence the name, "R Commander") that are saved in a script window (originally called a "log"). These commands call basic R functions, functions in the "recommended" packages that are part of the standard R distribution, and—as necessary—functions in contributed packages available from CRAN. Although I tried to avoid it, in a few instances, I introduced additional statistical functionality to the **Rcmdr** package: for example, functions to compute alpha-reliability for composite scales and to compute partial-correlation matrices. These functions, summarized in Table 5 in the next section of the paper, are usable independently of the **Rcmdr** GUI. Generating commands to be executed was not the only route to go: Statistical computations could have been, at least partly, subsumed in the code for the **Rcmdr** package, and the details of the computations hidden from the user. To do so, however, would have wasted some of the effort put into developing the statistical capabilities of R, and would also have contradicted one of the goals of the R Commander project—to draw a visible connection between choices made in the GUI and R commands.

[22]See the next section for the composition of the menu-definition file and information on how to extend the **Rcmdr** package.

- Statistical analyses are performed on an active data set, which is a standard R data frame. An alternative would be to allow the user to select a data set in each dialog, with the selection defaulting to the previous one. This seems to me to offer no advantage over the current scheme. Another possibility would be to permit multiple data frames to be attached to the search path. This approach provides more flexibility in handling data, but I find that even more advanced students than those in introductory statistics classes have difficulty dealing with issues, such as objects masking each other, that arise from managing the search path.[23] For similar reasons, all variable creation (for example, by the *Recode* and *Compute* dialogs, and the computation of residuals or other "case statistics" for statistical models) takes place in the active data set; an alternative would have been to allow variables to be created in the global environment, but such an approach risks doing damage, creating conflicts, and generating potentially cryptic errors.

- Similarly, operations on statistical models via the *Models* menu are performed primarily on an active statistical model, which is kept synchronized with the active data set—when the active data set is changed, there is initially no active model, and when an active model is selected from among recognized model objects in memory, the active data set is changed to the data frame on which that model was fit. This procedure is a bit constraining for advanced users (who will, I believe, in any event prefer to specify commands directly), but it helps novices to keep things straight.

- Menus and dialog boxes produce R commands as text strings. The R Commander causes these commands to be parsed and evaluated in the global R environment. Having the commands available as text is convenient for entry into the script and output windows, but I am not entirely satisfied with this approach: In particular, building text commands can be awkward, and the code to do so hard to read. My early efforts to proceed with tools such as `eval`, `substitute`, and `expression` were not successful, however. Likewise, although it has successively been improved, the script window is much less than a true R console, something that I have been unable to provide in a platform-independent manner.

- The original R Commander had a toolbar below the menu bar with information fields displaying the names of the active data set and active statistical model; buttons for editing and viewing the active data set; and a check box for determining whether commands were echoed to the script window. Somewhat later, the data-set and statistical-model information fields morphed into buttons that could be used to select the active data set and model, the log window became the current script window, and the check box was removed. A button was provided to submit lines in the script window for re-execution.

- Initially, output was directed to the R console. Although this arrangement is retained as an option, an output window was introduced, which receives printed output by default.

- Error messages and warnings were initially printed in the R console. Later, such messages were intercepted and presented to the user in pop-up message windows. Currently,

[23]Until recently, the active data set in the R Commander was, by default, attached to the search path, but that procedure was unnecessary, and led to awkwardly repetitive attaches and detaches of data frames.

error messages and warnings (along with other messages) are directed to a messages window. The main *R Commander* window therefore has evolved from one, to two, and then to three text sub-windows. The script and output windows are editable.

Along the way, many changes were made "beneath the hood" to improve the performance and maintainability of the **Rcmdr** package. At one point, for example, the size of the **Rcmdr** code was reduced by nearly 40 percent by modularizing repetitive elements, primarily in dialog-box generating functions. Some of this modularization employs macro-like functions (Lumley 2001).[24] At present, functions that create **Rcmdr** dialog boxes consist mostly of calls to utility functions to initialize and close a dialog, and to construct common elements such as variable lists, sets of radio buttons and check boxes, and the *OK*, *Cancel*, and *Help* buttons at the bottom of the dialog box. This process is illustrated in the next section.

Similarly, the original **Rcmdr** saved a great deal of state information in global variables, such as the name of the active data set, the names of variables within the active data set, and various options. Currently, all of this state information is saved instead in a special environment—a much neater and less problematic solution (see the functions `getRcmdr` and `putRcmdr` in Table 1 below).

3.1. How well has the R Commander met its goals?

Ease of use Over the years, I have used a variety of statistical software in introductory-statistics courses—more, indeed, than I would care to enumerate. Although I do not have formal evidence about the relative usability of the R Commander in this context, I can report that in the two years that I have been using it, students appear to have virtually no trouble in completing course assignments requiring the software. I have also had positive feedback from other individuals who have used the **Rcmdr** package for statistical instruction. This experience compares favorably with the other statistical software that I have used in teaching.

Coverage The R Commander now is much more extensive than required for the basic statistics texts that I have examined, and can reasonably support most of a low-level course in applied regression analysis.

Cross-platform functionality My own experience with the **Rcmdr** package is primarily under Windows, where the software works quite well. As mentioned, I and others also have used it successfully under Linux. Installation and use under Macintosh OS/X is possible but more challenging at present. I have occasionally received reports of particular aspects of the software proving problematic on non-Windows systems, but these have been isolated—for example, to the 3D scatterplots dialog, which depends upon the **rgl** package.[25]

Extensiblity As described in the next section, extension of the **Rcmdr** package requires some programming and editing of configuration files, though not necessarily rebuilding

[24]My initial attempts to provide common Tk dialog-box elements—such as sets of *OK*, *Cancel*, and *Help* buttons, check-boxes, radio-buttons, etc.—via standard R functions failed because of scoping problems. Macro-like functions, which execute in the environment of the calling function, provided a solution.

[25]I understand that a new version of the **rgl** package should resolve stability issues on non-Windows platforms.

the package itself. This process is facilitated by utility functions for the construction of dialog boxes that the package exports, and by the ability to add to and modify the **Rcmdr** menu-definition file, but it does presuppose some familiarity with R, the **tcltk** package, and Tcl/Tk itself.

Protecting the novice from errors Where possible, I have tried to limit users' choices to those that are reasonable within the current context. For example, the dialog-box for an independent-samples *t*-test presents only two-level factors in the variable-list box for defining groups and only numeric variables in the list-box for the response variable. Likewise, if there are no two-level factors or no numeric variables in the active data set (or, indeed, if there is no active data set), then the menu item for an independent-samples *t*-test is grayed-out. Errors and warnings are intercepted, and where it has been possible to anticipate certain kinds of errors, an effort has been made to report understandable error messages.

To expose users to R commands The script window displays the R commands that the R Commander GUI generates, but it is my impression that most students ignore these commands. This response probably partly reflects my emphasis on generating and interpreting the output of statistical procedures, but at least the commands are there for examination and experimentation. As well, as explained, the R Commander script window has some deficiencies as a simulated R console.

3.2. What is the future of the R Commander?

If the past is prologue, then I have only limited ability to foresee where the R Commander is headed. Nevertheless, several potential directions for future development seem clear:

Additional statistical functionality It is safe to predict modest extension of the statistical capabilities of the R Commander in response to users' requests and contributions. More ambitiously, I would like to add high-interaction statistical graphics, such as scatterplots that support dynamic variable transformations and possibly linkage between different plots [in the manner of Cook and Weisberg's Lisp-Stat based Arc software (Cook and Weisberg 1999)].

Improvements to the code and to usability As I have explained, I have worked over the code for the **Rcmdr** package more than once, but there is certainly still room for improvement—in particular, further elimination of redundancy in the code. At present, R Commander dialogs are used in Philippe Grosjean's SciViews GUI for R (`http://www.sciviews.org/SciViews-R/`), and it should not be difficult to make these dialogs more generally available outside of the R Commander GUI itself. Moreover, with the exception of the statistical-modelling dialogs, R Commander dialog boxes do not "remember" user selections from one invocation of a dialog to the next; it would not be difficult—though it might be tedious—to provide this feature. Similarly, if an extended set of Tk widgets becomes conveniently available to R users of Windows, I could rework the basic layout of R Commander dialog boxes by incorporating elements such as tabs and drop-down lists.

Internationalization Using the localization and internationalization facilities introduced in version 2.1.0 of R (Ripley 2005), I have prepared a new version of the **Rcmdr** package that supports translation into other languages. The current development version of the package includes translation files for Catalan, French, Japanese, and Slovenian (kindly provided, consecutively, by Manel Salamero, Philippe Grosjean, Takaharu Araki, and Jaro Lajovic), and translations into several other languages are underway.

4. Extending the R Commander

As is the case for any R package, a user can modify the source code for the **Rcmdr** package and rebuild the package. Two features make it possible to modify or add to the **Rcmdr** package without rebuilding it, however:

1. The *R Commander* menus are defined in the plain-text (ASCII) file `Rcmdr-menus.txt`, which resides in the package's `etc` directory. Modifying this file changes the menus. The format of the file is described below.

2. Files with extension (file type) `.R` in the `etc` directory are "sourced" (read into memory) when the R Commander starts up. Consequently, functions and variables defined in `.R` files are available in the global environment.

The following example assumes some familiarity with Tcl/Tk (Welch 2000) and the **tcltk** package (Dalgaard 2001, 2002): Suppose that we want to provide a menu-item and dialog box for multivariate Box-Cox transformations to normality. The **car** package (Fox 2002), which is one of the packages that **Rcmdr** loads at startup, contains a function to perform the necessary computations, `box.cox.powers`. Because none of the existing *R Commander* menus seems appropriate, I will add a *Transform* menu under *Statistics*, with the single item *Multivariate Box-Cox transformations...* . This item will lead to a dialog box to select the variables to be transformed. Finally, I will write a function, named `BoxCox`, to construct the dialog box and invoke `box.cox.powers`.

The modified `Rcmdr-menus.txt` is as follows, eliding most of the lines in the file (the elisions are marked by widely spaced ellipses, .   .   .). I have also "wrapped" each line in the file to fit on the page, and inserted a blank line between each menu definition.[26]

```
# R Commander Menu Definitions

# last modified 26 March 2005 by J. Fox

#   type    menu/item      operation/parent  label
       command/menu                  activation

    menu    fileMenu        topMenu         ""
       ""                            ""
```

[26]The reader may wish to print the `Rcmdr-menus.txt` file in landscape mode.

```
    item    fileMenu        command         "Open script file..."
        loadLog                     ""

    item    fileMenu        command         "Save script..."
        saveLog                     ""

    item    fileMenu        command         "Save script as..."
        saveLogAs                   ""

. . .

    menu    statisticsMenu  topMenu         ""
        ""                          ""

    menu    summariesMenu   statisticsMenu  ""
        ""                          ""

    item    summariesMenu   command         "Active data set"
        summarizeDataSet            "activeDataSetP()"

. . .

    item    modelsMenu      command         "Multinomial logit model..."
        multinomialLogitModel       "factorsP() && packageLoaded('nnet')"

    item    modelsMenu      command         "Proportional-odds logit model..."
        proportionalOddsModel       "factorsP() && packageLoaded('MASS')"


      menu    transformMenu   statisticsMenu  ""
          ""                          ""

      item    transformMenu   command         "Multivariate Box-Cox transformations..."
          BoxCox                      "numericP() && packageLoaded('car')"


    item    topMenu         cascade         "Statistics"
        statisticsMenu              ""

    item    statisticsMenu  cascade         "Summaries"
        summariesMenu               ""

. . .

    item    statisticsMenu  cascade         "Fit models"
        modelsMenu                  ""
      item    statisticsMenu  cascade         "Transform"
           transformMenu                 ""

    menu    graphsMenu      topMenu          ""
         ""                           ""
. . .
```

- Each line in the file contains six entries (fields) and defines either a menu or a menu item.
- Each menu has a "parent" menu; top-level menus, such as `File` and `Statistics`, have `topMenu` as their parent. Menu definition requires two lines: One to create the menu and another to place it under its parent.
- The "operation/parent" field in each line contains the parent menu (for menu creation), `cascade` (for placing a menu under its parent), or `command` (for a menu item that invokes a command).
- The "label" field contains the text that labels a menu or menu item. By convention, menu items leading to dialog boxes have labels ending in ellipses, `...` .
- The "command/menu" field contains the name of a function to be invoked by a menu item, or the name of a menu to be installed.
- The "activation" field contains a quoted R expression that, when evaluated, indicates whether a menu item is to be active, if the expression is `TRUE`, or inactive ("grayed out"), if it is `FALSE`. The **Rcmdr** package exports a number of functions (see the discussion below and Table 2) to test the current state of the R Commander—for example, `numericP` (a "predicate" to test for the presence, and possibly sufficient number, of numeric variables in the active data set), `factorsP` (to test for the presence and number of factors), and `packageLoaded` (to test whether a specific R package has been loaded). The status of menus is assessed at R Commander start-up; it is reassessed when the active data set or active statistical model changes, and whenever the function `activateMenus` is invoked. If the activation condition is empty (i.e., if the field contains `""`), then the corresponding menu item is always active.
- The last three fields are empty (`""`) for `menu` (as opposed to `item`) lines.

Note the line in the modified `Rcmdr-menus.txt` file creating `transformMenu` as a child of `statisticsMenu`; the line creating the Box-Cox item under `transformMenu`; and the line cascading `transformMenu` under `statisticsMenu`. (These lines are indented two additional spaces in the file listing.)

The remaining task is to write the `BoxCox` function. The **Rcmdr** package exports a number of functions to assist in writing dialogs and performing computations; these are shown in Tables 1 through 5.[27]

[27]Some of the functions are provided for convenience: For example `Factors` simply calls `listFactors` with no argument, which defaults to the active data set.

Function	*Purpose*
`activeDataSet`	Returns or sets the name of the active data set.
`ActiveDataSet`	Returns the name of the active data set.
`activeModel`	Returns or sets the name of the active model.
`ActiveModel`	Returns the name of the active model.
`Factors`	Names of factors in the active data set.
`getRcmdr`	Retrieve an object from the Rcmdr environment.
`GrabFocus`	Returns (or sets) the grab-focus status.
`listDataSets`	Lists names of data frames, by default in the global environment.
`listFactors`	Lists names of factors in a data set.
`listGeneralizedLinearModels`	Lists names of glm objects, by default in the global environment.
`listLinearModels`	Lists names of lm objects, by default in the global environment.
`listNumeric`	Lists names of numeric variables in a data set.
`listTwoLevelFactors`	Lists names of two-level factors in a data set.
`listVariables`	Lists names of variables in a data set.
`Numeric`	Returns names of numeric variables in the active data set.
`putRcmdr`	Store an object in the Rcmdr environment.
`twoLevelFactors`	Names of two-level factors in the active data set.
`UpdateModelNumber`	increment (or otherwise change) the model number.
`Variables`	Names of variables in the active data set.

Table 1: Functions exported by the **Rcmdr** package for setting and retrieving information .

Function	*Purpose*
`activeDataSetP`	TRUE if there is an active data set; FALSE otherwise.
`activeModelP`	TRUE if there is an active model.
`dataSetsP`	TRUE if there are data sets in memory.
`factorsP`	TRUE if there are (sufficient) factors in the active data set.
`glmP`	TRUE if the active model is a glm object.
`hclustSolutionsP`	TRUE if there are hclust objects in memory.
`lmP`	TRUE if the active model is an lm object.
`modelsP`	TRUE if there are statistical models in memory.
`NumericP`	TRUE if there are (sufficient) numeric variables in the active data set.
`packageLoaded`	Check whether a specific package is loaded.
`twoLevelFactorsP`	TRUE if there are (sufficient) two-levels factors in the active data set.

Table 2: "Predicate" functions exported by the **Rcmdr** package. These functions are used to determine menu-item activation.

Function	*Purpose*
`checkBoxes *`	Constructs a set of check boxes.
`closeDialog *`	Close a dialog box.
`dialogSuffix *`	Housekeeping to complete dialog definition.
`errorCondition *`	Reports an error and (optionally) restarts the dialog.
`getFrame`	Returns the frame of a listbox object.
`getSelection`	Returns the currently selected elements of a listbox object.
`groupsBox *`	Constructs a button and sub-dialog box for selecting a grouping factor.
`groupsLabel *`	Constructs a text field that shows the currently selected groups.
`initializeDialog *`	Initial housekeeping for a Tk dialog box.
`modelFormula *`	Constructs a dialog component for entering a model formula.
`OKCancelHelp *`	Constructs *OK*, *Cancel*, and *Help* buttons.
`radioButtons *`	Constructs a set of related radio buttons.
`subOKCancelHelp *`	Constructs *OK*, *Cancel*, and *Help* buttons for a sub-dialog.
`subsetBox *`	Constructs a text box for entering a subsetting expression.
`variableListBox`	Constructs an object containing a scrollable list box.

Table 3: Functions exported by the **Rcmdr** package that build elements of dialog boxes.
* Functions marked with an asterisk are "macro-like" in their behavior, in that they execute in the environment of the calling function. These functions were created with a slightly modified version of Thomas Lumley's `defmacro` function (Lumley 2001).

Function	*Purpose*
`activateMenus`	Enable or disable menu items.
`checkReplace`	Allows user to verify replacement of an object.
`CommanderWindow`	Returns the Tk *R Commander* window.
`doItAndPrint`	Executes a command, given as a character string, prints command and output.
`is.valid.name`	Checks that a character string is a valid R name.
`justDoIt`	Executes a character string without echoing it to the script window.
`logger`	Echoes a character string to output window without executing it.
`logWindow`	Returns the Tk *Script* window.
`Message`	Writes a message into the messages window.
`MessagesWindow`	Returns the Tk *Messages* window.
`OutputWindow`	Returns the Tk *Output* window.

Table 4: Miscelaneous functions exported by the **Rcmdr** package.

Function	*Purpose*
`assignCluster`	Create a cluster-membership variable.
`bin.var`	Bin a numeric variable.
`colPercent`	Column percentage table.
`Confint`	Confidence intervals.
`KMeans`	K-means clustering.
`partial.cor`	Matrix of partial correlations.
`plotMeans`	Plot profiles of means by one or two factors.
`reliability`	Reliability of composite scales.
`scatter3d`	Dynamic 3D scatterplot with regression surfaces.
`stem.leaf`	Stem-and-leaf displays.

Table 5: Statistical functions exported by the **Rcmdr** package. `stem.leaf`, for high-quality stem-and-leaf displays, was generously made available to me by Peter Wolf. I am grateful to Dan Putler for contributing `assignCluster`, `bin.var`, and `KMeans`.

The dialog box to be created is very simple: It should have a variable list from which one or more numeric variables are to be selected, along with *OK*, *Cancel*, and *Help* buttons. A relatively painless procedure is to find an **Rcmdr** dialog that is similar and modify it, rather than creating code from scratch. In this case, I started with the code for the `scatterPlotMatrix` dialog, removing a number of unnecessary elements and making small changes. The resulting code is as follows:

```
BoxCox <- function(){
    initializeDialog(title="Box-Cox Transformations")
    variablesBox <- variableListBox(top, Numeric(), selectmode="multiple",
        title="Select variables (one or more)")
    onOK <- function(){
        variables <- getSelection(variablesBox)
        if (length(variables) < 1) {
            errorCondition(recall=BoxCox,
              message="You must select one or more variables.")
            return()
            }
        closeDialog()
        command <- paste("box.cox.powers(na.omit(cbind(",
            paste(paste(variables, "=", ActiveDataSet(), "$", variables, sep=""),
                collapse=", "), ")))", sep="")
        doItAndPrint(command)
        tkfocus(CommanderWindow())
```

A few exported functions are retained for backwards compatibility with older versions of the **Rcmdr** package: `checkActiveDataSet`, `checkActiveModel`, `checkFactors`, `checkNumeric`, `checkTwoLevelFactors`, and `check-Variables`.

In addition, a few exported functions are not really for users: `commanderPosition`, `is.SciViews`, `RcmdrT-clSet`, and `RcmdrPager`.

Finally, some S3 methods are exported: `glm` and `default` methods for `Confint`; `reliability` and `stem.leaf` methods for `print`; and `listbox` methods for `getFrame` and `getSelection`.

```
        }
    OKCancelHelp(helpSubject="box.cox.powers")
    tkgrid(getFrame(variablesBox), sticky="nw")
    tkgrid(buttonsFrame, sticky="w")
    dialogSuffix(rows=2, columns=1)
    }
```

Notice that the dialog box is built and manipulated almost entirely by calls to functions exported by the **Rcmdr** package—making it simple, for example, to produce the variable-list box and the row of buttons at the bottom of the dialog. An illustrative dialog box created by the `BoxCox` function appears in Figure 17.

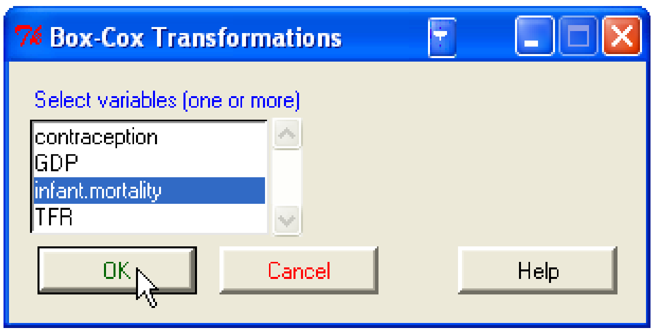


Figure 17: An illustrative dialog box produced by the `BoxCox` function.

The only potentially difficult part of the code is assembling the text string for the `box.cox.powers` command: The coding here is a bit complicated because `box.cox.powers` wants a numeric matrix as its argument, with the names of the variables as the column names. Notice the use of `doItAndPrint` to execute the command, send the command to the script window, and send the command and output to the output window. This approach will work in most cases.

The code for this example is in the file `BoxCox.demo` in the `etc` directory of the **Rcmdr** package. Rename the file to `BoxCox.R` to activate it. Likewise, the `Rcmdr-menus.txt` file distributed with the package contains commented-out lines for the example; remove the comment characters (`#`) from the beginnings of these lines to activate them.

5. Some suggestions for instructors

At the beginning of my introductory-statistics course, I distribute a manual for the R Commander based on the second section of this paper. When the software is required during the course, I begin by demonstrating its use for a particular kind of task, such as constructing a contingency table or performing a regression analysis, that is similar to the work that the students will do. Assignments that entail the use of the software are accompanied by directions that point the students towards the menus and dialogs that they will need. Students are given the opportunity to do these assignments in a supervised computer lab, but after the initial assignment, almost all work independently. With the exception of independence from the lab, this is essentially the same strategy that I previously employed with other statistical software.

Some of the social-science students whom I encounter in introductory statistics classes have difficulty installing and configuring software. I imagine that this situation varies with discipline and locale, but I also expect that it is reasonably common. I assume here that students will be using R and the R Commander under Windows, but it should not be hard to transpose these suggestions to other operating systems.[28]

I distribute to students a CD/ROM with a live, installed version of R, including all necessary packages, and configured to open R in SDI mode, to load the **Rcmdr** package at startup, and to use compiled HTML help in R. Students can simply double-click on the file `Run-R.bat` in the root directory of the CD to start R. This batch file contains a single line:[29]

```
start rw2001pat\bin\Rgui.exe
```

Starting with R version 2.0.1 "patched," it is possible to create a custom installer with packages additional to the "recommended" R packages and modified configuration files. Details are in the file
`src\gnuwin32\installer\INSTALL` of the R *source* distribution. A few tips:

- Although you have to download and unpack the R source distribution, you do not have to compile your own R Windows binary.
- You do have to install some the tools for building R, however, including Perl and the Inno Setup software for building Windows installers. Inno Setup should be installed at `c:\packages\inno4` (not in the default location under `Program Files`); alternatively, you can edit the `MkRules` file in the R source distribution to reflect the location of Inno Setup. See `http://www.murdoch-sutherland.com/Rtools/` for further information.
- The binary installation that you use as the "target" for the installer should be a complete installation of R—e.g., including all manuals, HTML help pages, etc.

I include a `ReadMe.txt` file in the root directory of the CD with the following contents:

```
    Installing the R Software and Data Files From the CD/ROM

This CD/ROM is intended for Windows 9x, ME, NT, 2000, and XP systems.
The CD/ROM contains the following files and directories:

o   The file rw2001pat.exe will install the R software on your computer
    and configure it for use in the course. Double-click on the
    file in the Windows Explorer to initiate the installation process.
    You can take all of the defaults in the R installer.
```

[28]As mentioned, an unfortunate exception at present is the Macintosh under OS/X, where more configuration is necessary to get the **tcltk** package to work. The **rgl** package, used in the **Rcmdr** for 3D scatterplots, also requires additional configuration on Macintosh systems.

[29]This following information refers to R version 2.0.1 patched, version 4 of Inno Setup, etc. Of course, these should be adjusted to current versions.

```
o   The file AdbeRdr60_enu_full.exe will install the Adobe Reader version
    6.0 on your computer. This is a viewer for PDF files; you do not have
    to install the Adobe Reader if you already have it or another PDF file
    viewer installed on your computer. You need a PDF file viewer to
    read the R Commander manual and the R manuals. Double-click on the file
    to initiate installation.

o   The directory rw2001pat\ contains a pre-installed copy of R that can
    be run directly from the CD/ROM. Double-click on the file Run-R.bat
    in the Windows Explorer to run R from the CD/ROM.

o   The directory R-Packages\ contains zip files for all of the
    packages on CRAN (the Comprehensive R Archive Network).

Note: Depending upon how your version of Windows is configured, you
may not see the file types ".bat" and ".exe" referred to here.

R is free software. Most of it is distributed under the GNU General Public
License; see the files rw2001pat\COPYING and rw2001pat\COPYRIGHTS for details.
Individual R packages have various licenses; license information is given
in the DESCRIPTION file of each package.

Prepared by John Fox <jfox@mcmaster.ca> 14 December 2004
```

Finally, the `Rprofile` file has the following contents:

```
options(chmhelp=TRUE)
library("Rcmdr")
```

while the `Rconsole` file contains the line

```
MDI = no
```

along with its other, unmodified, contents.

Acknowledgements

The work described in this paper was supported by a grant from the Social Science Research Board of McMaster University. I am grateful to Michael Ash, Philippe Grosjean, Martin Maechler, Dan Putler, and Peter Wolf, who have contributed code to the **Rcmdr** package described in this paper. Many individuals—too numerous to name here—have assisted me with suggestions and bug reports; they are acknowledged in the `CHANGES` file distributed with the package. I am also grateful to Tony Christensen for research assistance; to Bob Andersen for comments on a draft of this paper; and to two anonymous reviewers of earlier versions of the paper for their incisive criticisms and constructive suggestions. Finally, I wish to thank Peter Dalgaard for the **tcltk** package on which the **Rcmdr** package is based, and the other members of the R core team and the several package authors on whose work the statistical

functionality of the R Commander rests. This is a revised version of a paper presented at the useR! Conference, Vienna, May 2004.

References

Chambers JM, Hastie TJ (eds.) (1992). *Statistical Models in S.* Wadsworth, Pacific Grove CA.

Cook RD, Weisberg S (1999). *Applied Regression Including Computing and Graphics.* Wiley, New York.

Dalgaard P (2001). "A Primer on the R-Tcl/Tk Package." *R News*, **1**(3), 27–31.

Dalgaard P (2002). "Changes to the R-Tcl/Tk Package." *R News*, **2**(3), 25–71.

Fox J (2002). *An R and S-PLUS Companion to Applied Regression.* Sage, Thousand Oaks CA.

Fox J (2003). "Effect Displays in R for Generalised Linear Models." *Journal of Statistical Software*, **8**(15), 1–27.

Ihaka R, Gentleman R (1996). "R: A Language for Data Analysis and Graphics." *Journal of Computational and Graphical Statistics*, **5**, 299–314.

Lumley T (2001). "Programmer's Niche: Macros in R." *R News*, **1**(3), 11–13.

Moore DS (2000). *The Basic Practice of Statistics, Second Edition.* Freeman, New York.

Moore DS (2004). *The Basic Practice of Statistics, Third Edition.* Freeman, New York.

R Core Development Team (2004). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna.

Ripley BD (2005). "Internationalization Features of R 2.1.0." *R News*, **5**(1), 2–7.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S, Fourth Edition.* Springer, New York.

Welch BB (2000). *Practical Programming in Tcl and Tk.* Prentice Hall, Upper Saddle River NJ.

Wilkinson GN, Rogers CE (1973). "Symbolic Description of Factorial Models for Analysis of Variance." *Applied Statistics*, **22**, 392–399.

Affiliation:

John Fox
Department of Sociology
McMaster University
Hamilton, Ontario
Canada L8S 4M4
E-mail: jfox@McMaster.ca
URL: http://socserv.mcmaster.ca/jfox/

Journal of Statistical Software
September 2005, Volume 14, Issue 9.
http://www.jstatsoft.org/

Submitted: 2004-07-22
Accepted: 2005-08-19