A *Brief* Introduction to R

January 23, 2009

This is a document designed to help a person to begin to get to know the R statistical computing environment. It paraphrases and summarizes information gleaned from materials listed in the **References**. Please refer to them for a more complete treatment.

1 Installing R and the `RcmdrPlugin.IPSUR` Package

There are detailed instructions for installing R on your personal computer at the following website:

http://ipsur.r-forge.r-project.org/installation.php

For more complete and technical installation instructions see the *R Installation and Administration Manual* http://cran.r-project.org/doc/manuals/R-admin.html.

2 Communicating with R

There are three basic methods for communicating with the software.

1. At the Command Prompt (>).

   This is the most basic way to complete simple, one-line commands. R will evaluate what is typed there and output the results in the Console Window.

2. Copy & Paste from a text file.

   For longer programs (called *scripts*) there is too much code to write all at once at the Command Prompt. Further, for long scripts the user sometimes wishes to only modify a certain piece of the script and run it again in R.

   One way to do this is to open a text file with a text editor (say, NotePad or MS-Word®). One writes the code in the text file, then when satisfied the user copy-and-pastes it at the Command Prompt in R. Then R will compile all of the code at once and give output in the Console Window.

   Alternatively, R provides its own built-in script editor, called R Editor. From the console window, select *File* → *New Script*. A script window opens, and the lines of code can be written in the window. When satisfied with the code, the user highlights all of the commands and presses `Ctrl+R`. The commands are automatically run at once in R and the output is shown. To save the script for later, click *File* → *Save as...* in R Editor. The script can be reopened later with *File* → *Open Script...* in the Console Window.

   A disadvantage to these methods is that all of the code is written in the same way, with the same font. It can become confusing with longer scripts, and there is no way to efficiently identify mistakes in the code. To address this problem, software developers have designed powerful IDE / Script Editors.

3. IDE / Script Editors.

   There are free programs specially designed to aid the communication and code writing process. The advantage to using Script Editors is that they have additional functions and options to help the user write code more efficiently, including R syntax highlighting, automatic code completion, delimiter matching, and dynamic help on the R functions as they are written. In addition, they typically have all of the text editing features of programs like MS Word. Lastly, most script editors are fully customizable in the sense that the user can customize the appearance of the interface and can choose what colors to display, when to display them, and how they are to be displayed.

Some of the more popular script editors can be downloaded from the R-Project website at http://www.sciviews.org/_rgui/. On the left side of the screen (under **Projects**) there are several choices available.

- **RWinEdt**: You can get this from IDE/Script Editors, under the section on Uwe Ligges. This program has a window based on WinEdt for LaTeX and has features such as code highlighting, remote sourcing, and all of the familiar ones of WinEdt. Unfortunately, this one is only Shareware, so you first need to download WinEdt, and then it is only free for a while. Eventually, annoying windows will pop-up asking if you want to register. This would be a fine choice if you like LaTeX and have WinEdt already, or are planning on purchasing WinEdt in the future.
- **Tinn-R**: This one has the advantage of being completely free, with no additional requirements. It has all of the above mentioned options and lots more. It is simple enough to use that the user can virtually begin working with the program immediately after installation. Unfortunately, this program is only availabe for Windows based systems.
- **Bluefish**: This open-source script editor is for Mac OSX users. Other alternatives for Mac users are SubEthaEdit, AlphaTk, and Eclipse. I have not used these yet, so I cannot comment on their strngths and weaknesses. Try them out, and let me know!
- **Emacs** / **ESS**: Click Emacs (ESS) or Emacs (ESS/Windows). This will take you to download sites with sophisticated programs for editing, compiling, and coordinating software such as `S-Plus`, R, and `SAS` simultaneously. Emacs is short for *E*diting *MAC*ro*S* and ESS means *E*macs *S*peaks *S*tatistics. An alternate branch of Emacs is called XEmacs. This editor is – *by far* – the most powerful of the text editors, but all of the flexibility comes at a price. Emacs requires a level of computer-saavy that the others do not, and the learning curve is more steep. If you want to explore this option, then speak with me beforehand; I can give you some advice about getting started.

3 A First Session: Using R as a calculator

R is perfectly able to do standard calculations. For example, type $2 + 3$ and observe

```
> 2+3
[1] 5
>
```

The `[1]` means that the 5 is the first entry in the list, and the `>` means that R is waiting on your next command. Entry numbers will be generated for each row, such as

```
> 3:50
 [1]  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
[19] 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
[36] 38 39 40 41 42 43 44 45 46 47 48 49 50
```

Here, the 19^{th} entry in the list is 21. Notice also the `3:50` notation, which generates all numbers in sequence from 3 to 50. One can also do things like

```
> 2*3*4*5 # multiply
[1] 120

> sqrt(10) # square root
[1] 3.162278

> pi # pi
[1] 3.141593

> sqrt(-2)
[1] NaN
Warning message:
NaNs produced in:  sqrt(-2)
```

Notice that `NaNs` were produced; this stands for "not a number". Also notice the number sign `#`, which means *comment*. Everything typed on the same line after the `#` will be ignored by R. There is also a continuation prompt `+` which occurs if you press `Enter` before a statement is complete. For example, if you forget to close the parentheses or a command you may get something like the following:

```
> sqrt(27+32
+
+
```

To exit out of the continuation prompt, you can either complete the command - by entering a) in the above example - or you may press the Esc key.

Some other fuctions that will be of use are `abs()` for absolute value, `log()` for the natural logarithm, `exp()` for the exponential function, `factorial()` for computing permutations, and `choose()` for binomial coefficients.

Assignment. This is useful for storing values to be used later.

```
> y = 5 # stores the value 5 in y
> y
[1] 5
> y <- 5 # also stores the value 5 in y
> 7 -> z # stores the value 7 in z
```

You do not have to use the `<-` notation to store things; the equal sign `=` works just as well. I will use both symbols interchangeably.

Acceptable variable names. You can use letters, numbers, dots ".", or underscore "_" characters. You cannot use mathematical operators, and a leading dot may not be followed by a number. Examples: `x`, `x1`, `y32`, `x.variable`, `x_variable`.

Using `c()` to enter data vectors. If you would like to enter the data `74,31,95,61,76,34,23,54,96` into R, you may create a data vector with the `c()` function (short for *concatenate*).

```
> fred = c(74, 31, 95, 61, 76, 34, 23, 54, 96)
> fred
[1] 74 31 95 61 76 34 23 54 96
```

The vector `fred` has 9 entries. We can access individual components with bracket `[ ]` notation:

```
> fred[3]
[1] 95
> fred[2:4]
[1] 31 95 61
> fred[c(1, 3, 5, 7)]
[1] 74 95 76 23
```

If you would like to reset the variable `fred`, you can do it by typing `fred = c()`.

Using `scan()` to enter numeric data vectors. If you would like to enter the data 76 34 23 54 96 into a vector `x`, perhaps the quickest way would be to use the `scan()` function:

```
> x=scan()
1:  76
2:  34
3:  23
4:  54
5:  96
6:
Read 5 items
```

This method is best suited for use with small data sets and **only works if the data are numeric**. Notice that entering an empty line stops the scan. Another use of this feature is when you have a long list of numbers (separated by spaces or on different lines) already typed somewhere else, say in a text file. To enter all the data in one fell swoop, highlight and copy the list of numbers to the Clipboard with *Edit* → *Copy*, next type the `x=scan()` command in the R console, and paste the numbers at the `1:` prompt with *Edit* → *Paste*. All of the numbers will automatically be entered into the vector `x`.

Data vectors have type. There are numeric, character, and logical type vectors. If you mix and match then usually it will be character. Notice that characters can be identified with either single or double quotes.

```
> simpsons = c("Homer", 'Marge', "Bart", "Lisa", "Maggie")
> names(simpsons) = c("dad", "mom", "son", "daughter 1", "daughter 2")
> simpsons
     dad       mom       son  daughter 1  daughter 2
 "Homer"   "Marge"   "Bart"      "Lisa"     "Maggie"
```

Here is an example of a logical vector:

```
> x = c(5,7)
> v = (x<6)
> v
[1] TRUE FALSE
```

Applying functions to a data vector. Once we have stored a data vector then we can evaluate functions on it.

```
> fred
[1] 74 31 95 61 76 34 23 54 96
> sum(fred)
[1] 544
> length(fred)
[1] 9
> sum(fred)/length(fred)
[1] 60.44444
> mean(fred) # sample mean, should be the same answer
[1] 60.44444
> sd(fred) # sample standard deviation
[1] 27.14365
```

Other popular functions for vectors are `range()`, `min()`, `max()`, `sort()`, and `cumsum()`.

Vectorizing functions. Arithmetic in R is almost always done element-wise, also known as *vectorizing functions*. Some examples follow.

```
> fred.2 = c(4,5,3,6,4,6,7,3,1)
> fred+fred.2
[1] 78 36 98 67 80 40 30 57 97
> fred-fred.2
[1] 70 26 92 55 72 28 16 51 95
> fred - mean(fred)
[1] 13.5555556 -29.4444444 34.5555556 0.5555556 15.55556 -26.44444
[7] -37.4444444 -6.4444444 35.5555556
```

The operations `+` and `-` are performed element-wise. Notice in the last vector that `mean(fred)` was subtracted from each entry, in turn. This is also known as *data recycling*. Other popular vectorizing functions are `sin()`, `cos()`, `exp()`, `log()`, and `sqrt()`.

4 Getting Help

When you are using R, it will not take long before you find yourself needing help. Fortunately, R has extensive help resources and you should immediately become familiar with them. Begin by clicking *Help* on the console. The following options are available.

- **Console**: gives useful shortcuts, for instance, `Ctrl+L`, to clear the R console screen.
- **FAQ on R**: frequently asked questions concerning general R operation.
- **FAQ on R for Windows**: frequently asked questions about R, tailored to the Windows operating system.

- **Manuals**: technical manuals about all features of the R system including installation, the complete language definition, and add-on packages.
- **R functions (text)...**: use this if you know the *exact* name of the function you want to know more about, for example, `mean` or `plot`. Typing mean in the window is equivalent to typing `help("mean")` at the command line, or more simply, `?mean`.
- **Html Help**: use this to browse the manuals with point-and-click links. It also has a Search Engine & Keywords for searching the help page titles, with point-and-click links for the search results. This is possibly the best help method for beginners.
- **Search help...**: use this if you do not know the exact name of the function of interest. For example, you may enter `plo` and a text window will return listing help files with an alias, concept, or title matching 'plo' using regular expression matching; it is equivalent to typing `help.search("plo")` at the command line. The advantage is that you do not need to know the exact name of the function; the disadvantage is that you cannot point-and-click the results. Therefore, one may wish to use the Html Help search engine instead.
- **search.r-project.org...**: this will search for words in help lists and archives of the R Project. It can be very useful for finding other questions that useRs have asked.
- **Apropos...**: use this for more sophisticated partial name matching of functions. Try `?apropos` for details.

Note also `example()`. This initiates the running of examples, if available, of the use of the function specified by the argument.

5 Other tips

It is unnecessary to retype commands repeatedly, since R remembers what you have entered on the command line. To cycle through the previous commands, just push the ↑ (up arrow) key.

Missing values in R are denoted by `NA`. Operations on data vector `NA` values treat them as if the values can't be found. This means adding (as well as subtracting and all of the other mathematical operations) a number to `NA` results in `NA`.

To find out what all variables are in the current work environment, use the commands `ls()` or `objects()`. These list all available objects in the workspace. If you wish to remove one or more variables, use `remove(var1, var2)`, and to remove all of them use `rm(list=ls())`.

6 Some References

- Dalgaard, P. (2002). *Introductory Statistics with* R. Springer.
- Everitt, B. (2005). *An* R *and* `S-Plus` *Companion to Multivariate Analysis*. Springer.
- Heiberger, R. and Holland, B. (2004). *Statistical Analysis and Data Display. An Intermediate Course with Examples in* `S-Plus`, R, *and* `SAS`. Springer.
- Maindonald, J. and Braun, J. (2003). *Data Analysis and Graphics Using* R: *an Example Based Approach*. Cambridge University Press.
- Venables, W. and Smith, D. (2005). *An Introduction to* R. http://www.r-project.org/Manuals.
- Verzani, J. (2005). *Using* R for *Introductory Statistics*. Chapman and Hall.

7 Exercises with R (Verzani, 2005)

Directions: Complete the following exercises and submit your answers. *Please Note*: only answers are required; it is not necessary to submit the R output on the screen.

1. Let our small data set be 5, 6, 2, 3, 1, 9.

   (a) Enter this data into a vector `x`.

   (b) Square each of the numbers in `x`.

   (c) Subtract 9 from each number in `x`.

   (d) Add 5 to all of the numbers in `x`, then take the (natural) logarithm of the answers.

   Use vectorization of functions to do all of the above, using a single line of code for each.

2. The asking price of used MINI Coopers varies from seller to seller. An online listing has these values in thousands: 17.3, 21.4, 18.9, 21.9, 20.0, 16.5, 17.9, 17.5, 18.2, 19.1, 17.3, 16.5, 18.7.

   (a) What is the smallest amount? The largest?

   (b) Find the average amount with `mean()`.

   (c) Calculate the difference of the mean value from the largest and smallest amounts (the first number will be positive, the second will be negative).

3. The twelve monthly sales of Hummer H2 vehicles in the United States during 2002 were

   ```
   [Jan] 2700 2400 3050 2900 3000 2500 2600 3000 2800
   [Oct] 3200 2800 3400
   ```

   Note that the first entry above was the sales from January, the second entry was from February, and so forth.

   (a) Enter these data into a variable `H2`. Use `cumsum()` to find the cumulative total sales for 2002. What was the total number sold?

   (b) Using `diff()`, find the month with the greatest increase from the previous month, and the month with the greatest decrease from the previous month. *Hint:* Dont know how to use `diff()`? No problem! Check it out using the *Help* system.

4. You track your commute times for 10 days, recording the following times (in minutes): 19, 16, 20, 24, 22, 15, 21, 15, 17, 22.

   Enter these data into R. Use the function `max()` to find the longest travel time, `min()` to find the smallest, `mean()` to find the average time, and `sd()` to find the sample standard deviation of the times.

   Oops! The 17 was a mistake. It should have been 18, instead. How can you fix this (without retyping the whole vector)? Correct the mistake and report the new `max`, `min`, `mean`, and sample standard deviation.