Getting Started with R

Lecture 1

Nicholas Christian

BIOST 2094 Spring 2011

Outline

1. Introduction
   - What is R?
   - Installing R
2. R Console and Script Editor
   - R as a calculator
   - Assigning objects and function calls
3. Documentation
4. Other Features
   - Working Directory and Workspace
   - Setting options

What is R?

- R is a high-level computer language and environment for statistics and graphics
  - Performs a variety of simple and advanced statistical methods
  - Produces high quality graphics
  - R is a computer language so we can write new functions that extends R's uses
- R was initially written by Ross Ihaka and Robert Gentleman at the Department of Statistics of the University of Auckland in Auckland, New Zealand (hence the name).
- R is a free open source software maintained by several contributors. Including an "R Core Team" of 17 programmers who are responsible for modifying the R source code.
- The official R home page is

`http://www.R-project.org`

Installing R

- The R system can be installed on, Windows, Mac or Linux.
- You will want to install the base system. To do so visit the R website and follow the installation directions.
- In addition to the base system there are user contributed add-on packages.
- Packages are a collection of functions, examples and documentation that usually focus on a specific task.
- The base system contains some packages. To install an additional package, say gplots, be connected to the Internet and type

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

  You will be asked to select the mirror site nearest to you, after that everything is automatic.
- Before using the contents of the package we need to load it,

  ```
  > library(gplots)
  ```
- See the R website for a complete list of contributed packages

R Console

- The R Console is where computations are performed
- An expression is inputed into the console and the expression is evaluated. Depending on the expression, the system may respond by outputting results to the console or creating a graph in a new window. Then another expression is inputed and evaluated.
- An R session is this interaction between you and the system
- To get the last expression entered use the up arrow
- To get the value of the last evaluated expression type `.Last.value`
- Press Esc to stop evaluating the current expression

Calculator

- Enter a math expression and the result is outputted to the console

Binary Operators	`+`	`-`	`*`	`/`	`^`	`%%`
Math Functions	`abs`	`sqrt`	`log`	`exp`	`log10`	`factorial`
Trig Functions	`sin`	`cos`	`tan`	`asin`	`acos`	`atan`
Rounding	`round`	`ceiling`	`floor`	`trunc`	`signif`	`zapsmall`
Math Quantities	`Inf`	`-Inf`	`NaN`	`pi`	`exp(1)`	`1i`

%% is for modular arithmetic

```
> 5 %% 4
[1] 1
> log(2)
[1] 0.6931472
> cos(pi)
[1] -1
> ceiling(3.2)
[1] 4
> 0/0
[1] NaN
> 1/Inf
[1] 0
```

Objects and Functions

- What we normally do is create objects and then perform functions on those objects (functions are also considered objects, more on this later).
- Assign an object a name "x" using either

  `x <-` *object* (*object* `->` `x`)

  `x =` *object*
- Call a function by

  function name(*list of arguments separated by commas*)

  - Each function has a set of formal arguments some with default values; these can be found in the function's documentation.
  - A function call can include any subset of the complete argument list.
  - To specify a particular argument use the argument's name.
  - Arguments do not have to be named if they are entered in the same order as the function's formal argument list. However, to make your code easier to understand it is usually a good idea to name your arguments.
  - When specifying values for an argument use an $=$.
- R is CASE SENSITIVE.

Example - Assigning Objects & Function Calls

- Suppose I want to find the mean of a set of numbers. I first assign the vector of numbers a clever name `x` and then call the function `mean()`.

```
> x = c(0,5,7,9,1,2,8)
> x
[1] 0 5 7 9 1 2 8
> mean(x)
[1] 4.571429
> X
Error: object 'X' not found
```

- Now suppose I want to sort a vector `y` so that the numbers are descending. By default `R` will sort ascending so I need to change the formal argument `decreasing` to `TRUE` (the default value for `decreasing` is `FALSE`).

```
> y <- c(4,2,0,9,5,3,10)
> y
[1]  4  2  0  9  5  3 10
> sort(y)
[1]  0  2  3  4  5  9 10
> sort(y, decreasing=TRUE)
[1] 10  9  5  4  3  2  0
```

Script Editor

- The script editor is used for writing programs in R.
- To start a new script, click File> New Script
- The easiest way to run code is keyboard shortcuts
  - To run part of a program, highlight the lines you want and hit Ctrl+R
  - To run an entire program, select all the code then run, Ctrl+A then Ctrl+R
- When you save a script file in R you need to include the .R extension, R doesn't automatically add the file type.
- You don't need semi-colons at the end of each line. However, if you enter more than one expression on the same line you will need semi-colons.
- You can wrap a long expression onto several lines, but you need to be careful that R does not treat your unfinished code as a complete expression.
- To comment a line of code use a #. There is no block commenting in R.

Commenting Code

Comments are notes that help users understand portions of your code. They are also useful reminders to yourself of what was done and why it was done. Including meaningful comments in code is a major part of writing good programs, this semester we will practice commenting our programs.

Example - Sample Script

```
# Calculate the mean of x
x = c(0,5,7,9,1,2,8)
mean(x)

# How and how not to wrap expressions
long.variable.name <- 5
really.long.variable.name <- 7

# R views the first line as a complete expression.  Thus the code is
# treated as two separate expressions instead of one long expression.
long.answer.name <- 500*factorial(long.variable.name)
                              + sqrt(really.long.variable.name)

# Here the first line is not a complete expression (trailing + sign) so
# R continues reading lines of code until the expression is complete
long.answer.name <- 500*factorial(long.variable.name) +
                              sqrt(really.long.variable.name)

# Writing two expressions on the same line requires a ;
mean(x); var(x)
```

Documentation

- R function documentation contains:
  - A general description of the function
  - A list and description of the function's formal arguments and default settings
  - Details about the function
  - A list and description of values returned by the function
  - References
  - An executable example

  Sometimes the documentation is very complete and other times it is vague. It is more of a quick reference and not intended for instruction.
- Better educational resources are the R manuals, contributed manuals, and FAQs available on the R website.
- There are also mailing lists. You can search archived posts or post new questions. Posting is not for the faint of heart. If you post, be sure to have a thoughtful question and read the posting guide.

Documentation

- To look at the documentation for a specific function from a loaded package,

  `?`*function name*

  `help(`*function name*`)`

- To look at the documentation of a symbol put quotations around the symbol, i.e. ?":"
- To search the documentation of all installed packages for key words,

  `??"`*key words*`"`

  `help.search("`*key words*`")`

  A list of functions with the corresponding package is returned, *package*::*function name*
- To run the example included with the documentation,

  `example(`*function name*`)`
- There are also demos included with the base system and some packages. To see a complete list of demos in the base system, `demo()`. To run a particular demo, `demo("`*topic*`")`.

Example - Documentation

- Suppose we need help with the linear regression function. If we know the name of the function, `lm`, but have a question about the syntax,

```
> ?lm
```

- On the other hand if we don't know what function to use,

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

- To execute the example include with `lm`,

```
> example(lm)
```

Package Documentation

- To get an overview of an installed package and its contents,

  `help(package=`*package name*`)`

- Some packages include a vignette, a file with additional documentation and examples. To get a list of all vignettes from all installed packages,

  `vignette()`

- To see a particular vignette,

  `vignette("`*topic*`")`

- We don't need to load the package to look at the contents or vignette, but we do need to load the package to look at the documentation for a specific function or to run a demo.

Example - Package Documentation

- Suppose we are interested in making a flow chart. After browsing the list of contributed packages on the R website we decide to use `diagram`.
- First download, install and load the package `diagram`

```
> install.packages("diagram")
> library(diagram)
```

- List of functions with brief descriptions in the `diagram` package

```
> help(package=diagram)
```

- View the vignette

```
> vignette("diagram")
```

- Look at the documentation for the function `plotmat()`

```
> ?plotmat
```

Working Directory

- When we load/save datasets, load source files or save graphs we will need to specify the file path. To avoid typing the path every time we can specify a working directory.
- To set the working directory click File > Change dir... or type `setwd`(*file path*)

Workspace

- The workspace is where all the objects you create during a session are located.
- When you close `R` you will be prompted to "Save workspace image?" If you say yes, the next time you open `R` from the same working directory the workspace will be restored. That is all of the objects you created during your last session will still be available. Also see `save.image()`.
- Depending on what you are working on, it is usually wiser to write a script and then run the entire script with each new session. This way you know exactly what objects you have created; sometimes lingering objects can cause errors in programs. If a particular object is very important then save it to a file.
- Managing the Workspace
  - To list what variables you have created in the current session, `ls()`
  - To see what libraries and dataframes are loaded, `search()`
  - To see what libraries have been installed, `library()`
  - To remove an object, `rm`(*object names*)
  - To remove all objects, `rm(list=ls())`

options()

- The function `options()` sets several global options that affect how `R` computes and displays results.
- To look at the value of a single option, `getOption("`*option name*`")`
- Options reset to the defaults with each new `R` session, even if you save the workspace.
- For example suppose we want to change the maximum number of digits printed from 7 (default) to 15.

```
> defaults <- options()     # Save all default options
> getOption("digits")
[1] 7
> pi
[1] 3.141593
> options(digits=15)        # Change to 15 digits
> pi
[1] 3.14159265358979
> options(defaults)         # Restore all default options
> getOption("digits")
[1] 7
```