R Overview
Howard Seltman June 2010

Installation: For Windows/Mac/Linux. Go to http://www.r-project.org, and choose CRAN under download, then a mirror (USA/Statlib is us), then a platform. Only “base” is needed. I recommend choosing custom installation so that you can change the default from MDI to SDI. (If you forget this, you can add “-sdi” to the Windows shortcut. The R homepage has a “Manuals” link.

Startup and shutdown: Double click the big “R”. Shut down by typing q() or using File/Exit on the menu. Usually you will answer “yes” to “save the workspace” so that when you return, everything is as you left it, including data and results.

Workspace management: To keep data and results for each project from interfering with each other, use a separate folder for each project. Create the folder before starting R. From R use the menu item File/ChangeDir... to select your folder the first time you work with the project. Save the workspace when you exit. This will create an .Rdata file in your specific folder. In the future for this project, start R by double clicking on the .Rdata file (and continue to “save the workspace” whenever you quit R).

Package menu:
• Install packages: installs add-ins from the standard repository. Only needs to be done once per package (until you install a new version of R). Highly automated.
• Load package: makes an installed package available for the current session. Alternatively enter, e.g., library(rube) at the prompt. (The library() command can be entered into the .First function for a particular folder/.Rdata file to make this automatic.)
• Install packages from local zip files: For packages not (yet) submitted to the repository (like rube).

File menu:
• Source R code: runs code from a file (usually with a .R extension)
• New script: creates an empty text file for entering R commands that you want to save for future use or documentation purposes.
• Edit script: open a previously created script (code file). Inside scripts, control-r runs a line or several highlighted lines.
• Save to file: saves the visible input/output for documentation purposes

Concept overview:
• Most work is done at the command prompt (>) rather than from menus.
• Whatever you type at the prompt is evaluated. If it starts with “varName=” the evaluated result of the right hand side is stored in the object with that variable name. This is called an assignment. “<-“ is a synonym for “-“. Expressions to be evaluated look like standard math form with “*” for multiply, “^” for power, and standard function names like log(2), log10(2), and exp(2). Typing just a variable name is a shorthand for print(varName) and will display the result.
• Variable names should start with a letter, followed by numbers and letters and “_” and “.”. They can be up to 256 characters long. They are case sensitive. (Some system variable start with “.”.)
• Help takes the form of help(glm), ?glm, ??glm, and help.start(). Single question marks only work if what you are searching for is an R function. http://www.rseek.org is an attempt to limit searching to R, which is hard in Google.
• Functions always use parentheses to specify arguments and use empty parenthesis if no arguments are needed, e.g., options(). Function calls have a flexible argument matching system. Multiple arguments are separated by commas. Usually the first one or few arguments are the key arguments and are entered “unnamed” and must be in the correct order. Less used arguments are defaulted or changed by entering them in the correct order position or out of order with argName=argValue format. Incomplete argument names are allowed.
• Line continuation is tricky: If you type an incomplete R expression followed by “Enter”, you will get a “+” prompt and can continue the expression. In script files this can cause silent problems if your expression is syntactically complete when you go to a new line to continue it; the second portion is silently evaluated, but not used. At any time R is confused about your input or you change your mind, use the Escape key to reset the prompt.

Objects in R  There are many object types in R, and you can create new ones.

• Vectors (of numbers, logics (TRUE/FALSE), strings, etc.) are the basic variable type. A single value is a vector of length one. Numbers use standard formatting including 3.4e-7 standard scientific format. Strings are enclosed in single or double quotes. Logicals are written in upper case and should be written out in full. Vectors cannot mix types. The simple vector creation function is c(), e.g., ages=c(24,55,22,17) or names=c(“Jamie”,“Howard”,“Courtney”,“Turand”).
  The class(myVariable) function always identifies object types, and the length(myVariable) function tells the size of a variable. (variable size can be changed at will.)
• Vector element extraction takes many forms, e.g., ages[4], names[2:3], names[c(TRUE,FALSE,FALSE,TRUE)], and, because the value of ages>23 is c(TRUE,TRUE, FALSE,FALSE), the expression names[ages>34] gives c(“Jamie”,“Howard”).
• Lists flexibly allow any mixed types, e.g. instructor=list(name=“Howard”, married=TRUE, spouse=“Kathy”, age=55). List elements can be extracted with either the “$” operator as in instructor$age or the “[“ operator as in instructor[[“name”]] or by number as in instructor[[4]]. Single brackets return sublists.
• A special list type is the data.frame, created with a statement like myDf=data.frame(ages, names, male=c(0,1,0,1)) or from, e.g., read.csv(“myFile.csv”). (Use saveAs with the “csv” type on some rectangular data in Excel to try this.) A data.frame object represents rectangularly formatted data with the same length for each column, but different data types are allowed. The “$” extractor pulls off columns, while the myDf[3:5,2:7] format pulls off a sub-data.frame of particular rows and columns, and e.g., myDf[c(1,4),“ages”] pulls of a column as a vector. The names(myDf) function shows the column names.
Matrices are created with `matrix(data, nrow=, ncol=)` and higher dimensional objects with `array(data, dim=)`. Matrices and arrays can be queried with `dim(myMatrix)` and `dimnames(myMatrix)`. These also work for `data.frame`

Specific statistical function produce complex object types, e.g., “lm” for a regression result. These can be queried with `names(myLmObject)` followed by “$” extraction.

Advanced topic: User defined functions can be created with `double=function(x) return(2*x)`. Now `double(12)` gives 24 as an answer. Like other objects, functions can be examined by typing their names without the parentheses, e.g., `double`, or `summary.lm`.

Some useful functions

- `source("someFile.R")` or “Source R code on the menu” loads objects including data and new functions from some file of valid R code.
- `ls()` lists all currently available objects that you have created.
- `print()` displays an object based on what is in the specific print() function for the class of the object being printed. `str()` shows everything in an object in a less user-friendly way. `summary()` tends to give more detailed info than `print()`.
- `apply()` repeatedly runs a function on rows or columns of a matrix or array. `sapply` and `lapply` repeatedly apply a function to the elements of a list, including the columns of a `data.frame`. The “s” means that a simplification of the result will be attempted. E.g., `lapply(myDtf,class), sapply(myDtf,class), and sapply(myDtf, mean)`.
- `table(myDtf$male, myDtf$ages)` does a cross-tabulation, while `prop.table(table(myDtf$male, myDtf$ages),1)` show proportions for a cross-tab (1=rows, 2=columns).
- `with()` is a convenience that saves typing “data.frame$” repeatedly: `with(myDtf, table(male,ages))`.
- `with(myDtf, aggregate(ages, list(first=names, isMale=male), median))` makes a complex table by calculating medians of the ages cross-classified by name and gender.
- `Mod1=lm(ages ~ male, data=myDtf)` runs a regression of age on gender taking the data from myDtf. The result can be examined with `Mod1, print(Mod1), summary(Mod1), plot(Mod1)`. Other functions extract useful info: `coefficients(Mod1), coefficients(summary(Mod1)), residual(Mod1), fitted(Mod1)`.
- `methods(summary)`, e.g., lists all of the different type of summary function for different classes of objects. More useful is `methods(class=class(Mod1))` which shows all the built-in functions that know how to do something special with your object.
- `plot(myDtf$ages ~ myDtf$male), plot(myDtf$ages ~ myDtf$male, xlab="age in years", ylab="Wrong axis"), with(myDtf, plot(ages~male)), and with(myDtf, boxplot(ages~male))` are quick examples of plotting commands.
- `curve(dgamma(x,shape=2,rate=5),from=0, to=5)` draws a density curve (pdf).
- `write.table(myDtf, "newFile.txt", sep=" ",row.names=FALSE, col.names=TRUE, quote=FALSE)` is an example of a way to export data.
- `is.na(someVariable)` tests for missing data, e.g., `sapply(c(3,5,NA,7,NA,12), is.na)`.