1/14/2010 36-402/608 ADA-II H. Seltman
Handout #2: R Review

1. Finish Tuesday's lecture notes
2. R housekeeping details
   (a) Single document interface: on install or with –sdi option
   (b) Windows shortcuts: Under "Properties" change "Start in"
   (c) Google search for R-related documents: http://www.rseek.org/
   (d) FAQ: http://cran.r-project.org/doc/FAQ/R-FAQ.html
   (e) ? vs ?? for help. Backticks for operators: ?`[` or ?`%*%`
   (f) Before any "dangerous" operation: `save.image()`
   (g) Avoid defining c, C, D, F, I, q, t, T, time. Check with `conflicts()`.
   (h) Use `options(locatorBell=FALSE)` to turn off the annoying `locator()` bell.
3. Objects
   (a) "Everything is an object"; examine with `typeof()` or `mode()`
   (b) Vectors: logical, integer, double, complex, character (string), (raw). Elements are accessed using [foo] notation. What can go inside the square brackets:
      i. positive index number(s), possibly with repeats
      ii. negative index number(s): for excluding values
      iii. a character vector matching names(), possibly with repeats
      iv. a logical vector of TRUE/FALSE values (usually full length)
      v. can be used on the left: x[3:5]=y[1:3] replaces elements 3, 4, and 5 in x
   (c) Lists are "generic vectors": may have mixed modes; may be nested. Single elements are accessed using [[foo]] notation. Sub-lists are accessed using [foo] notation.
   (d) Lists and vectors (and other things) have `length()` values.
   (e) A data.frame is a special list with equal length components.
   (f) Arrays have a "dim"ension attribute and optionally a "dimnames" attribute, but are still, at their core, vectors. Their values are accessed using `foo[a,b,...,drop=TRUE]` notation. A matrix is a 2-dimensional array with a special `matrix()` convenience function for construction. Matrices can also be indexed with a 2-column matrix with row numbers in the first column and column numbers in the second column:

```
> x=matrix(1:18, nrow=6, dimnames=list(LETTERS[1:6],1:3))
> x[matrix(c(4,3, 6,2, 1,1), 3, byrow=T)]
[1] 16 12  1
```

(g) Functions are objects with an argument list, a body, and an environment. Arguments are passed by value (copied). Some object must be returned (even if just NULL).

(h) Argument lists can be specified with implicit or explicit names, e.g., `read.table("abc.txt", sep=",")`. Use, e.g., `args(read.table)`, to list the arguments of non-method, non-hidden functions.

(i) Environments 1) hold named objects in a "frame" and 2) have a parent ("enclosing") environment.

i. On starting R, your (current) environment is ".GlobalEnv", examined with `ls()`.

ii. The `search()` path shows the parents of .GlobalEnv, in order back to "base" which is the only one to have no parent (actually, R_EmptyEnv).

iii. If the value of an object that is not in the current environment is needed, the parent environment is checked, then its parent, etc. When an object is stored with "foo=bar" or "foo<-bar" it is assigned in the current environment (overwriting or as a new variable).

iv. Calling a function changes your current environment to a new environment that contains the function arguments (but possibly with delayed evaluation). The parent environment of that environment is the environment within which the function was *created*, not *called* (and is a reference, not copied).

v. You can use, e.g., `get("foo", envir=.GlobalEnv)` to read directly from another environment (here the global environment), e.g., to access a "hidden" variable. You can use, e.g., `assign("bar", foo, envir=.GlobalEnv)` to assign in another environment, but do this with caution. You can use, "foo<<-bar" to assign wherever an existing variable is found, but this is too dangerous to use.

vi. `library()` and `attach()` install new items on the search path (by default in position 2).

vii. (The "namespace" mechanism overrides the search path.)

(j) Attributes of (non-null) objects

i. Generic read: `attributes(foo)`, `attr(foo, "bar")`

ii. Reading with convenience functions: `names(foo)`, `dim(foo)`, `class(foo)`, etc.

iii. Generic set: `attr(foo, "bar") = foobar`

iv. Setting with convenience functions: `names(foo)=paste("Rx",1:5,sep="")`, etc.

4. `read.table()` complications
   (a) comma-separated-values: use `sep=","`
   (b) tab-separated-values: use `sep="\t"`
   (c) R makes bad factor decisions: use `as.is=TRUE`, then `factor()` as needed
   (d) Use `count.fields()` to see how R sees your data.
   (e) text contains unquoted apostrophe's: use `quote="\""`
   (f) text contains unquoted pound signs (hash marks): use `comment.char=""`
   (g) file contains NA codes: use, e.g., `na.strings=c("-99","-999")` or `na.strings="."`
   (h) non-ASCII file: check "foreign" package or "RExcell"

5. Writing functions
   (a) Usually takes the form:

```
foobar = function(foo, bar) {
  ...
  ...
  return(someValue)
}
```

   (b) Safest practice is to have all input as arguments.
   (c) Best practice is to have good defaults.
   (d) Best practice is to check input.
   (e) Comment, comment, comment. (But not totally obvious things.)
   (f) Indent sensibly and consistently.
   (g) Explicit `return()` line(s) are easiest to read.
   (h) It's OK to use `return(NULL)`.
   (i) It's OK to use, e.g., `return(list(a=5, reject=p<=0.05))`, but not `return(c(a=5, reject=p<=0.05))`.
   (j) Test with known input/output. Test some more.

6. Debugging
   (a) Insert `browser()` in your function, re-`source()` it, then use `where` to see what function you are in (including nesting), `n` to execute the next line, `c` for continue, or `Q` to quit the browser. Also e.g., `x` or `qt(x/y,3)` to display expressions,

as well as `ls()`, `print(n)`, and `sys.function()` to display the code for the function you are in. Also, e.g., `browser(expr=is.na(x))` will start the browser only if x equals NA.

(b) Use `debug(foo)`, followed by, e.g., `foo(arg1, arg2)`. Use the browser commands above to step through and examine the function from the beginning. Debugging remains active until the next R session or `undebug(foo)` or redefinition of "foo", e.g., with `source(foo.R)`.

(c) Use `trace(foo, exit=browser)` to stop in the browser just before "foo" returns whenever you call "foo". Or use `as.list(body(foo))` to see the internal numbering of "foo", then use, e.g., `trace(foo, tracer=browser, at=5)` to stop in the browser just before step 5 in "foo". Or use `trace(foo, edit=TRUE)` to get an editor into which you can insert your own browser() statement(s). The trace remains in effect, even across R sessions, until you `untrace(foo)` (which removes the browser statements) or until you redefine "foo".

7. `*apply()` family vs. looping

(a) R is a vectorized language. Loops can and should often be avoided. My favorite example is a student who brought me code that I vectorized in 6 minutes to reduce run time from 6 hours to 6 seconds.

(b) `apply(foo, index, bar)` works on matrix "foo" by applying the function `bar()` to each row (if index=1) or column (if index=2). Optional arguments can be passed to the function, e.g., `apply(foo, 1, mean, na.rm=T)`.

(c) `lapply(foo, bar)` works on list "foo" by applying function `bar()` to each element of the list, returning a list.

(d) `sapply(foo, bar)` works on list or vector "foo" by applying function `bar()` to each element of the list or vector, returning a vector if possible. Sometimes it's useful to use `foo=1:n` or `foo=rep(foobar, n)`, just to collect the results of "n" repetitions of a call to `bar()`, e.g.:

```
myFun = function(n, std=1) return(mean(rnorm(n,0,std)))
sd( sapply(rep(100,1000), myFun, std=5) )
```

(e) `tapply(foo, bar, foobar)` works on vector "foo", applying function "foobar", to each set of "foo" elements corresponding to a level of factor "bar".

```
> factor((1:30)%%3)
 [1] 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0
Levels: 0 1 2
> tapply(1:30, factor((1:30)%%3), mean)
    0    1    2
 16.5 14.5 15.5
```

8. Anonymous functions

   (a) Form: `function(x) {y=x^2; z=log(x); return(y/z)}`

   (b) Example 1:

```
sd( sapply(rep(100,1000), function(n) mean(rnorm(n,0,5)) ) )
```

   (c) Example 2:

```
> myMatrix = matrix(rnorm(500,3,4), 5)
> apply(myMatrix, 1, function(x) {
+                       m=mean(x); s=sd(x);
+                       return(c(mean=m, sd=s, cv=s/m))})
          [,1]     [,2]      [,3]      [,4]      [,5]
mean 4.4441972 3.786894 3.271766 3.090033 3.125422
sd   4.0784275 4.131218 4.046073 4.303494 3.899681
cv   0.9176972 1.090925 1.236663 1.392702 1.247729
```

9. Graphics

   (a) `dev.copy();dev.off()` vs., e.g., `pdf();dev.off()` (or win.metafile, png, jpeg, etc.)

   (b) Use `par(mar=c(,,,))` to increase margins and `cex.axis=` and `cex.label=` to make legible axes for posters and projected presentations.

   (c) Consider `axes=FALSE` with `plot()` followed by `axis()` to get highly customized axes.