STAT 237: Introduction to R

1 R basics

R is a programming language primarily used by statisticians. It is both a functional and object oriented language. R is free, and can be installed on both PCs and Macs and runs under Linux. A strength of R is that there are hundreds of ‘packages’ written by users. Packages contain functions that do special tasks. For example, ‘maanova’ is a package for analyzing micro-array data.

1.1 Installation

Download R from CRAN (Comprehensive R Archived Network): Google the letter ‘r’ and the first entry will take you to CRAN. Once you install R you can install the LearnBayes package.

1.2 References and getting Help

Good references for learning the basics:

- A First Course in Statistical Programming with R, (2007), Braun and Murdoch, Cambridge.
- Data manipulation with R, (2008), Phil Spector, Springer.
- A good web reference is http://zoonek2.free.fr/UNIX/48_R/02.html#2. Some of the material on this handout was taken from this page. Google “programming in R” and look for the the URL.
- The R reference card by Tom Short is a very helpful four page summary of R commands. Google “R reference card” to find it.

Getting help during an R session:

```
> help(median) or
> ?median
> Help.search("standard deviation") or
> ? "standard deviation "
```

Note that R is case-specific. > represents the R prompt.

1.3 Script editors

R can be used interactively, or you can write and run scripts, i.e. lines of R code that are run together. For writing scripts, I find it very useful to use an editor that matches brackets ('(','{' etc.), that recognizes generic R functions and that allows me to run my code directly from the editor. The editor that comes with the Mac installation of R has these features. I find the editor that comes with the PC installation of R inadequate. I use tinn-R, which is free. There are other options as well, for example EMACS and extensions for WinEdt.

2 Data Structures

R operates on tables of numbers. There are different kinds of tables: vectors (tables of dimension 1), matrices (tables of dimension 2), arrays (tables of any dimension), "Data Frames" (tables of dimension 2, in which each column may have a different data type – for instance, a table containing the results of an experiment, with one row per subject and one column per variable, where one variable may be numeric and the other character). The following presents more detail on constructing and manipulating vectors, matrices and data frames.

2.1 Assignment and data types

Assigning values:

```
> x=3 (or you can use the "<-" assignment operator: x <- 3)
> y=sqrt(x)+3    # this is a comment
> q=exp(1)
> v=log(y),
```

Vectors:

```
> y=1:5
> z=c(1,3,8,10,15)
> z=seq(1,11,2)
> z=rep(1:3,5)
```

Matrices:

```
> n=1:12   # n is a vector of dimension 1 x 12
> dim(n)=c(3,4)    # n is now a 3x4 matrix
> m=cbind(y,z)    # bind together two vectors column-wise
> matrix(1:12,nrow=3,byrow=T)    # result is same as n above
```

Reference the element in the 2nd row, 3rd column as m[2,3], elements in the second row as m[2,]
t(x) to transpose

IMPORTANT: In R, vectors do not have dimension. This differs from mathematics where vectors are either column or row vectors and have corresponding dimension. Try the following:

```
> m=matrix(1:12,nrow=3,byrow=TRUE)
> dim(m)
> a=m[,1]    # pick off the first column of the matrix m
> dim(a)
```

To retain the dimension, use ‘drop=FALSE’, as follows.

```
> a=m[,1,drop=FALSE]    # pick off the first column of the matrix m
> dim(a)
```

I have had many programs crash when, during computations, the dimension of either the rows or columns of a matrix was 1. The resulting dimension of the object is NULL unless you use ‘drop=FALSE’. This is a very irritating feature of R.

Character:

```
> y=c("yes","no","yes","yes")   # single quotes ok, too
```

Logical:

```
> y=x>3
```

Factors: The ‘factor’ function takes character or numeric vectors and converts them to data with mode ‘factor’. This is primarily useful when fitting regression models with a categorical predictor.

```
> pain=c(0,1,3,1,3,2,3)
> fpain=factor(pain,levels=0:3,labels=c("none","mild","moderate","severe")
```

Character variables in data frames are automatically stored as a factor.

2.2 Data Frames

A data frame is a matrix-like structure whose columns may be of differing types (numeric, logical, factor and character and so on). Use “read.table” to read in data in a table format into R. The resulting R object will be a data frame. Here’s how I read in the “studentdata” after I installed the LearnBayes package. Use forward slashes!

```
> a=read.table("C:/Users/Jeff/Documents/R/win-library/2.7/LearnBayes/data/
studentdata.txt",sep="\ t",header=TRUE)
> mean( a$Height )   # compute the mean of the variable ‘Height’
> attach(a)   # this allows us to refer to the variable using only the variable name
> mean( Height )
```

2.3 Lists

Lists are a key concept in R. A list is an object that contains one or more objects. The objects contained in a list can be of different types.

```
> x=c(2,3,4)
> y=c("yes","no","no","yes")
> z=matrix(1:100,nrow=10)
> b=list(x,y,z)
> b   # print out the entire list
> b[[1]]   # print out the first element of the list
```

3 Basic matrix operations

Here we describe how to do matrix multiplication and inversion. Matrix multiplication can be done using the '%*%' operator. Alternatively, the 'crossprod' function is often faster.

```
> x=matrix(c(1,2,3,4,5,6),3)   # x is a 3 x 2 matrix
> y=rnorm(9)
> dim(y)=c(3,3)   # y is a 3 x 3 matrix
> t(x) %*% y
```

or equivalently

```
> crossprod(x,y)
```

Elementwise multiplication is done using the '*' operator. For example, we can multiply the first column of x into each column of y as follows:

```
> x[,1] * y   # try this and see what happens
```

To invert a matrix, use the 'solve' function:

```
> inv.y=solve(y)
> inv.y %*% y   # should get the identity matrix back
```

4 Random variables

For a random variable with a given distribution, we can generate random variates, evaluate the density and distribution functions and compute quantiles using the prefixes: r=random, d=pdf , p=cdf, q=quantiles. For example, suppose $X \sim N(2,5)$ (X is normal, with mean 2 and variance 5). We generate n random variates from this distribution using

```
> rnorm(n,mean=2,sd=sqrt(5))
```

If we're interested in computing $\Pr(X < 2)$ we use

```
> pnorm(2,mean=2,sd=sqrt(5))
```

Random variables from a variety of distributions can be generated using the following functions: rnorm(n,mean,sd) rbinom(n,trials,p) , rgeom,rpois,rhyper,rexp,rgamma .

5 Control structures

```
if(...) {
...
} else {
...
}   # end if
```

Note that "..." represents R code.

```
for (i in 1:10) {
...
}   # end for
```

```
while(...) {
...
}   # end while
```

You can also use the 'apply' function and its variants to loop through certain calculations. The apply function can make code more compact, and *sometimes* will increase the speed of your program.

6 Functions

Functions are central to R, and learning to write functions is easy. Generally, functions take inputs, perform operations on the inputs, and return a value or list of values.

```
> cv=function(x) {sd(x)/mean(x)}   # or you can write
> cv=function(x) {return(sd(x)/mean(x)) }
> y=rnorm(1000);   cv(y)
```

Functions can only return one object. If you have multiple objects, return them in a list.

```
> m.sd=function(x){ m=mean(x); std=sd(x); return(list(m,std))}
```

Functions have NO SIDE EFFECTS: all the modifications are local. In particular, you cannot write a function that modifies a global variable. Conversely, it is wise to avoid using global variables to transmit data to functions. For example, suppose we create x and y as

```
> x=rnorm(1000); y=rnorm(1000)
```

Instead of

```
> mdiff=function(b) {return( mean(x-b) ) }
> mdiff(y)
```

use

```
> mdiff=function(a,b) {return( mean(a-b) ) }
> mdiff(x,y)
```

7 Saving your work

To save workspace with just objects you created: save.image("filename.RData")
To save just commands: Save in a script file (.R or .txt) or Save History.
To get commands back, either Load History, or type source("filename")
To save commands and output: click File/Save to File and save a txt file
To save graphs only: right click graph/ save as metafile (.emf) or copy/paste into a file.
To put output (not graphs) in a text file, type sink("filename"). Type sink() to stop.

8 Some useful generic functions

Descriptive Stats:
sd(), mean(), median(), max(), summary(y). For a vector: table(m) Contingency table: z=table(x,y) summary(z) gives chisq, etc. cor.test(x,y)

Graphs:
hist(), plot(), boxplot(), qqnorm(), pie(),

Get the R reference card for lots of additional functions.