A brief introduction to R

Cavan Reilly

September 4, 2019

Table of contents

Background

Most biostatisticians are trained in the use of R and SAS.

Academic statisticians increasing rely on R rather than SAS.

However SAS is still very widely used, especially in industry.

R is open source and there are no guarantees associated with its use.

R is a programming language

R is based on a piece of commercial software called S.

S was developed by Bell labs and saw widespread usage in the 1990s.

There are many add on packages that increase the functionality of "base R".

Using R

There are a variety of ways to use R:

- create text files with R commands: can run at a linux command prompt or run in an R window using *source*
- open an interactive R session (details of that are OS specific)
- use any of a number of graphical user interfaces designed for using R (e.g. R Studio)

We will use method 2 for this tutorial, I use method 1 for serious consulting work (most reproducible method).

R Objects

After installing, click on a program icon (or type R at linux command prompt).

R holds a variety of objects in active memory during an R session: use `ls()` to see.

While you can create your own types of objects, for typical statistical applications the following are the most common

1. integers and floats (i.e. numbers)
2. vectors
3. matrices
4. character strings
5. factors
6. lists

Operations on objects

You can do different operations on different types of objects: you can add numbers, vectors and matrices.

You can also create variables that hold these objects-here we use the assignment operator and the c function to create a vector.

```
> 3+4
[1] 7
> a=3+4
> a
[1] 7
> b=c(4,7,3)
> b
[1] 4 7 3
```

Operations on objects

You can get information about and modify charater strings in a variety of ways. Here is a vector that holds character strings: note that we can use *subscripts* to access elements of the vector.

There are some functions that are useful for working with character strings: here `nchar` gets the number of characters.

```
a=c("string 1","string 2")
> a[1]
[1] "string 1"
> a[2]
[1] "string 2"
> nchar(a[1])
[1] 8
```

Factors

A factor is a commonly used data type that is used to encode a categorical variable.

We can create factors from character strings.

```
> gender <- c("Male","Male","Female","Female","Female")
> gender
[1] "Male"   "Male"   "Female" "Female" "Female"
> gender <- factor(gender)
> gender
[1] Male   Male   Female Female Female
Levels: Female Male
```

There are also functions that are specific to factors, for example here is how to determine the number of levels of a factor.

```
> nlevels(gender)
[1] 2
```

Reading data into R

There are a variety of functions that can read data from an external file.

R will attempt to read data in from its current working directory: to find out what that is you can type

```
> getwd()
[1] "/home/merganser/faculty/cavanr/SIBS/SIBS2016"
```

You can also see what files are there by typing

```
> list.files()
 [1] "2016-SIBS-Application.docx"
 [2] "2016-SIBS-Application-File-Template.docx"
```

Reading data into R

If you have an excel file, first save it as a csv file in excel then read into R using the `read.csv` function.

Other programs will allow you to save data as tab delimited data: then use the `read.delim` function (there is also `read.table`).

Files with irregular formating can be difficult to read in: useful functions in these cases are `readLines` and `scan`.

If you type

```
> ?readLines
```

you can read the help page for that function.

Dataframes

R uses dataframes to store datasets-the previous commands for reading in data create a dataframe.

R will try to convert data from the file to the appropriate format: there will frequently be conversions to factors and this can be confusing

```
> as.numeric(factor(c("2","2","1","2","1")))
[1] 2 2 1 2 1
> as.numeric(factor(c("3","3","1","3","1")))
[1] 2 2 1 2 1
```

Writing results to file

You can write results to an external file using `write`, or the more user friendly `write.table`:

```
> mat1=matrix(c(3,2,6,5,7,3),ncol=2)
> write.table(mat1,"temp-file.txt",row.names=F,
+ col.names=F,quote=F)
```

There is also a very useful package called `xtable` that allows generation of nice tables using the text editor latex.

Generating figures

While using the interactive version of R from a window, one can create plots directly.

```
> x1=c(4,7,2,5,4)
> x2=c(6,3,7,5,9)
> plot(x1,x2)
```

There are many types of figures (histograms, boxplots and scatterplots) and even more ways to customize these figures.

Generating figures

I generally want to create a file that has a figure-this then gets incorporated into another document.

To do this you first issue a command to start up a graphics device: for example to create a pdf file with our previous file you type `pdf` the use `dev.off` to generate the file.

```
> pdf("figure1.pdf")
> plot(x1,x2)
> dev.off()
```

Issuing these commands will create a pdf file called figure1.pdf in your current working directory.

There are also similar commands for generating jpeg, tiff, bmp and png files.

Missing data

In real life, data sets have missing values: in R this is represented with the symbol NA.

This symbol is a special character that is not a character string:

```
> NA
[1] NA
> MA
Error: object 'MA' not found
```

There is also the special character Inf:

```
> 1/0
[1] Inf
> Inf+Inf
[1] Inf
> Inf-Inf
[1] NaN
```

Random numbers

One can also create random numbers-this is useful for randomization but also for scientific computing.

To simulate draws from a normal distribution you use `rnorm`, for the t-distribution you use `rt` and for the binomial you use `rbinom`.

```
> mean(rnorm(10))
[1] -0.2722795
> mean(rnorm(100))
[1] -0.03682329
> mean(rnorm(1000))
[1] -0.03143072
> mean(rnorm(10000))
[1] -0.01100216
> mean(rnorm(100000))
[1] 0.003214296
```

Control structures

When you analyze data there are often a sequence of steps that one follows to complete the process-these sorts of recipes are called algorithms.

Control structures determine how an algorithm works.

For example: determining which subjects are eligible in PREVAIL IV.

For eligibility, need at least 1 positive sample in last 42 days...

Control structures

1. Start with a list of participant IDs.
2. Get all visit dates for each participant.
3. Compute number of days from visit to date of randomization.
4. If this is less than or equal to 42 then the visit "counts".
5. If a participant has 2 visits that count and at least 1 positive sample, then add that participant's ID to the list of eligible participants.

Control structures

We can use a for loop to go through the list of PIDs.

We can use if statements to see if something is true.

```
evd_pcr=read.delim("http://www.biostat.umn.edu/~cavanr/
+ evd_pcr052016.txt")
tdy=as.numeric(as.Date("5/20/16",origin="01/01/70",format="%m/%d/%y"))
cnt=0
for(i in 1:length(unique(evd_pcr$PID))){
  id=which(evd_pcr$PID==unique(evd_pcr$PID)[i])
  dts=evd_pcr$SPECDT[id]
  dts=as.numeric(as.Date(dts,origin="01/01/70",format="%m/%d/%y"))
  tmsd=tdy-dts
  if(length(tmsd)>1 & sort(tmsd)[1]<42 & sort(tmsd)[2]<42) cnt=cnt+1
}
> cnt
[1] 20
```