EPIB-613 - Four Lecture Overview of R

R is a package with enormous capacity for complex statistical analysis.

We will see only a small proportion of what it can do.

The R component of EPIB-613 is divided into four lectures:

1. Introduction
2. Data Entry and Manipulation
3. Basic Analysis and Graphics
4. Programming and Functions

1 Introduction to R

In today's introduction, we will cover:

1. R background
2. How to download and install R
3. Various ways to put commands into R
4. Using R as a calculator
5. How to get help in R
6. Data types and basic functions applied to these data types

1.1 R background

Quoted directly from the R WWW page (http://www.r-project.org/)

> "R is a language and environment for statistical computing and graphics. It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R.
>
> "R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, ...) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity.
>
> "One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.
>
> "R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License in source code form. It compiles and runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS."

In short: R is “free Splus”, a very sophisticated statistics and graphics package, used by most statisticians these days for modern analysis and statistical research.

2 Download & Installation

Installation follows a few trivial steps:

1. Go to R web page, at http://www.r-project.org/
2. On the left hand side menu on the screen, click on “CRAN” which is under the “Download” item.
3. Pick a country site from which to download. For example you can click on one of the Canadian sites, but really you can pick any, all this effects is download speed.
4. Near the top of the page (e.g., http://cran.stat.sfu.ca/), inside the box labeled “Download and Install R’, pick the right file to download, depending on your operating system (e.g., click on “Download R for Windows” if you have any Windows system, and so on).
5. This brings you to a page where you select the part of R you need. While you may later want to download the set of user contributed functions, for now just click on “base”, which gets you the basic R program (very extensive by itself).
6. At the next screen, click on “Download R 2.13.1 for Windows” (or similar). At this point you should be asked (via a prompt box) where you want to save the file. Pick a place on your computer to save this (e.g., c:\temp).
7. After file has downloaded, in your explorer (in Windows, use equivalent if you have another operating system), find where you have saved it, and click on “R-2.13.1-win.exe” (or similar).
8. Follow directions on screen until program is installed. In general, you can just leave all options unchanged.
9. After installation is completed, click on the “R” icon on your desktop to begin the program. At this point, your screen should have an R prompt on it, and you are ready to begin using the program.

2.1 Adding Packages

You can easily extend R's basic capabilities by adding in functions that others have created and made freely available through the web. Rather than surfing the web trying to find, them, R makes adding these capabilities trivially easy by using the "Packages" menu items.

For example, to add a new package not previously downloaded onto your computer, click on "Packages $\longrightarrow$ Install Package(s)", pick a site from the list (usually a Canadian site) and choose the package you want to install from the long list (currently, well over 1000 packages are available). To activate this package, you then need to click on it in the 'Packages $\longrightarrow$ Load Package" menu item. You are then ready to use it, and the help files for the package will also be automatically added to your installation. See, for example, the "Help $\longrightarrow$ HTML help" menu item, and click on "Packages" in the browser window that opens.

We will see one package next class, for reading and writing excel files. Basic R, without adding any packages is extremely powerful by itself. Inevitably, however, something you need for analysis will be missing, but available as an add-on package. Since R is the main package used by statisticians to develop new methods, they usually first appear in R, often simultaneous with their first publication in the statistical literature.

3 Getting commands into R

There are various ways to get commands into R:

- Type directly into the command line interactively.
- Use the file menu item to open a script, create the required program lines, then cut and paste into R as needed, or use the Edit $\longrightarrow$ Run line or selection (or Run all) menu command.
- Use any text editor to type commands into R, then cut and paste into R command line. Similar to using scripts, but not necessarily built into R.
- Save a file, and use the source command at the R prompt

  ```
  > source("c:\\temp\\myfile.txt")
  ```

  where myfile.txt is a plain text file containing a series of R commands.
- Use the Rcmdr, which will pop up a graphical interface. This is found in the Rcmdr add-on package Not good for all jobs. Better to learn commands without GUI help, as not all commands are available in the windows. However, useful for simple analyses without a need to remember many commands.

Depending on the purpose of the R session, one of these methods may be optimal, but for many jobs all five will work, so depends in part on your preferences. Saving your commands for re-running or re-using with or without modifications is generally useful, though.

4 R Basics

R can be used in many ways:

Use R as a Calculator:

```
> 2+3
[1] 5
> pi
[1] 3.141593
> 2+3*pi
[1] 11.42478
> log(2+3*pi)
[1] 2.435785
> exp(2.435785)
[1] 11.42478
> 6/7
[1] 0.8571429
> cos(pi)
[1] -1
> 4^5
[1] 1024
```

And so on. All the usual calculator functions are built-in, and, of course, much more.

5 Getting help on any R function

To get help on any R function (including add-on packages, once they are loaded)

```
> help(sum)
```

and a help window will pop up, in this case for the function “sum”. R also comes with extensive manuals, html help files, a nice introduction, etc. Each help file contains a lot of information, but most useful sections include the description of the command and what it calculates, the definition of each parameter input and their order, and at the bottom., some examples of command use.

For example:

```
sum {base}
Sum of Vector Elements

Description

sum returns the sum of all the values present in its arguments.

Usage

sum(..., na.rm = FALSE)

Arguments
...  numeric or complex or logical vectors.
na.rm  logical. Should missing values be removed?

Details

This is a generic function: methods can be defined for it directly or
via the Summary group generic.  For this to work properly, the arguments ...
should be unnamed, and dispatch is on the first argument.

If na.rm is FALSE an NA value in any of the arguments will cause a value of
NA to be returned,  otherwise NA values are ignored.

Logical true values are regarded as one, false values as zero. For historical
reasons, NULL is accepted and treated as if it were integer(0).

Value

The sum. If all of ... are of type integer or logical, then the sum is integer,
and in that case the result will be NA (with a warning) if integer
overflow occurs. Otherwise it is a length-one numeric or complex vector.

NB: the sum of an empty set is zero, by definition.

S4 methods
```

```
This is part of the S4 Summary group generic. Methods for it must use the
signature x, ..., na.rm.

plotmath for the use of sum in plot annotation.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) The New S Language.
Wadsworth & Brooks/Cole.

See Also

colSums for row and column sums.
```

Quick reminder of the parameters of any function - the args command:

```
> args(rbinom)
function (n, size, prob)
> args(matrix)
function (data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)
> args(cor)
function (x, y = NULL, use = "everything", method = c("pearson",
    "kendall", "spearman"))
```

6 Data types and functions on data types

There are many types of variables in R. Each entry in any data item can be a numeric, character, logical, factor or a date. These can be stand alone scalars, or combined into vectors, matrices, data frames, and lists.

Scaler variables:

```
> a <- 3
> a
[1] 3
> b <- 5
> b
```

```
[1] 5
> b-a
[1] 2
> b/a
[1] 1.666667
```

Logical variables

The following logical operators are defined:

```
! x   [not x]
x & y [x and y]
x | y [x or y]
```

For example:

```
> c <- T
> d <- F
```

```
> c & d
[1] FALSE
> c | d
[1] TRUE
```

Vector variables:

```
> x<-c(2,3,1,5,4,6,5,7,6,8)
> x
 [1] 2 3 1 5 4 6 5 7 6 8
> y <- c(10, 12, 14, 13, 34, 23, 12, 34, 25, 43)
> y
 [1] 10 12 14 13 34 23 12 34 25 43
> x[4]
[1] 5
> y[3]
[1] 14
> x[1:10]
 [1] 2 3 1 5 4 6 5 7 6 8
```

```
> x[6:8]
[1] 6 5 7
> x[x > 5]
[1] 6 7 6 8
```

Functions on vectors:

```
> length(x)
[1] 10
> length(y)
[1] 10
> sum(x)
[1] 47
> sum(y)
[1] 220
> sum(x^2)
[1] 265
> mean(x)
[1] 4.7
> mean(y)
[1] 22
> var(x)
[1] 4.9
> var(y)
[1] 136.4444
> sqrt(var(x))
[1] 2.213594
> sqrt(var(y))
[1] 11.68094
> sum((x-mean(x))^2)
[1] 44.1
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    3.25    5.00    4.70    6.00    8.00
> summary(y)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  10.00   12.25   18.50   22.00   31.75   43.00
> summary(x^2)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00   10.75   25.00   26.50   36.00   64.00
> cor(x,y)
[1] 0.7176257
```

Matrices:

```
> z<- matrix(c(1,2,3,4,5,6,7,8,9), nrow=3, byrow=T)
> z
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> z.transpose<- matrix(c(1,2,3,4,5,6,7,8,9), nrow=3, byrow=F)
> z.transpose
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> z[2,3]
[1] 6
> z[,1]
[1] 1 4 7
> z[1,]
[1] 1 2 3
> z[3,]
[1] 7 8 9
```

You can also have character variables:

```
> sex <- c("male", "female", "female", "male", "male", "female", "female")
> sex
[1] "male"   "female" "female" "male"   "male"   "female" "female"
> smoking <- c("yes", "no", "no", "yes", "no", "yes", "no", "no", "yes", "no")
> smoking
 [1] "yes" "no"  "no"  "yes" "no"  "yes" "no"  "no"  "yes" "no"
```

You can declare these variables to be factor variables for regression analyses:

```
> sex.factor<- as.factor(sex)
> sex.factor
[1] male   female female male   male   female female
Levels: female male
```

Dates in R

Dates are represented as the number of days since 1970-01-01, with negative values for earlier dates.

Below are some useful functions for creating and manipulating dates.

```
> some.dates <- as.Date(c("2009-09-22", "2005-03-14"))
>  some.dates
[1] "2009-09-22" "2005-03-14"
>  days <- some.dates[1] - some.dates[2]
>  days
Time difference of 1653 days
```

The function as.Date( ) can be used to convert character data to dates. The format is as.Date(x, ”format”), where x is the character data and format gives the appropriate format.

An example:

```
> strDates <- c("01/05/1965", "08/16/1975")
> strDates
[1] "01/05/1965" "08/16/1975"
> dates <- as.Date(strDates, "%m/%d/%Y")
> dates
[1] "1965-01-05" "1975-08-16"
```

The default format is yyyy-mm-dd:

```
> mydates <- as.Date(c("2007-06-22", "2004-02-13"))
> mydates
[1] "2007-06-22" "2004-02-13"
```

You can convert dates to character data using the as.Character( ) function.

```
# convert dates to character data
> strDates <- as.character(dates)
> strDates
[1] "1965-01-05" "1975-08-16"
```

Lists

Lists are like vectors, but instead of only having single numerical entries, you can have anything.

```
> a <- 1:10
> b <- "yes"
> c <- matrix(c(1,2,3,4), byrow=T, nrow=2)
> a
 [1]  1  2  3  4  5  6  7  8  9 10
> b
[1] "yes"
> c
     [,1] [,2]
[1,]    1    2
[2,]    3    4
> mylist <- list(a, b, c)
> mylist
[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10

[[2]]
[1] "yes"

[[3]]
     [,1] [,2]
[1,]    1    2
[2,]    3    4
```

Lists are useful for storing output from functions, for example regression output, which has many different types (regression coefficient,s residuals, variable names, etc.).

You can also give names to items within lists:

```
> mylist.named <- list(name1=a, name2=b, name3=c)
> mylist.named
$name1
 [1]  1  2  3  4  5  6  7  8  9 10

$name2
[1] "yes"
```

```
$name3
     [,1] [,2]
[1,]    1    2
[2,]    3    4

> mylist.named$name2
[1] "yes"
```

Note that once a list has names, you may refer to items within that list by their name using the $ notation, as shown above. This is useful in output such as from the `lm` command, as each output from the command will be separately available (such as $residuals, $coefficients, and so on).

Random number generation

It is trivially easy to generate random numbers in R, useful for all kinds of statistical analyses and simulations:

```
> x<-rnorm(10, mean=3, sd=2)
> x
 [1] 2.135011 7.573118 3.840814 5.544555 3.393277 2.731562
 [6] 1.780588 4.576343 3.250077 3.908082
> y<-rbinom(10, size=1, prob=0.4)
> y
 [1] 0 1 1 1 0 0 1 0 1 0
> z<- 5 + 2*x + 3*y +rnorm(10, mean=0, sd=0.5)
> z
 [1]  9.44732 23.30763 15.34688 19.62479 11.30383 10.89258
 [6] 11.22488 13.33018 15.11767 12.81894
```

Data Frames:

Most data sets are entered into R as data frames. To create a data frame, you can enter:

```
> my.data.frame<-data.frame(x,y,z)
> my.data.frame
         x y        z
1  2.135011 0  9.44732
2  7.573118 1 23.30763
3  3.840814 1 15.34688
```

```
4  5.544555 1 19.62479
5  3.393277 0 11.30383
6  2.731562 0 10.89258
7  1.780588 1 11.22488
8  4.576343 0 13.33018
9  3.250077 1 15.11767
10 3.908082 0 12.81894
```