Lab 1: R Tutorial

Nora Mitchell

January 2017

Contents

1 Introduction **1**
1.1 Installing R 2
1.2 Installing RStudio 2

2 Getting Started **2**
2.1 Writing & Running Code 2
2.2 Basic Math 3
2.3 More Basic Functions 4
2.4 Common R Objects 4
2.5 Installing Packages 6

3 Analyzing Data **7**
3.1 Changing Working Directory 7
3.2 Importing Data 7
3.3 Manipulating and Saving Data 8
3.4 Graphics in R 9

4 Example Problem Set **9**

5 JAGS Introduction **10**

6 References **11**

1 Introduction

R is a powerful, open-source program used for statisics and graphics. R uses its own programming language and base, but much of its power lies in special "packages" that users can write to perform detailed, field-specific analyses. R can also communicate with other programs (such as JAGS), making analyses easier to run on both PC and Mac systems.

RStudio is one of a number of programs which interface with R (and the one I will be using throughout the course). Feel free to use your own if you wish.

1.1 Installing R

R can be installed from `http://www.r-project.org/`. You will have to select a specifc CRAN (Comprehensive R Archive Network), scroll down to USA and select the Statlib one from Carnegie Mellon. Choose the appropriate download for your OS, and install either the "base" R v3.1.2 for Windows or the correct version for your Mac OS.

1.2 Installing RStudio

You can now open R, but you will see only a console window. You can type and run lines of code in here, but it is not user friendly and is difficult to work and save materials in. Therefore, we wil install RStudio.

Download RStudio (**open-source desktop version**) at `http://www.rstudio.com/products/rstudio/download/`. Once you have done this, open RStudio. It should automatically communicate with R

You may wish to play around with the windows within RStudio. These include:

- main window "script": for editing text/writing code
- console: displays code actually run (plus output)
- a window displaying all variables, functions, values, etc. currently in use
- a window for viewing files in your working directory, plots, packages installed, and help documentation

2 Getting Started

Learning to use R takes a lot of time and love, but it is also incredibly valuable and the new standard in ecology and evolutionary biology. This tutorial will walk you through some of the very basics. A lot of learning R is looking at documentation for individual packages or doing Google searches.

If you're familiar with R, you may wish to skip this section.

2.1 Writing & Running Code

A few basics before getting started. We will be working in the main window writing our code.

- To actually run a line of code, you highlight the line and hit CTRL+Enter (or apple enter on a Mac), or click the "Run" icon. The line which has run, and any output, will then be displayed in the active console.
- To comment out lines (type in notes to yourself that won't actually be run), use the pound sign #, followed by any text. R will not read this as code. Hint: It's best to keep your code well-documented, so that you can come back to it months later and actually know what you did/what you were thinking.
- **functions** take a set of **arguments** (input by the user) and return an **object**. Functions can be pre-written and contain code that transforms the arguments into the output. You can also write your own functions (woo!)
- Help in R: If you have a question on a specific package or function, simply run a line of code with a ? and then the function's name. The documentation for that will open under the "Help" tab in RStudio.

  ```
  input: ?mean
  ```

  Check the help console: it will now have information about the function "mean".

2.2 Basic Math

Let's start very basic. Keep the same script (main window), as we may reuse some pieces in later examples. In your script, type

```
4+8
```

and hit CTR+Enter (or Run). In the console, it should now say

```
> 4+8
[12]
```

You can also store results as variables in R, which can be used later. You can use either = or <- for assignment. I typically prefer <-, since this has a broader scope (ask me for details if you want).

```
a <- 15 + 16
a
```

Run both lines of code. You're output should look like:

```
[1] 31
```

You can also store multiple variables and then do math on them

```
b < -23
c <- 42
d <- b*c
d
```

Output:

```
[1] 966
```

2.3 More Basic Functions

There are a number of built-in mathematical function in R, including constants, basic transformations, trigonometry, and different statistical distributions (which we will go over later in more detail).

Basic Math in R	
abs(x)	absolute value of x
cos(x), sin(x), tan(x)	cosine, sine, and tangent of angle x
exp(x)	exponential function
log(x)	natural (base-e) logarithm
log10(x)	common (base-10) logarithm
mean($x_1, x_2, ..., x_n$)	takes the average of included numbers
range(...)	returns the minimum and maximum of the given arguments
pi	constant π
rnorm(x)	x random numbers from normal distribution N(0,1)
runif(x)	x random numbers from uniform distribution U(0,1)
sqrt(x)	square root of x

2.4 Common R Objects

R has several different types of objects, with different attributes to them. We'll *very briefly* go over a few here. There is a TON that you can do with these, feel free to explore on your own. When naming objects, it is important to remember that R is case sensitive, and object names cannot begin with numbers!

- **Vectors** are 1-dimensions arrays of numbers, and can be used in calculations like a normal number

  ```
  input: x <- c(1:8)
         x
  output: [1] 1 2 3 4 5 6 7 8
  ```

  Here, the 'c' is a generic function that combines elements. '1:8' generates a list from 1 to 8, going up by 1. There are many other ways to make vectors, including '*seq*' and '*rep*'.

- **Matrices** are 2-dimensional arrays of numbers

  ```
  input: y <- matrix(c(1,2,3,4,5,6),2,3)
  ```

```
        y
output: > y
              [,1] [,2] [,3]
        [1,]  1    3    5
        [2,]  2    4    6
```

Here we've made made a matrix consisting of 6 values, with 2 rows and 3 columns. It does this by column automatically, but you can use byrow=T to change this

```
input: z <- matrix(c(1,2,3,4,5,6),2,3, byrow=T)
        z
output: > z
              [,1] [,2] [,3]
        [1,]  1    2    3
        [2,]  4    5    6
```

- **Data frames** are extremely common and useful. Data frames have multiple columns of equal-length vectors, and rows of individuals or observations. We'll explore some example datasets in data frame format in the next section. For now, here are some important functions we use with data frames:

Data Frames	
data.frame($x, y, ...$)	combines x, y into a data frame
names(x)	access the column names in the data frame x
as.data.frame(x)	converts an object into a data frame
attach(x)	includes columns of x in the workspace
head(x)	view first few rows of x
tail(x)	view last few rows of x
summary(x)	gives 5-number summaries and NA's for data frame x
str(x)	gives the internal structure of data frame x

Let's do a quick example of creating a data frame from 3 separate pieces on the Buffalo Bills running backs:

```
input:
        first <- c("LeSean", "Mike", "Reggie")
        last <- c("McCoy", "Gillislee", "Bush")
        age <- c(28, 26, 31)
        running_backs <- data.frame(first, last, age)
        running_backs
        str(running_backs)
```

Run all lines of code. You should be able to see that you've made a data frame with 3 columns and 3 rows: 2 of the columns are factors and the last is a number. To get the data from just one column, use the $and the column name. You can also use this to add a column name:

input:

```
running_backs$last
running_backs$weight <- c(208, 219, 205)
str(running_backs)
```

You should see that the first line returns just the last names, and then you've added a column for each player's weight. Go Bills!

str() is especially useful, because it tells you what kind of data type each column is (integer, numeric, character, factor, etc.)

2.5 Installing Packages

R comes pre-installed with the {base} package, but again, the sky's the limit! You have to install other packages. RStudio makes it especially easy to install packages in two ways-

1. Use 'Tools' from the main toolbar.
   Go to 'Tools'-> 'Install Packages'. Search for a package by beginning to type its name. Here we'll install 'R2jags'. Make sure 'Install Dependencies' is checked–many packages depend on functions from other packages to work. Click 'Install' and it should work-though it may take a few seconds to finish.

2. Use the 'Packages' tab in the open window.
   In the 'Files, Plots, Packages, Help' window, select 'Packages'. Then click 'Install Packages', and follow the directions as above.

Now you have the packages installed. To actually use them, you have to **call** them

```
library(R2jags)
```

...and the package (and all of its functions) is ready to use!

3 Analyzing Data

3.1 Changing Working Directory

Download the .csv files from the webpage and place them in a folder of your choice on your laptop. In order to access these files, you have to navigate so that R has access to that **working directory** (folder). You will then have access to files in those folders, and any files you save will also be in that folder. To do this:

Go to 'Session'-> 'Set Working Directory' -> 'Choose Directory' and navigate to the folder with your files in it.

Your console will now show a shortened path to your folder

    setwd(" /EEB 5348/Lab1 RTutorial")

You can also set your working directory to 'Source file location', if you have saved your R script in a particular location. You can also type in the *setwd* code itself and save it for when you open this R script again in the future.

3.2 Importing Data

Now that you've navigated to the proper working directory, you can import data saved in various file formats, including text or tab-delimited files in .csv or .txt format, or more specific formats like .nex or .tre. We will be working mostly with .csv or .txt files, so I will use one of these as an example

To read in a csv file, we use

read.csv("filename.csv", header=TRUE, na.strings=".")

The header portions means that the first row of the data is a header, and that missing values are indicated by a period ".". You can edit this if need be, but this is fairly standard. There are corresponding functions to read in other data file types, such as *read.table.*
Let's read in an example data set on some plant measurements I made on two different species in one location in South Africa. Use different data frame functions to explore the data a little bit.

    input: protea<-read.csv("Blesberg_Data.csv", header=TRUE, na.strings=".")

Take a look at the "Workspace" console–you should now see several different Values and Data objects with brief descriptions. You can click on these and a new window will open to view the data.

3.3 Manipulating and Saving Data

Once your data is in a workable format, the possibilities are endliess. Here are just a few basic things you can do with it:

- *length*(x) takes the length of an object (x)
  For a vector or matrix, it will return the number of values, for a data frame it will usually give the number of columns. To get number of rows, you can use length(data$column.name)

  ```
  input: length(x)
         length(protea)
         length(proteas$Plant_ID)
  ```

  How do the lenghts for the data frame differ?

- *rep* (x, ...) replicates the value (x) '...' times
  (x) can be a number of character, for instance:

  ```
  input: rep(5,6)
  output: [1] 5 5 5 5 5 5

  input: rep("tuna", 3)
  output: [1] "tuna" "tuna" "tuna"
  ```

  Note that the quotes around 'tuna' make it a character, not an object.

- *sort* (x, ..., decreasing=FALSE) sorts the data in an increasing fashion

  ```
  input: sort(running_backs$weight, decreasing=FALSE)
         sort(running_backs$weight, decreasing=TRUE)
  ```

- *cbind* and *rbind* bind by column or row (vectors, matrices, or data frames
  An example with vectors:

  ```
  input: cbind(first, last, age)
         rbind(first, last, age)
  ```

- *tapply* (X, INDEX, Function,...) applies a function to certain groups or subsets of data. This is a little more complicated, but can, for instance, find the range in values for a specific variable for each group. Let's check the range of values for FWC (fresh water content) for each species of *Protea*

  ```
  input: tapply(protea$FWC, protea$Species, range)
  ```

This will find the range of values for FWC for each species. Run the code, what do you get for the range of *punctata* values? Going back to the data frame, you can see that there are a number of missing values for FWC in *punctata*. Some functions require a specific instruction for what to do with NAs. Let's try removing them.

```
input: tapply(protea$FWC, protea$Species, range, na.rm=TRUE)
```

That should look better.

Real data is almost always complicated and messy. There are a ton of ways to analyze your data in R, especially using packages. Be prepared to learn on the fly!

3.4 Graphics in R

We could dedicate the rest of our lives to this. There are a ton of ways to graphically represent data, here we will just get you used to using the graphics in {base}, although each package will often have its own special graphing features, and the package {*ggplot2*} is all the rage these days. If we need to be able to do certain advanced graphics, we will walk through it. Else, try and use the internet to search for how to make basic graphs!

I will say, that once you have made a plot it will appear in the "Plots" window. You can then click "Export" ->Save Plot as Image, select an image type, name it, and it will then save in your working directory for later.

4 Example Problem Set

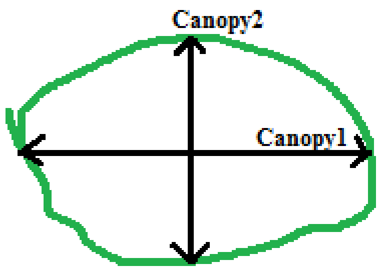
Here is an easy example to accustom yourself to these basic mathematical functions, reading, and writing data files. Download the data files "punctata", "venusta", "cynaroides", and "repens" from the class website. Each contains list of canopy measurements made on plants in the field. The data file name indicates a different species of *Protea*, and columns in the list refer to the plant number, diameter of the widest part of the plant (Canopy1), and the orthogonal diameter (Canopy2), which are used to estimate the entire canopy area of an individual plant (how big is it looking down in 2D space?).

Your job is to:

1. Take the four separate files and combine them into one data frame

2. Calculate the canopy areas for each individual plant using the equation for an ellipse

$$Area = \pi \times \frac{Canopy1}{2} \times \frac{Canopy2}{2}$$



3. Create side-by-side boxplots for the canopy areas for each species. We have not gone over this during lab: try and look it up on the internet! Is it easy to compare the four different species?

4. Challenge: If you know how, make these boxplots using ggplot2@

5. Try a log (base-e) transformation and make another boxplot.

6. Calculate the species averages and put them in a final data frame and save this as '"Species_means.csv". Hint: Take all four means at once using *tapply*!

Please show me your boxplots, "Species_Means.csv" file, and code before you move onto the JAGS portion! Thanks!

5 JAGS Introduction

Kent will be using some basic JAGS code in the next few days. We will be going over the statistical theory behind JAGS in more detail next week in lab, but let's play with the code today anyway!

First you need to install JAGS

- Step 1: Install JAGS `http://mcmc-jags.sourceforge.net/` R libraries "rjags" and "R2jags".
- Step 2: Download and and save the code from Lab 1 (all into the same folder).
- Step 3: Navigate to the correct working directory, open up and run the code for the binomial model (".R" file!).
- Step 4: Get output and look through it

6 References

Much of this tutorial was inspired by Dr. Paul Hurtado's tutorial from the 2013 Joint MBI-NIMBioS-CAMBAM Summer Graduate Workshop at the University of Tennessee in Knoxville, TN.

- Hurtado, Paul. 2013. An introduction to programming in R. Joint 2013 MBI-NIMBioS-CAMBAM Summer Graduate Workshop

- R Development Core Team. 2004. R: A language and environment for statistical comput- ing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

- RStudio. 2012. RStudio: Integrated development environment for R (Version 0.96.122) [Computer software]. Boston, MA.