BISC-869, Quantitative Methods

January 18, 2022

Introduction

About me

Introductions

Web-site: https://www.sfu.ca/~lmgonigl/qm.html

Syllabus

Course aims

Primary goal: Get comfortable working with and analyzing biological data

“Secondary” goal: Become R savvy

R

History

Base R vs RStudio/Tidyverse

R Pros

- Packages
- Scripts
- Functions
- Online help
- Great built in graphics (produces vectorized images)
- Great for handling data
- Good for making your work reproducible
- Can be interfaced with other langaugues (e.g., C, Fortran, etc)

R Cons

- Difficult to learn
- Many ways to do the same thing
- Some seemingly easy tasks can be difficult

Workflow & data management

Most of the time spent “analyzing” data is often spent “cleaning” and “preparing” the data for an analysis.

The “structure” of a dataset can provide insights into how you should structure an analysis.

Once you’ve got your raw data entered into a spreadsheet, don’t touch it again. Do this all in R. This gives you a full history of all the data processing you’ve done.

Data recommendations

1. Use a scripted program such as R for both data prep and analysis
   - Command-line programs are harder to learn but reduce problems in future.
   - Menu-based programs leave no record of the analyses you carried out. You will forget. Menus change.
   - Scripts become your written records of both what you did to the data and how you conducted your analyses.
   - Annotate script files with detailed comments explaining your actions.
2. Store data in a nonproprietary format
   - For example: use comma delimited text files, .csv.
   - Text files can always be read, whereas proprietary formats can become difficult to work with in the future.
   - You can still use spreadsheet applications to develop the text files (Google Sheets, Excel, etc.).
   - Avoid spaces in filenames (use underscores instead).

Data recommendations

3. Leave your data file uncorrected
   - Otherwise you might change something that you later discover was correct.
   - Corrections directly to the data file go unrecorded – you have no record of the change you made.
   - Make corrections instead using the scripted language (R) so that you have a record, and can undo later if necessary.
   - Keep comments in your script (command) file that explains reasons for corrections, so you can redo or re-evaluate later.
4. Use plain ASCII text for names and data values
   - Avoid the use of special symbols when entering data.
   - Avoid commas because they separate fields in .csv format
   - Avoid special symbols (e.g., α, , $^\circ$C, etc)

Data recommendations

5. When you add data to a database, add rows not columns
   - Set up data files to maximize consistency of column content
   - Use “long” format rather than “wide”.
     Wide:

lizard	sprintSpeed1984	sprintSpeed1985
1	1.43	1.37
2	1.56	1.30
3	1.64	1.36
...		

Long:

lizard	Speed	Year
1	1.43	1984
2	1.56	1984
3	1.64	1984
1	1.37	1985
2	1.30	1985
3	1.36	1985
...		

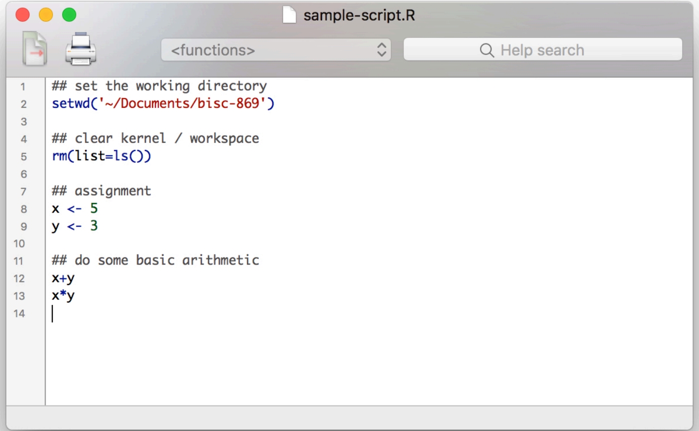
6. Record full dates using standardized ISO format
   - For dates use YYYY-MM-DD

     or use different columns for year, months, day, time, etc
   - Note that Excel will sometimes reformat dates

Data recommendations

7. Create a relational database
   - Put separate information collected at different scales into different files.
   - For example: One file for SITE data (temperature, elevation). Another file for measurements of SPECIES collected at those sites. Both files contain the SITE variable, allowing data to be matched as needed (using `match` or `merge` commands in R).
   - SQL is an advanced tool for managing relational databases and can be used from within R using the 'RSQLite' package (we could add to a workshop later, if people are interested).
8. Maintain effective metadata (data about the data)
   - Ten years from now you won't remember what the site looked like, which sample you dropped, or how you assigned a single value for "depth" of a pond.
   - Write down details of methods.
   - Include names of all files associated with the study, definitions for data and treatment codes, missing value codes, definitions, unit of measurement for each variable.

Structure of an R script



```r
## set the working directory
setwd('~/Documents/bisc-869')

## clear kernel / workspace
rm(list=ls())

## assignment
x <- 5
y <- 3

## do some basic arithmetic
x+y
x*y
```

Workflow for analysis of data

Typically I have three types of scripts for an analysis

1. clean a data-set
2. prepare data for an analysis
3. run analysis

It is also often convenient to have additional scripts that contains other code you want to run (e.g., functions you will use). These can be loaded directly into R using the `source` command.

Functions

R has many built in functions. For example

`exp(x)`, `log(x)`, `sin(x)`

If you want to know how to use a function, you can load the help with, for example

```
?exp
```

Some can get quite advanced and take a while to figure out

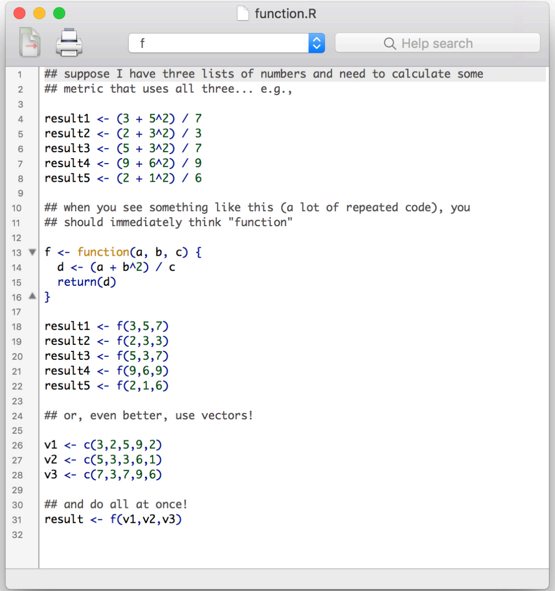
`match(...)`, `split(...)`, `apply(...)`

You can load packages to get even more!

```
library('vegan')
```

But, best of all, you can write your own functions. This is something you should start doing early and frequently.

Functions



```r
## suppose I have three lists of numbers and need to calculate some
## metric that uses all three... e.g.,

result1 <- (3 + 5^2) / 7
result2 <- (2 + 3^2) / 3
result3 <- (5 + 3^2) / 7
result4 <- (9 + 6^2) / 9
result5 <- (2 + 1^2) / 6

## when you see something like this (a lot of repeated code), you
## should immediately think "function"

f <- function(a, b, c) {
  d <- (a + b^2) / c
  return(d)
}

result1 <- f(3,5,7)
result2 <- f(2,3,3)
result3 <- f(5,3,7)
result4 <- f(9,6,9)
result5 <- f(2,1,6)

## or, even better, use vectors!

v1 <- c(3,2,5,9,2)
v2 <- c(5,3,3,6,1)
v3 <- c(7,3,7,9,6)

## and do all at once!
result <- f(v1,v2,v3)
```

A few extras about functions

R, by default, returns the last line of a function, so

```
f <- function(x) {
  y <- 3*x + 2
  return(y)
}
```

is the same as

```
f <- function(x) {
  y <- 3*x + 2
  y
}
```

which is the same as

```
f <- function(x) {
  3*x + 2
}
```

A few extras about functions

A function can have multiple arguments. Note that I have dropped curly braces here.

```
f <- function(a, b)
  3*a + b
```

You could call this function using any of

```
f(4,5)
f(a=4,b=5)
f(b=5,a=4)
```

You can also specify default values.

```
f <- function(a=2,b=7)
  3*a + b
```

The following call would be evaluated with a=2, and b=11

```
f(b=11)
```