R Workshop, Session 2
Packages, project organization, and notes on formatting

1. Packages

Many programs include all packages with the initial program installation by default, but R does not. Instead, when you first install R, the program itself is installed along with a set of "base" packages. These packages are simply text files with the ".R" extension that contain functions to perform specific tasks. Every time R is opened, these "base" packages are loaded ("sourced") and the contents of the scripts are submitted to your current R session.

There are thousands of additional packages available on the CRAN website to do all kinds of specialized analyses (phylogenetic analyses, ecological analyses, special plotting functions, etc.). These packages are not automagically downloaded with R, but you can download them yourself. To see what's available, check http://cran.r-project.org under Packages.

To install a package:

```
install.packages("package-name")  # the quotation marks are necessary
# R will prompt you to choose a CRAN mirror site; select one that's nearby.
```

To source a package:

```
library("package-name") # Now all the functions in the package are available for your use.
```

2. Project Organization

Why?

Facilitate future work: good organization makes it easier to pick up where you left off the next time you return to a project

Collaborations are easier because you can send a single folder to colleague(s) for modification.

Exact repeatability is easier by you or others in the near or distant future.

Saves you time plowing through files and folders, trying to remember where things are and what you did last.

Three elements for good organization:

1. **Standard folder and file hierarchy**. Try to strike a balance between having enough folders to organize a project, but not so many that you spend a lot of time navigating your folder hierarchy. House each project in a folder, with subdirectories for different aspects of the project:
2. **Foolproof data storage**. Data are sacrosanct. It is far too easy to accidentally alter or destroy any or all of your data, so you need to have good strategies for preventing data

corruption. I keep data in two forms: a spreadsheet where it's easy to initially enter the data, and a text file (tab- or comma-delimited) which is easy to load into R, but difficult to edit. *Separate your data and analyses into two entirely separate entities*. R allows you to load in and mess with data in any number of ways without altering the original data.
3. **Well-documented R scripts.** These allow you to repeat any analyses at any time with your current data, modified versions of that data, or even new datasets.

General organizational practices:

Set the working directory first
Then, load any libraries
Then, add in any project-specific functions
Then, load data, run analyses, and make figures.

Your goal is to optimize legibility and functionality. Balance between the number of scripts and script length/complexity (e.g. you may want separate scripts for figures and functions). It's generally best to have a main project script that functions as a "parent" script – the only one you need to open because all other scripts are called by it. But if you use this approach, be sure to document in your subscripts what they depend on and where they are called from!

3. A few brief notes on formatting…

R is very flexible, so it is useful to adopt guidelines to make your code easier to return to over time, and to share with others. I recommend finding general style guides to develop your own formatting standards so your code is legible and easier to share with others. Below is a brief outline of suggestions derived from Google's R Style Guide: https://google.github.io/styleguide/Rguide.xml

A. File names should end in .R and be meaningful
    a. Good: evaluate_risk.R
    b. Bad: foo.R
B. Variable names should be meaningful. Don't use pre-existing function names!
    a. ex. variable.name or variableName
C. Use action verbs for function names, and capitalize the first letter.
    a. Good: GetWeights
    b. Bad: Weights
D. Spacing: Put spacing around all binary operators (`=, +, -, <-`, etc…). Place one space after a comma (but not before a comma).
    a. Extra spaces can be helpful for aligning separate lines of code. Avoid tabs!
E. Assignment: ALWAYS use `<-` for assignment, NOT `=` !!
    a. Use an equal sign for arguments within a function
        i. `normdist <- rnorm(1, mean=10, sd=1)`
F. # Comment. # Your. # Code. # Short comments on the same line. Two spaces after code, then `#` space `Comment`.
    a. # Longer comments on a separate line.

Consistent style allows your collaborators to focus on *what* you are saying, not *how* you are saying it.

4. Where to go for help

Built-in documentation

R has a comprehensive built-in help system. The help documentation may seem unfriendly at first, but it is all organized in the same way. Familiarize yourself with the structure of the help documentation to make the most of it.

```
help(foo)    # Help about function foo
?foo         # Same thing.

help.search("fooish")  # If you don't remember the exact function name
??fooish               # Same thing

apropos("foo")  # List all functions containing the string foo
example("foo")  #Show an example of function foo
```

Some packages also have vignettes for how to use the package:

```
vignette()  # Show available vignettes
vignette("foo")  # Show specific vignette for foo
```

Help from other human beings

Stack Overflow: http://stackoverflow.com

A question-and-answer site. Input search terms to narrow down to specific questions and examples, or to determine whether anyone has answered your question before. Many people are helpful, but may get snarky if it becomes clear that you have not clearly described your question/problem or looked around at all for a previous answer.

There is also an R-help e-mail list: http://stat.ethz.ch/mailman/listinfo/r-help

Read the FAQ before posting. The archives are also searchable.

Create your own R Users Group as needed!