Practical R for MBM- Syllabus

Chuck Frost

6/7/2020

Practical R for MBM field data manipulation and analysis

The purpose of this ongoing short course is to familiarize MBM personnel with the concepts of tidy data, basic data manipulation, and basic programming techniques as implemented in R. It is expected that participants have a working understanding of R and RStudio. We will not go into depth on other types of programming, specifically statistical programming and computer programming. The topics covered here will foster appropriate treatment of MBM data throughout the data life cycle and lead to cleaner and more useful data in the future.

Topics

1. R, RStudio, packages, help
2. Loading and saving, data types
3. Visualization
4. Working with dates
5. Looping and functions
6. Tidy data

What is R?

An open-source programming environment that serves as:

1. A giant calculator
2. An interface for data analysis and visualization
3. An interface for data manipulation and file handling
4. An interface for a simple and efficient programming language (S, that became R)

R isn't the "method." For example, "We used R to estimate population size." [yuck]

What is RStudio?

An integrated development environment (IDE) for R.

Provides tools, menus, help, and easy access to the best parts of R.

##Packages:

Base R automatically provides you with many common functions.

```
mean(c(1,2,3))

sum(c(1,2,3))

var(c(1,2,3))
```

Other useful functions exist thanks to R users. Collections of related functions get wrapped into packages.

If you're looking for something NOT in base R, you need to install the related package.

The CRAN provides R community tested and approved packages. Check the packages tab in RStudio for some you already have installed but (probably) not loaded. Just click them to load. If you know what package you need, there are 2 easy ways to install it. The first is by clicking the install button in the packages tab. Search the CRAN for the package you want and install it. The second is:

```
install.packages("PackageName")
```

Don't forget to load the package after installation, either by checking the box under packages, or by calling:

```
library(PackageName)
```

Note that in install.packages() you are searching a string in quotes, "PackageName" while in library() you are calling a package object without quotes, PackageName.

##Installing from GitHub

If a package isn't uploaded to CRAN, it can still be made available through GitHub. Packages installed from GitHub require an intermediate step:

```
install.packages("devtools")

library(devtools)

install_github("USFWS/AKaerial", ref = "development")

library(AKaerial)
```

This code does 4 things: 1. Install devtools package 2. Load devtools package 3. Call install_github (from devtools), look for the GitHub account "USFWS" and repository "AKaerial" and load the package in the "development" branch 4. Load AKaerial package

Installing and loading AKaerial may take a while since it includes dependencies. Dependencies are packages that are specified within another package that must be included for that package to operate appropriately. Akaerial "depends" on several other packages and will check if you have these installed.

##Functions

Once you have a package loaded, you can access functions written and included in that package. Typing the name of a function will spit out the code that makes up the function. This can be messy for a long or complicated function. Typing ?FunctionName will access the help file for a given function.

```
AdjustCounts

?AdjustCounts
```

This can be extremely helpful (if the help file is!) when troubleshooting why your code isn't working as you want it to. Note that the top of the help file also tells you FunctionName {PackageName} in case you can't figure out where your function is coming from. This could be the case if you have dozens of packages loaded or have borrowed someone else's code to help run your analysis. Don't underestimate the power of internet searches to find if something already exists to help you do your thing!

We will talk about functions in more depth later, but for now, the general idea is that functions take arguments, run processes, and (usually) return products saved as R objects.

```r
numbers = c(1,2,3)

avg = mean(numbers)
```

In this example, numbers is initiated as a vector of 3 integers: 1,2,3. We then run the function mean with numbers as its argument, and save the result as the object avg.

It can be helpful to think of functions as recipes and arguments as ingredients.

```r
Mix = function(what.to.mix){
  dough = sum(what.to.mix)
  return(dough)
  }

Bake = function(what.to.bake, time.to.bake){

  cookies = what.to.bake * time.to.bake

  return(cookies)

}


my.ingredients = c("sugar", "eggs", "butter", "flour")

my.dough = Mix(my.ingredients)

my.cookies = Bake(my.dough, 10)
```