Writing Effective R Functions

Jasmine Dumas

September 20, 2016

## Tips & Methodology

* If you need to write the same piece of R code more than once it is best to write it as a function.
* If the code is domain or application specific it still might make more sense to write a function. This can help with code clutter and ease readability.
* Functions are best when they perform **one** process really well.
* Functions should only have there output depend on there input.
* Functions should not change the environment in which it resides.
* If the function is able to be widely generalized, think about placing it in a package along with other functions.
* R packages exist in many forms being CRAN, GitHub, and internal (in order of greatest ease of access).
* R packages are the the fundamental units of shareable R code which include documentation and examples
* Sharing code does not have to be the goal of creating effective functions but will enhance collaboration among teams.
* Instead of emailing functions, point teammates to a centralized folder location with a relative path.
* If the functions don't belong to a package, it may not be necessary to add **FULL** [Roxygen](http://roxygen.org/) style documentation but it will ease the transition to developing a R package if those comments are available.
* In writing documentation for functions, it is worthwhile to include:
  + what the function does?
  + what type or class of data the function works on?
  + what the successful output of the function looks like?
* Naming a function can be hard. Its only job is to quickly give the programmer an idea of what it does. Feel free to use any style but **be consistent**! Options for style include: [camelCase](https://en.wikipedia.org/wiki/CamelCase), [snake\_case](https://en.wikipedia.org/wiki/Snake_case), and many [more](https://en.wikipedia.org/wiki/CamelCase#See_also). Also long and descriptive *beats* short and confusing names!
* Not everything should be in functions - or rather some R functions are intended to be only utilized in a interactive setting, which brings up the discussion of NSE vs. SE.
* [Non-Standard Evaluation](http://adv-r.had.co.nz/Computing-on-the-language.html) (or NSE) are the incorporation of a shortcut method for R in specifying the column names from data objects, which reduces redundancy as opposed to Standard Evaluation (or SE) which expect that the data object name is specified, such as when using the $ notation. An example of this is subset(). In the example below notice how we only specified the iris data frame once and not in the second argument slot? It would be best to use the traditional bracket notation when needing to subset a data frame within a function.
* I would also not be so cavalier when mixing NSE functions in shiny applications, as things can become even more confusing with the shiny functions. The tidyverse packages (aka hadleyverse) make use of NSE function but do have compliment functions usually denoted by and suffix \*\*\_\*\* such as count() and count\_().

# the subset function is for interactive use, so it does not know about other object names unrealted to the data.frame  
nse\_func <- function(df = iris, col = "Sepal.Length", ...){  
 subset(df, col < 4.4)  
}  
  
nse\_func()

## [1] Sepal.Length Sepal.Width Petal.Length Petal.Width Species   
## <0 rows> (or 0-length row.names)

# this bracket notaion is perfered for this function  
se\_func <- function(df = iris, col = "Sepal.Length", ...){  
 df[df[col] < 4.4, ]  
}  
  
se\_func()

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
## 14 4.3 3 1.1 0.1 setosa

## Functions in Practice

* To source in a function or read R code from a file or a connection, use the source() function
* To source in a function to have it used as part of the local environment, for example in sections of a shiny application use the local = TRUE argument
* Functions should be sourced in at the very top of a script for debugging purposes
* keep argument names consistent to remove confusion try not to name them after already existing objects unless that is for testing
* Its useful to use the return() function inside the function when you want something to be returned to the environment for further use and there by associate a new name with the object. If you forget you can easy get the last called value by using this function: variable.name <- .Last.value

## Further reading

* <http://r-pkgs.had.co.nz/>
* <http://r4ds.had.co.nz/>
* <https://en.wikipedia.org/wiki/Functional_programming>
* <https://en.wikipedia.org/wiki/Pure_function>
* <https://www.r-bloggers.com/how-to-write-and-debug-an-r-function/>
* <http://www.statmethods.net/management/userfunctions.html>

fin.