Data Science + Public Policy

Jeffrey Chen

2018-12-28

0.1 Functions

Data manipulation tasks are often repeated for many different projects and it is not uncommon for two or more scripts to contain the same exact steps, but the code is hardcoded. Same logic and different variables names equates to a significant amount of time spent editing and modifying programs.

Rather than tediously modifying programs, try to write your code once, then never again. Each set of code can serve as re-usable tools that can be re-applied to similar problems, but only if it is standardized with well-laid logic. This is the basis of *user-defined functions*: a coder can define some set of standard required inputs on which a set of steps can be applied to produce a standard output.

A typical function is constructed as follows. Using `function`, a set of input parameters are specified as placeholders for any kind of object. For example, `df1` represents a data frame and `var1` is a variable name in string format. Within the curley brackets, we insert code treating the parameters of actual data. In the example below, we calculate the mean of `var1` in data frame `df1`, then assign it to a `temp.mean`. These calculations are executed in a *local environment*, meaning that any calculations steps within the function are temporary. Thus, `temp.mean` is wiped once the function finishes. The `temp.mean` object can be extracted by passing it to `return`. All of the above steps are assigned to the `meanToo` object that is treated like any other function.

```
meanToo <- function(df1, var1, ...){

  #Code goes here
  temp <- mean(df1[[var1]])

  #Return desired result
  return(temp)
}
```

To execute the function, we will simply need to call the function with a data frame and a variable name. Basically any script can be genericized into a standardized function.

```
meanToo(data, "x1")
```