Writing Functions In R - a practical example: customized output table for Simple Linear Regressions

Martin Schweinberger

December 28, 2017

Functions are an extremely powerful feature of r especially as they can easily be written and customized. Very generally, functions are objects in R which can be called to perform customized calculations or tasks. Functions typically require arguments on which they perform some action.

The general form of a function is ¡em¿function(argument¡sub¿1¡/sub¿, argument¡sub¿2¡/sub¿, . . . , agument¡sub¿n¡/sub¿)¡/em¿. To exemplify what this means and how you can write your own function, I am going to write a function which will produce a customized output table for a simple linear regression model. But before we write a rather complicated function, we are writing a very simple example, so you understand the basic logic behind it. We will write a function called my.summary which reports the mean, the median, and the standard deviation for a numeric vector.

```
### writing a very simple function
my.summary <- function(x)
{
  ifelse(is.numeric(x) == FALSE, return("ERROR: vector must
     be numeric"), x == x)
    m1 <- mean(x)
    m2 <- median(x)
    sd1 <- sd(x)
    result <- list(m1, m2, sd1)
    names(result) <- c("Mean", "Median", "Standard Dviation")
    return(result)
}

### --- Test our function
```

```
14
15 # create avector
16 test <- c(1,2,3,4,5,6,7,8,9)
17 my.summary(test)
18
19 # below is the output of our function
20 #> $Mean
21 #>[1] 5
22 #>
23 #>$Median
24 #>[1] 5
25 #>
26 #>$`Standard Dviation`
27 #>[1] 2.738613
```

Our simple example function works and return the mean, the median, and the standard deviation when we apply it to a numeric vector.

Now for the more complex example: The way that our more complex function will work is that the "summary" function for a regression model will produce a list which contains several objects. Our function will then extract some of these objects from the regression summary and reorganize them into a neat table.

The table will consist out of 7 columns and 8 rows. We are thus creating 8 vectors with 7 elements each. Empty elements will be represented as empty cells.

I am going to annotate the code to make it as clear as I can so that you can follow what I do.

So, here we go...

As a first step, we clean our workspace

```
1 # Remove all lists from the current workspace
2 rm(list=ls(all=T))
```

We are going to start right away by creating an object which we call "lm.summary" and we define it as a function with a single argument (x). Th argument is a lm object so a simple linear regression model - to make it more comprehensible, you can simply think of x as standing for
`lm(prep.ptw  year, data = slrm.data)`

```
1 lm.summary <- function(x) {
```

Next, we are defining another function within the main function which will produce summaries of p-values. So instead of the actual value, the `p.nice`-function will output the level of significance of the p-value. For example, if the p-value is .03, the p.nice function will output "p¡.05*".

```
  p.nice <- function(z) {
    as.vector(unlist(sapply(z, function(w) {

# the line below reads: if the value is lower than 0.001
   print "p<.001***"
      ifelse(w < .001, return("p < .001***"),

# the line below reads: if the value is lower than 0.01 print
    "p<.01**"
      ifelse(w < .01, return("p < .01**"),

# the line below reads: if the value is lower than 0.05 print
    "p<.05*"
# else print the exact value (of w)
      ifelse(w < .05, return("p < .05*"), return(w)))) } )))
          }

# the next section will create a vector called "intercept"
# with 6 elements which we are going to extract from the
# summary of the lm object. This vector will represent a row
   in
# the table with the parametsr of the coefficient, i.e. its
# estimate, standard error, t-value, and p-value.

  intercept <- c(

# the first element of "intercept" is the first element of
   fourth object
# of the summary list. And because this element is a
   coefficient
# estimate, i.e. a numeric value, we are going to round it
# to 2 decimal places.
    round(summary(x)[[4]][1], 2),

# the next element is empty because it will be in a column
   showing
# standardized betas and we do not get standardized betas for
    the
# intercept in a linear regression model.
    "",
    round(summary(x)[[4]][3], 2),
    round(summary(x)[[4]][5], 2),
    round(summary(x)[[4]][7], 4),
    p.nice(summary(x)[[4]][7]))
```

```
36
37 # the next vector will hold the parameters for the predictor
38   predictor <- c(
39
40 # extract the estimate for the predictor and round the value
41 # to 4 decimal places
42     round(summary(x)[[4]][2], 2),
43
44 # extract the standardized beta for the predictor and round
    the value
45 # to 4 decimal places
46     round(lm.beta(x)[[1]], 4),
47
48 # extract the standard error for the predictor and round the
    value
49 # to 4 decimal places
50     round(summary(x)[[4]][4], 2),
51
52 # extract the t-value for the predictor and round the value
53 # to 4 decimal places
54     round(summary(x)[[4]][6], 2),
55
56 # extract the p-value for the predictor and round the value
57 # to 4 decimal places
58     round(summary(x)[[4]][8], 4),
59
60 # apply the p.nice function to the p-value and return the
    nice p-value
61     p.nice(summary(x)[[4]][8]))
62
63 # in the fowllowing , we create vectors which hold only one
64 # character string as the last element: we now add a row
65 # to the table which separates the rows for the predictor
66 # from the overall model statistics
67
68 # first, we create a vector which will serve as a header for
69 # the model statz
70   mdl.statz <- c("", "", "", "", "", "Value")
71
72 # we are now extracting the number of cases in our model
73   nbcases <- c("", "", "", "", "", length(summary(x)[[3]]))
74
75 # extract the residual standard error
76   rse <- c("", "", "", "", "",
77     round(summary(x)[[6]], 2))
78
79 # extract the common R<sup>2</sup> value
80   multR2 <- c("", "", "", "", "", round(summary(x)[[8]], 4))
81
```

```
# extract the adjusted R<sup>2</sup> value (this R<sup>2</sup> value
# provides an estimate of how much the common R<sup>2</sup> value
# will vary, if we the cross-validate it, i.e. how much variation there
# would be, if we took many samples and replicated the model many times
  adjR2 <- c("", "", "", "", "", round(summary(x)[[9]], 4))

# extract the F statistic of the overall model
  F <- c("", "", "", "", "",
    round(summary(x)[[10]][1], 2))

# extract the p
  p <- c("", "", "", "", "", round(summary(x)[[4]][8], 4))

# we are almost done: now, we are going to create a table
# called <em>slrm.tb</em> by binding the vectors together
# (the command/function <em>rbind</em> means: bind
# vectors as rows together
  slrm.tb <- rbind(intercept, predictor, mdl.statz, nbcases,
     rse, multR2, adjR2, F, p )

# now, we add column names to the table
  colnames(slrm.tb) <- c(colnames(summary(x)[[4]])[1],
    "Std. Beta",
    colnames(summary(x)[[4]])[c(2:4)],
    "P-value sig.")

# now, we add rownames to the table
  rownames(slrm.tb) <- c(
    rownames(summary(x)[[4]])[1],
    rownames(summary(x)[[4]])[2],
      "Model statistics", "Number of cases in model",
      paste("Residual standard error", paste("on", summary(x)
         [[7]][2],"DF")),
      "Multiple R-squared", "Adjusted R-squared",
      paste("F-statistic",
        paste("(", round(summary(x)[[10]][2], 0), ", ",
          round(summary(x)[[10]][3], 0), ")", sep = "",
             collapse = "")),
      "Model p-value")

# we converting the table into a data frame
  slrm.tb <- as.data.frame(slrm.tb)

# now we tell R to print the resulting table to the console, once
```

```
# it is done with this prodedure.
  return(slrm.tb)
}
# voila: we are done with writing a function which outputs
# a customized summary table for a simple linear regression
   model
```

The next issue is how we can utilize our function whenever we want. R makes that really easy: all we need to do is to define the source (the path to the function). So you only need to save the above code in an editor (I use TinnR as it highlights R syntax) and save it as an R file (I saved the code above as ¡em¿slr.summary.tb.R¡/em¿) in my directory for customized R functions. The path thus is "C:
R /slr.summary.tb" (I need to use double backslashes in front of directories as my machine runs on Windows). The only thing you need to add is the ¡em¿source¡/em¿ command which tells R that it is supposed the function from the path which is its argument.

So let' check how to call and execute our function.

```
### Calling functions

# clean our workspace
rm(list=ls(all=T))

# Initiate packages we need
library(QuantPsyc)
library(car)

# load our function
source("C:\\R/slr.summary.tb.R")
# now we read in some data
slr.data <- read.delim("D:\\MyProjects\\
   SimpleLinearRegression/slr.data.txt", header = TRUE)

# remove columns we do not need
slr.data <- as.data.frame(cbind(slr.data$datems, slr.data$P.
   ptw))
colnames(slr.data) <- c("year", "prep.ptw") # add column
   names
slr.data <- slr.data[!is.na(slr.data$year) == T, ] # delete
   incomplete cases

prep.lm <- lm(prep.ptw ~ year, data = slr.data)

# call our function
slr.summary(prep.lm)
```

The output of the function that is displayed on the R GUI will look slightly different from the table below but has the same structure and outline (I transformed it for editorial purposes since LaTeXwill not display the function output as it is shown in the R GUI.

	Estimate	Std. Beta	Std. Error	t value	Pr(> |t|)	P-value sig.
(Intercept)	102.09		10.86	9.4	0	p< .001 ∗ ∗∗
year	0.02	0.1039	0.01	2.56	0.0107	p< .05∗
Model statistics						Value
Number of cases in model						603
Res. std error (601DF)						21.11
Multiple R-squared						0.0108
Adjusted R-squared						0.0091
F-statistic (1, 601)						6.55
Model p-value						0.0107