STAT 408 - Week 5: R Miscellanea and Debugging R Code

February 8, 2018

Course Goals

With this class, we cannot cover every possible situation that you will encounter. The goals are to:

1. Give you a broad range of tools that can be employed to manipulate, visualize, and analyze data, and
2. teach you to find help when you or your code "gets stuck".

R Miscellanea - more tools

Lists

We have used lists (some), but it is worth talking about them in more details. Here are some questions to get started.

- Where have lists shown up in this course?
- How do we typically index elements in a list?
- What other functions have we used to manipulate lists?

Exercise: Lists

Consider the two lists, write out what gets printed from R.

```
msu.info <- list( name = c('Waded Cruzado','Andy Hoegh'),
         degree.from = c('University of Texas at Arlington','Virginia Tech'),
         job.title = c('President', 'Assistant Professor of Statistics'))
msu.info

msu.info2 <- list(c('Waded Cruzado','University of Texas at Arlington',
                    'President'), c('Andy Hoegh',
                  'Virginia Tech','Assistant Professor of Statistics'))
msu.info2
```

What do all of those brackets mean?

Solution: Lists

```
msu.info <- list( name = c('Waded Cruzado','Andy Hoegh'),
         degree.from = c('University of Texas at Arlington','Virginia Tech'),
         job.title = c('President', 'Assistant Professor of Statistics'))
msu.info
```

```
## $name
## [1] "Waded Cruzado" "Andy Hoegh"
##
## $degree.from
## [1] "University of Texas at Arlington" "Virginia Tech"
##
## $job.title
## [1] "President"                         "Assistant Professor of Statistics"
```

```
msu.info2 <- list(c('Waded Cruzado','University of Texas at Arlington',
                    'President'), c('Andy Hoegh',
                  'Virginia Tech','Assistant Professor of Statistics'))
msu.info2
```

```
## [[1]]
## [1] "Waded Cruzado"                     "University of Texas at Arlington"
## [3] "President"
##
## [[2]]
## [1] "Andy Hoegh"                        "Virginia Tech"
## [3] "Assistant Professor of Statistics"
```

Lists - indexing

With the current lists we can index elements using the double bracket `[[ ]]` notation or if names have been initialized, those can be used too.

So the first element of each list can be indexed

```
msu.info[[1]]
```

```
## [1] "Waded Cruzado" "Andy Hoegh"
```

```
msu.info$name
```

```
## [1] "Waded Cruzado" "Andy Hoegh"
```

Exercise: Lists

Explore the indexing with these commands.

```
msu.info <- list( name = c('Waded Cruzado','Andy Hoegh'),
         degree.from = c('University of Texas at Arlington','Virginia Tech'),
         job.title = c('President', 'Assistant Professor of Statistics'))
msu.info[1]
msu.info[[1]]
msu.info$name[2]
msu.info[1:2]
unlist(msu.info)
```

Solution: Lists 1


```r
msu.info[1]
```

```
## $name
## [1] "Waded Cruzado" "Andy Hoegh"
```

```r
msu.info[[1]]
```

```
## [1] "Waded Cruzado" "Andy Hoegh"
```

```r
msu.info$name[2]
```

```
## [1] "Andy Hoegh"
```

Solution: Lists 2


```r
msu.info[1:2]
```

```
## $name
## [1] "Waded Cruzado" "Andy Hoegh"
##
## $degree.from
## [1] "University of Texas at Arlington" "Virginia Tech"
```

```r
unlist(msu.info)
```

```
##                                name1                                name2
##                       "Waded Cruzado"                          "Andy Hoegh"
##                          degree.from1                          degree.from2
##  "University of Texas at Arlington"                       "Virginia Tech"
##                           job.title1                           job.title2
##                          "President" "Assistant Professor of Statistics"
```

Lists - nested lists


```r
list(list('a','b'),list('c','d'))
```

```
## [[1]]
## [[1]][[1]]
## [1] "a"
##
## [[1]][[2]]
## [1] "b"
##
##
## [[2]]
## [[2]][[1]]
## [1] "c"
##
## [[2]][[2]]
## [1] "d"
```

Arrays

Arrays are a general form a matrix, but have a higher dimension.

```
array.1 <- array(1:8, dim=c(2,2,2)); array.1
```

```
## , , 1
##
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## , , 2
##
##      [,1] [,2]
## [1,]    5    7
## [2,]    6    8
```

```
array.1[2,2,1]
```

```
## [1] 4
```

Exercise: Arrays

Create an array of dimension 2 x 2 x 3, where each of the three 2 x 2 subarray (or matrix) is the Identity matrix.

Solution: Arrays

Create an array of dimension 2 x 2 x 3, where each of the three 2 x 2 subarray (or matrix) is the Identity matrix.

```
array(c(1,0,0,1), dim = c(2,2,3))
```

```
## , , 1
##
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
##
## , , 2
##
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
##
## , , 3
##
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

Merge

Another important skill is merging or combining data sets.

Consider the two data frames, how can we merge them and what should be the dimensions of the merged data frame.

```
df1 <- data.frame(school = c('MSU','VT','Mines'),
            state= c('MT','VA','CO'), stringsAsFactors = F)
df1
```

```
##   school state
## 1    MSU    MT
## 2     VT    VA
## 3  Mines    CO
```

```
df2 <- data.frame(school = c('Mines','MSU','VT'),
            enrollment = c(5794,15688,30598), stringsAsFactors = F)
df2
```

```
##   school enrollment
## 1  Mines       5794
## 2    MSU      15688
## 3     VT      30598
```

sort() and order()

One possibility is to use the `sort()` / `order()` functionality as a first step.

```
order(df1$school)
```

```
## [1] 3 1 2
```

```
order(df2$school)
```

```
## [1] 1 2 3
```

```
df1 <- df1[order(df1$school),]
df1
```

```
##   school state
## 3  Mines    CO
## 1    MSU    MT
## 2     VT    VA
```

```
df2 <- df2[order(df2$school),]
df2
```

```
##   school enrollment
## 1  Mines       5794
## 2    MSU      15688
## 3     VT      30598
```

RBIND() AND CBIND()

Now, given that the data frames are both sorted the same way, we can bind the rows together.

```
comb.df <- cbind(df1,df2)
comb.df
```

```
##   school state school enrollment
## 3  Mines    CO  Mines       5794
## 1    MSU    MT    MSU      15688
## 2     VT    VA     VT      30598
```

```
comb.df <- comb.df[,-3]
```

rbind() and cbind()

Now assume we want to add another school to the data frame.

```
new.school <- c('Luther', 'IA',2337)
rbind(comb.df, new.school)
```

```
##   school state enrollment
## 3  Mines    CO       5794
## 1    MSU    MT      15688
## 2     VT    VA      30598
## 4 Luther    IA       2337
```

Note: if your strings are saved as factors, this chunk of code will give you an error.

JOIN()

We could have also used some of the more advanced merge (join) features from dplyr.

```
library(dplyr)
new.df <- full_join(df1,df2, by='school')
new.df
```

```
##   school state enrollment
## 1  Mines    CO       5794
## 2    MSU    MT      15688
## 3     VT    VA      30598
```

Exercise: merging

Combine the two data sets

```
df.cost <- data.frame( ski.resort = c('Bridger Bowl', 'Big Sky', 'Steamboat', 'Jackson'),
                       ticket.cost = c(60, 'depends',145, 130))
df.acres <- data.frame( ski.hill = c('Bridger Bowl', 'Jackson', 'Steamboat', 'Big Sky'),
                        skiable.acres = c(2000, "2500+",2965, 5800))
```

Solution: merging

Combine the two data sets

```
df.cost <- data.frame( ski.resort = c('Bridger Bowl', 'Big Sky', 'Steamboat', 'Jackson'),
                       ticket.cost = c(60, 'depends',145, 130))
df.acres <- data.frame( ski.hill = c('Bridger Bowl', 'Jackson', 'Steamboat', 'Big Sky'),
                        skiable.acres = c(2000, "2500+",2965, 5800))

kable(full_join(df.cost, df.acres, by = c('ski.resort' = 'ski.hill')))
```

ski.resort	ticket.cost	skiable.acres
Bridger Bowl	60	2000
Big Sky	depends	5800
Steamboat	145	2965
Jackson	130	2500+

Debugging R code

Process for writing code

When writing code (and conducting statistical analyses) an iterative approach is a good strategy.

1. Test each line of code as you write it and if necessary confirm that nested functions are giving the desired results.
2. Start simple and then add more complexity.

Debugging Overview

Finding your bug is a process of confirming the many things that you believe are true – until you find one which is not true. - Norm Matloff

Debugging Guide

We will first focus on debugging when an error, or warning is tripped.

1. Realize you have a bug (if error or warning, read the message)
2. Make it repeatable
3. Identify the problematic line (using print statements can be helpful)
4. Fix it and test it (evaluate nested functions if necessary)

Warnings vs. Errors

R will flag, print out a message, in two cases: warnings and errors.

- What is the difference between the two?
- Is the R process treated differently for errors and warnings?

Warnings vs. Errors

- Fatal errors are signaled with `stop()` and force all execution of code to stop triggering an `error`.
- Warnings are generated with `warning()` and display potential problems. Warnings **do not** stop code from executing.
- Messages can also be passed using `message()`, which pass along information.

Bugs without warning/error

In other cases, we will have bugs in our code that don't necessarily give a warning or an error.

- How do we identify these bugs?
- How can we exit a case where:
  - R is running and may be stuck?
  - the code won't execute because of misaligned parenthesis, braces, brackets?

Note: `NA` values often return a warning message, but not always.

Exercise: Debugging a Warning

Fix the script that determines if each item in a sequence is less than zero.

```
val.in <- seq(-1,1,by=.25)

if (val.in < 0){
  print(paste(val.in, 'less than 0'))
}
```

```
## Warning in if (val.in < 0) {: the condition has length > 1 and only the
## first element will be used
```

```
## [1] "-1 less than 0"    "-0.75 less than 0" "-0.5 less than 0"
## [4] "-0.25 less than 0" "0 less than 0"     "0.25 less than 0"
## [7] "0.5 less than 0"   "0.75 less than 0"  "1 less than 0"
```

Solution: Debugging a Warning


```r
val.in <- seq(-1,1,by=.25)

ifelse(val.in < 0,paste(val.in, 'less than 0'),paste(val.in, 'greater than (equal to) 0'))
```

```
## [1] "-1 less than 0"                 "-0.75 less than 0"
## [3] "-0.5 less than 0"               "-0.25 less than 0"
## [5] "0 greater than (equal to) 0"    "0.25 greater than (equal to) 0"
## [7] "0.5 greater than (equal to) 0"  "0.75 greater than (equal to) 0"
## [9] "1 greater than (equal to) 0"
```

Exercise: Debugging an Error

Identify the issue(s) with this function

```
MergeData <- function(data1, data2, key1, key2){
  # function to merge two data sets
  # Args: data1 - first dataset
  #       data2 - second dataset
  #       key1 - key name in first dataset
  #       key2 - key name in second dataset
 # Returns: merged dataframe if key matches,
  #          otherwise print an error
  if (key1 = key2){
    data.out <- join(data1,data2, by = key1)
    return(dataout)
  } else {
    stop('keys are not the same')
  }
}
```

Solution: Debugging an Error

Step 1 - fix '='

```
MergeData <- function(data1, data2, key1, key2){
  # function to merge two data sets
  # Args: data1 - first dataset
  #       data2 - second dataset
  #       key1 - key name in first dataset
  #       key2 - key name in second dataset
  # Returns: merged dataframe if key matches,
  #          otherwise print an error
  if (key1 == key2){
    data.out <- join(data1,data2, by = key1)
    return(dataout)
  } else {
    stop('keys are not the same')
  }
}
```

Solution: Debugging an Error

Step 2 - load dplyr() & use full_join

```
MergeData <- function(data1, data2, key1, key2){
  # function to merge two data sets
  # Args: data1 - first dataset
  #       data2 - second dataset
  #       key1 - key name in first dataset
  #       key2 - key name in second dataset
  # Returns: merged dataframe
  library(dplyr)
  if (key1 == key2){
    data.out <- full_join(data1,data2, by = key1)
    return(dataout)
  } else {
    stop('keys are not the same')
  }
}
MergeData(df1,df2,"school","school")
```

Solution: Debugging an Error

Step 3 - correct dataout to data.out

```
MergeData <- function(data1, data2, key1, key2){
  # function to merge two data sets
  # Args: data1 - first dataset
  #       data2 - second dataset
  #       key1 - key name in first dataset
  #       key2 - key name in second dataset
  # Returns: merged dataframe
  library(dplyr)
  if (key1 == key2){
    data.out <- full_join(data1,data2, by = key1)
    return(data.out)
  } else {
    stop('keys are not the same')
  }
}
```

Solution: Debugging an Error

Step 3 - correct dataout to data.out

```
MergeData(df1,df2,"school","school")
```

```
##   school state enrollment
## 1  Mines    CO       5794
## 2    MSU    MT      15688
## 3     VT    VA      30598
```

```
MergeData(df.cost,df.acres, 'ski.resort','ski.hill')
```

```
## Error in MergeData(df.cost, df.acres, "ski.resort", "ski.hill'
```

Advanced Debugging

Overview

We can often fix bugs using the ideas sketched out previously and this becomes *easier* with more experience coding in R. Trial and error can be very effective and strategic use of print function help to identify where bugs are occuring.

However, R does also have advanced tools to help with debugging code.

- `traceback()`
- "Rerun with debug"
- `browser()`

traceback()

Consider the following code:

```
f <- function(a) g(a)
g <- function(b) h(b)
h <- function(c) i(c)
i <- function(d)  "a" + d
f(10)
```

```
## Error in "a" + d: non-numeric argument to binary operat
```

TRACEBACK()

Consider the `traceback()` function. Which identifies which functions have been executed (along with the row number of the function).

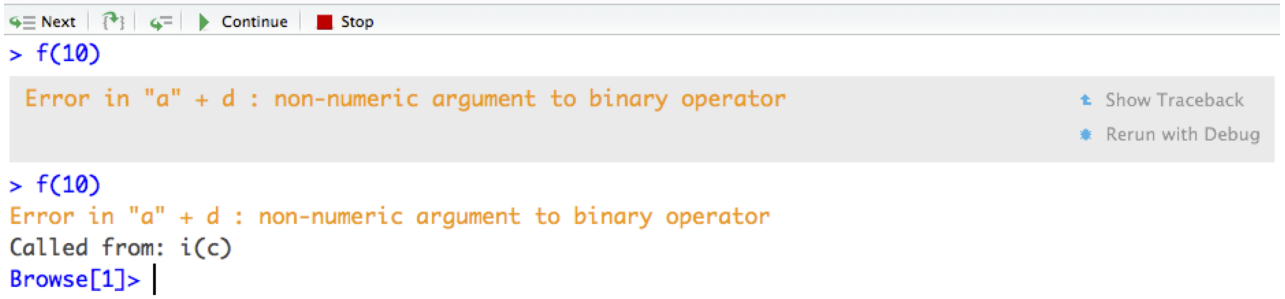
```
> traceback()

4: i(c) at #1
3: h(b) at #1
2: g(a) at #1
1: f(10)
```

Note: due to the way that R Markdown is compiled, `traceback()` needs to be run directly in R, not R Markdown.

Browsing on an error

Another option (in R Studio) is to browse on the error. This gives you an interactive way to move through the function calls to identify the problem of the location. This can also be called explicitly using `debug()`.



BROWSER()

The browser function can also be used to interactively step through a function.

```
SS <- function(mu, x) {
  browser()
  d <- x - mu
  d2 <- d^2
  ss <- sum(d2)
  ss
}
```

browser() step 1

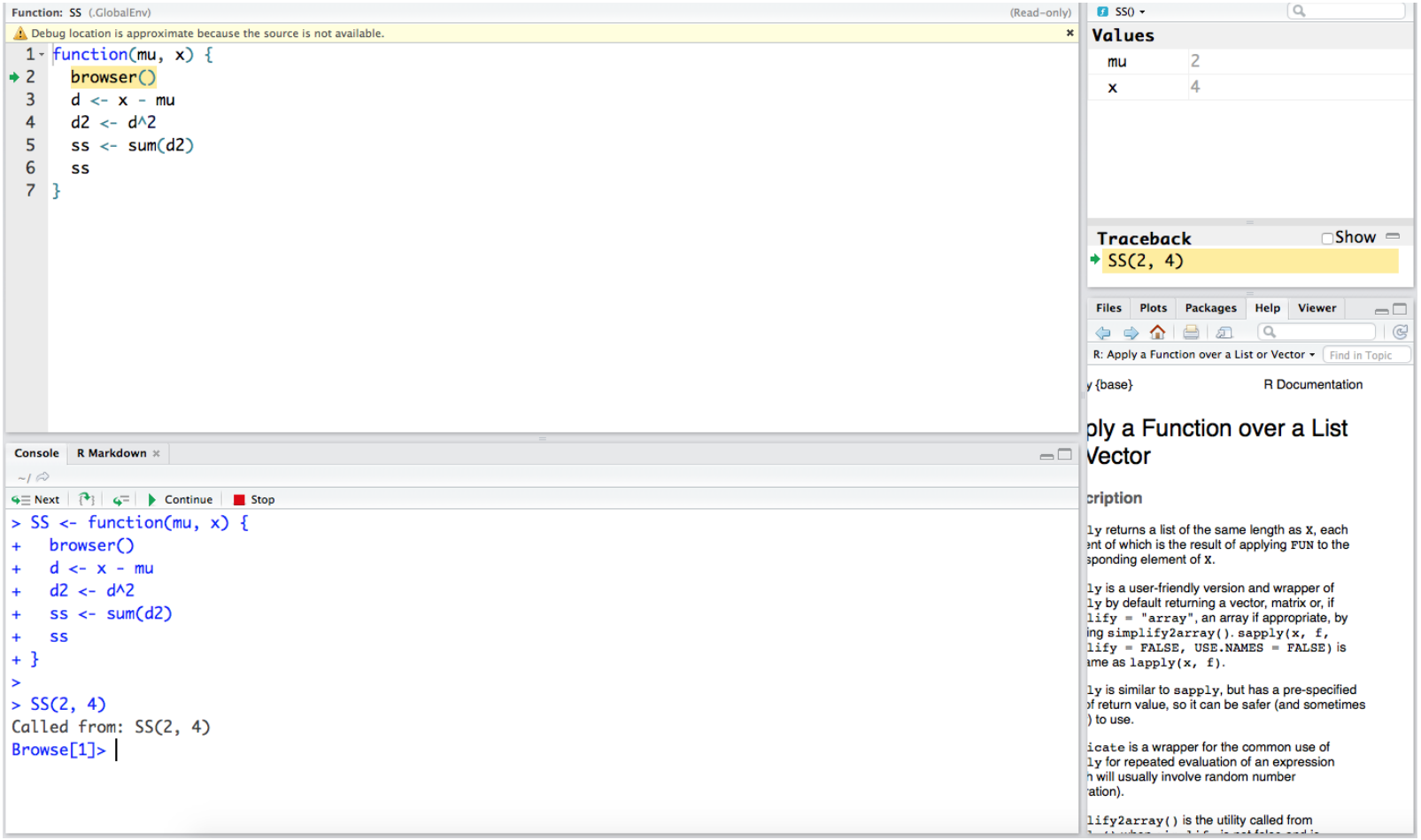


Figure 2

browser() step 2

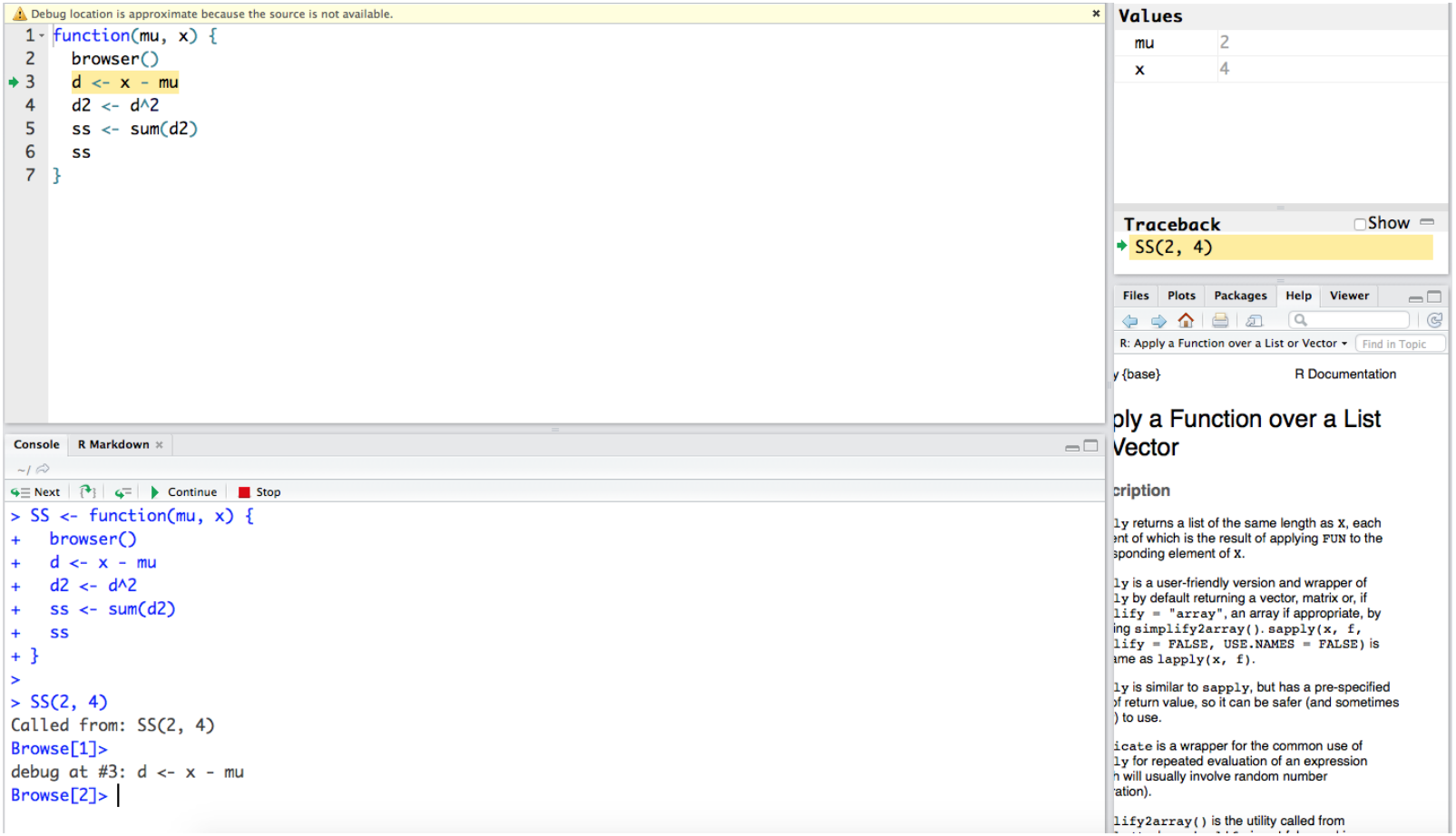


Figure 3

browser() step 3

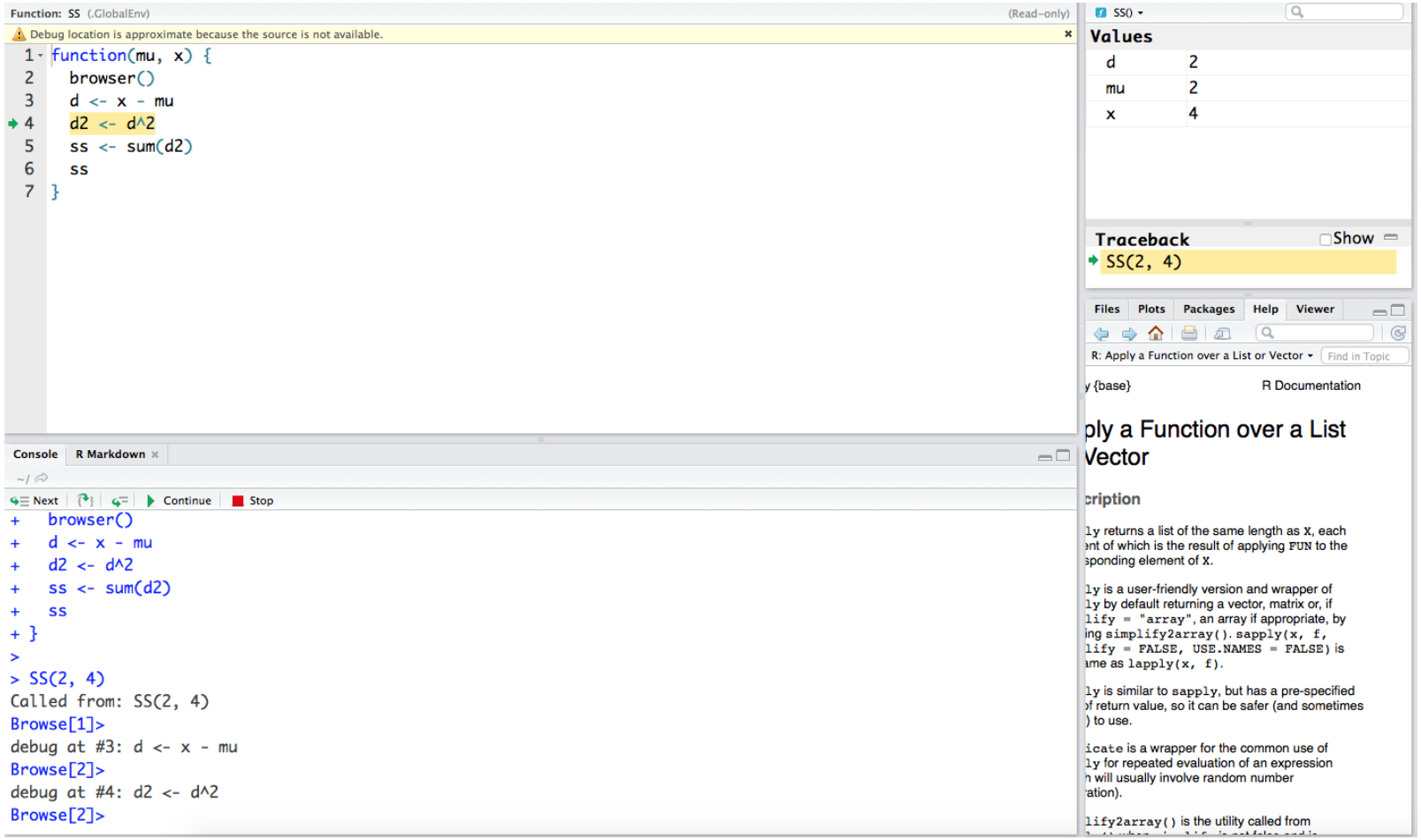


Figure 4

browser() step 4

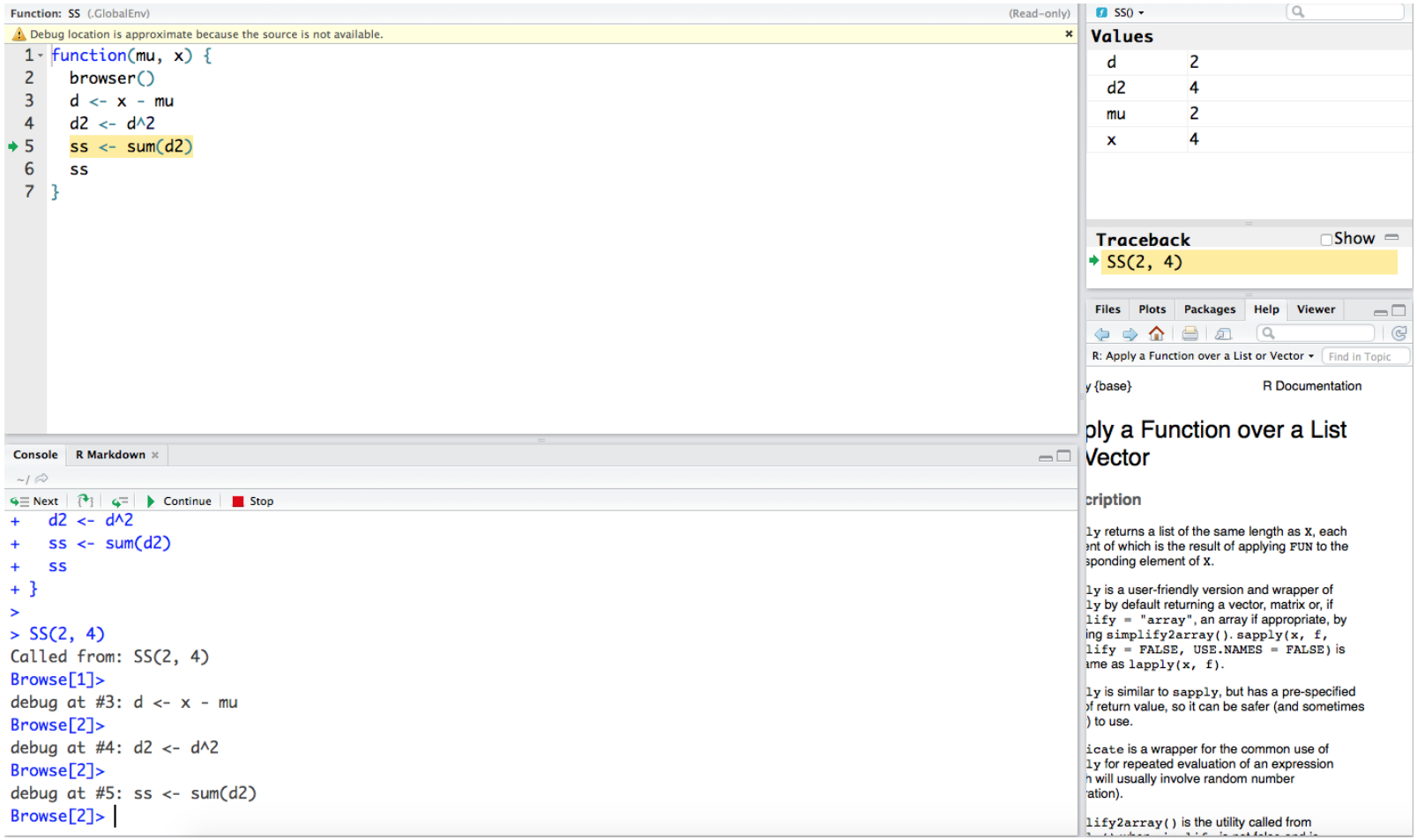


Figure 5

browser() step 5


```
SS <- function(mu, x) {
  browser()
  d <- x - mu
  d2 <- d^2
  ss <- sum(d2)
  ss
}

SS(2, 4)
```

```
+ }
>
> SS(2, 4)
Called from: SS(2, 4)
Browse[1]>
debug at #3: d <- x - mu
Browse[2]>
debug at #4: d2 <- d^2
Browse[2]>
debug at #5: ss <- sum(d2)
Browse[2]>
debug at #6: ss
Browse[2]>
[1] 4
>
```

Functions

f	function (a)
g	function (b)
h	function (c)
i	function (d)
MakeErr…	function (in.val)
SS	function (mu, x)

R: Apply a Function over a List or Vector

ply a Function over a List
Vector

Figure 6