Debugging with recover

## Learning Objectives

* Use recover to debug when code crashes

## Turn off RStudio's debugging features

Just like last lesson, make sure to turn of RStudio's debugging features. In the menu, go to "Debug". From the dropdown menu, go to "On Error" and choose the setting "Message Only".

Recall the function we wrote earlier to calculate a summary statistic which was the average of multiple specified metric columns.

calc\_sum\_stat <- function(df, cols) {  
 df\_sub <- df[, cols]  
 sum\_stat <- apply(df\_sub, 1, mean)  
 return(sum\_stat)  
}

Then we could create new metric columns, for example, the average of the multiple Facebook metrics or the average of the download metrics.

counts\_raw$facebookAverageCount <- calc\_sum\_stat(counts\_raw, grep("facebook",  
 colnames(counts\_raw),  
 value = TRUE))  
counts\_raw$downloadsAverageCount <- calc\_sum\_stat(counts\_raw, grep("Downloads",  
 colnames(counts\_raw),  
 value = TRUE))

Now what do we expect would happen if we passed only one column name to the function? While this would not be very informative since taking the average of one metric would return the same metric. But there is no reason to think it should not work. Let's try it.

calc\_sum\_stat(counts\_raw, "mendeleyReadersCount")

But in fact this does fail. The error message informs us that the error occurs during the call to apply. While we could use debug or browser to investigate this error, a convenient way to interrogate an error is to use the function recover. Specifically, we can set the option error so that the function recover is called any time there is an error.

options(error = recover)

Now let's run the function again to produce the error:

calc\_sum\_stat(counts\_raw, "mendeleyReadersCount")

Enter a frame number, or 0 to exit   
  
1: calc\_sum\_stat(counts\_raw, "mendeleyReadersCount")  
2: #3: apply(df\_sub, 1, mean)

Selection:

Essentially, recover allows us to explore the state of the code right before the error was thrown. It first asks us to select a frame number. The frames refer to the different environments that were created. First we called our function calc\_sum\_stat which has its own environment. But within that function, apply was called creating an additional environment within the environment of calc\_sum\_stat. These frames make up the call stack, which we can visualize as the list of sub-functions that have been called.

Since we know the error occured in apply, we'll choose frame #2.

Selection: 2

Called from: calc\_sum\_stat(counts\_raw, "mendeleyReadersCount")

Browse[2]>

This brings us to the familiar debugger environment. Since the state is frozen we can't actually use these commands, but in fact typing one of these commands will bring us back to the frame menu (simply hitting the Enter key works as well). But we can investigate the state of the variables.

Browse[1]> ls()

[1] "dl" "FUN" "MARGIN" "X"

To remind ourselves of the argument names of apply, we can open the help page.

Browse[1]> ?apply

This opens the Help tab, where we can read that X contains the data passed to apply:

X an array, including a matrix.

Investigating what X is, we see that is a vector instead of a one-column data frame.

Browse[1]> str(X)

int [1:24331] 4 17 0 0 32 10 0 6 2 24 ...

This is strange. Let's confirm this is what we passed to apply. Type c to exit the debugger and return to the frame menu.

Browse[1]> c

Enter a frame number, or 0 to exit   
  
1: calc\_sum\_stat(counts\_raw, "mendeleyReadersCount")  
2: #3: apply(df\_sub, 1, mean)

Selection:

Now we'll enter frame #1 to investigate df\_sub.

Selection: 1

Browse[3]> str(df\_sub)

int [1:24331] 4 17 0 0 32 10 0 6 2 24 ...

So it turns out that the first line that selects the one column returns a vector and not a data frame. This is actually the default behavior of R's extract function (yes, the brackets are actually a function, try ?"[" to learn more).

Let's exit recover so that we can return to fix our function.

Browse[1]> c

Enter a frame number, or 0 to exit   
  
1: calc\_sum\_stat(counts\_raw, "mendeleyReadersCount")  
2: #3: apply(df\_sub, 1, mean)

Selection: 0

To fix this problem, we need to pass a third argmument when subsetting a data frame. If we set drop = FALSE, then it will remain a one column data frame instead of being converted to a vector.

calc\_sum\_stat <- function(df, cols) {  
 df\_sub <- df[, cols, drop = FALSE]  
 sum\_stat <- apply(df\_sub, 1, mean)  
 return(sum\_stat)  
}

And now it works.

head(calc\_sum\_stat(counts\_raw, "mendeleyReadersCount"))

[1] 4 17 0 0 32 10

Unfortunately you will not always be able to easily solve the problem yourself. But since you know how to isolate a problem using the debugging tools, you are prepared to ask for help on online forums. For best results, follow advice for creating a minimal, reproducible example:

* [Stack Overflow - How to make a great R reproducible example?](http://stackoverflow.com/questions/5963269/how-to-make-a-great-r-reproducible-example)
* [R-bloggers - Three tips for posting good questions to R-help and Stack Overflow](http://www.r-bloggers.com/three-tips-for-posting-good-questions-to-r-help-and-stack-overflow/)
* [Stack Overflow - How to create a Minimal, Complete, and Verifiable example](http://stackoverflow.com/help/mcve)