DATA 133 - Introduction to Data Science I

Instructor: Renzhi Cao
Computer Science Department
Pacific Lutheran University

Announcements

- Go over Quiz 4
- Continue to work on project 1

Reference book

- R Programming for Data Science. By Roger Peng. **ISBN-10:** 1365056821, April 20, 2016.

Pictures from: https://www.google.com/search?

Learning in today

- R basics - Debugging

Something is wrong

R has a number of ways to indicate to you that something's not right.

- message: A generic notification/diagnostic message produced by the message() function; execution of the function continues
- warning: An indication that something is wrong but not necessarily fatal; execution of the function continues. Warnings are generated by the warning() function
- error: An indication that a fatal problem has occurred and execution of the function stops. Errors are produced by the stop() function.
- condition: A generic concept for indicating that something unexpected has occurred; programmers can create their own custom conditions if they want.

Something is wrong

Example:

- log(-1)

Something is wrong

Example:

```
 printmessage <- function (x) {
+ if (x > 0 )
+  print("x is greater than zero" )
+ else
+  print("x is less than or equal to zero" )
+  invisible(x)
+  }

>  printmessage(1)
[1 ] "x is greater than zero”
```

Seems no errors, warnings, or messages

How about:

```
printmessage(NA )
```

Something is wrong

Fixed Example:

```
> printmessage2 <- function (x) {
+ if (is.na(x))
+  print("x is a missing value!" )
+ else if (x > 0 )
+  print("x is greater than zero" )
+ else
+  print("x is less than or equal to zero" )
+  invisible(x)
+  }

 > printmessage2(NA )
 [1 ] "x is a missing value!"
```

Something is wrong

Think about the following:

```
>  x <-  log(c(-1 , 2 ))
Warning in  log(c(-1 , 2 )):  NaNs produced

> printmessage2(x)
Warning in if (is.na(x)) print("x is a missing value!") else if (x > 0)
print("x is greater than zero") else print("x is less than or equal to
zero"): the condition has length > 1 and only the first element will be
used
[1] "x is a missing value!"
```

The printmessage2() is not vectorized

Something is wrong

One solution:

```
>  printmessage3 <- function (x) {
+ if (length(x) > 1L )
+  stop("'x' has length > 1" )
+ if (is.na(x))
+  print("x is a missing value!" )
+ else if (x > 0 )
+  print("x is greater than zero" )
+ else
+  print("x is less than or equal to zero" )
+  invisible(x)
+  }

>  printmessage3(1:2 )
Error in  printmessage3(1:2 ): 'x'  has length > 1
```

Something is wrong

Another solution:

```
> printmessage4 <- Vectorize(printmessage2)
> out <- printmessage4(c(-1, 2))
[1] "x is less than or equal to zero"
[1] "x is greater than zero"
```

Figuring out what is wrong

Some basic questions you need to ask are

- What was your input? How did you call the function?
- What were you expecting? Output, messages, other results?
- What did you get?
- How does what you get differ from what you were expecting?
- Were your expectations correct in the first place?
- Can you reproduce the problem (exactly)?

Debugging tools in R

- traceback(): prints out the function call stack after an error occurs; does nothing if there's no error
- debug(): flags a function for "debug" mode which allows you to step through execution of a function one line at a time
- browser(): suspends the execution of a function wherever it is called and puts the function in debug mode
- trace(): allows you to insert debugging code into a function a specific places
- recover(): allows you to modify the error behavior so that you can browse the function call stack

Debugging tools in R

traceback():

```
> mean(x)
Error in mean(x) : object 'x' not found
> traceback()
1: mean(x)
```

The traceback() function must be called immediately after an error occurs

Debugging tools in R

debug() initiates an interactive debugger (also known as the “browser” in R) for a function.

```
> debug(lm) ## Flag the 'lm()' function for interactive debugging
> lm(y ~ x)
debugging in: lm(y ~ x)
debug: {
    ret.x <- x
    ret.y <- y
    cl <- match.call()
    ...
   if (! qr)
      z$ qr <- NULL
   z
}
```

Now, every time you call the lm() function it will launch the interactive debugger
Use undebug() to turn off: undebug(lm)

Debugging tools in R

There are a few special commands you can call in the browser:
• n executes the current expression and moves to the next expression
• c continues execution of the function and does not stop until either an
error or the function
exits
• Q quits the browser

Here's an example of a browser session with the lm() function.

```
Browse[2]> n ## Evalute this expression and move to the next one
debug: ret.x <- x
Browse[2]> n
debug: ret.y <- y
Browse[2]> n
debug: cl <- match.call()
Browse[2]> n
debug: mf <- match.call(expand.dots = FALSE)
Browse[2]> n
debug: m <- match(c("formula", "data", "subset", "weights", "na.action",
"offset"), names(mf), 0L)
```

Practice of R

Debugging R code
Continue to work on Project 1
Next Tuesday is review day for mid-term exam
Read book