*Debugging in R*

**R.Phillips**

Last week I attended a session at the Data Science Initiative at UC Davis led by Dr. Duncan Temple Lang. Dr. Lang followed through on the obvious (but rarely explored) idea of actually using the debugging tools that we all carry around with us in our R installations.

The procedure is pretty simple, it’s a matter of going to the tool, rather than just ‘hacking around’.

As we all know, when your code breaks, you want to:

1) figure out where it breaks,

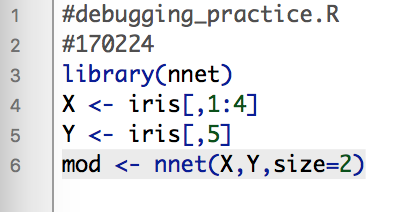
2) ‘zoom in’ on that point,

3) find out what the problem is,

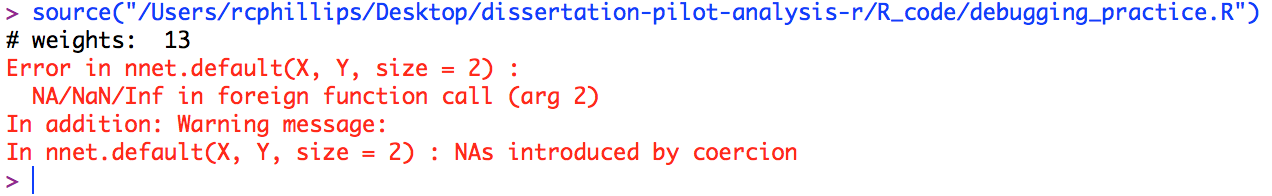
4)  and fix it.

Duncan’s method was the same, but more powerful, because it used the tools that were designed to achieve this purpose.

So, lets go through the same example we did in the session.

Here’s our code:  


When we run it, we get:



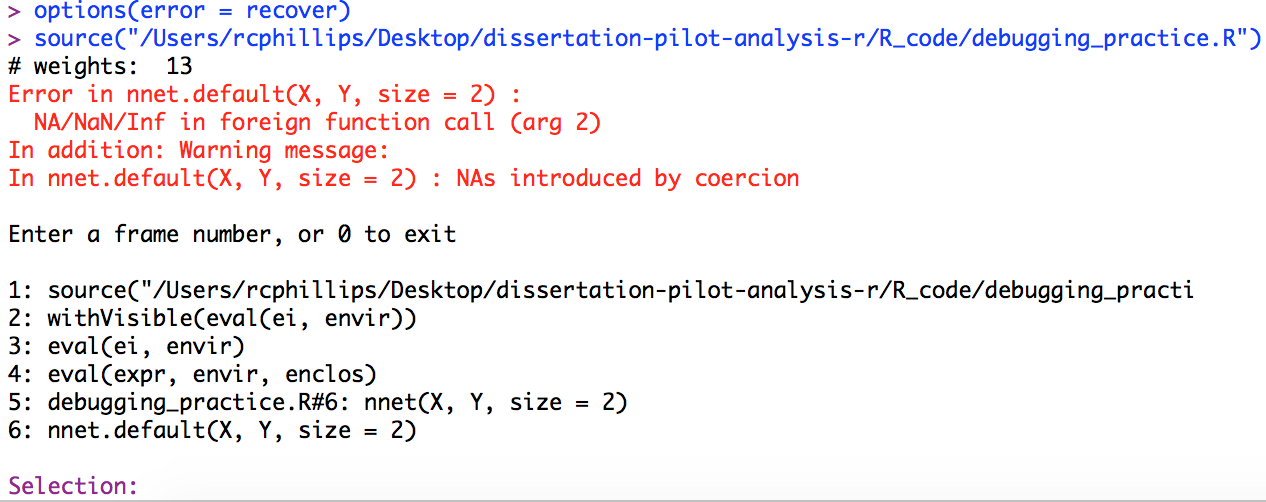
Here we have an error.

We will use some of the options contained under the options() function.

*options(error == dump.frames)*

This will change our console’s response to errors. Dump.frames will “dump” the frames into the console for your observation. (I feel like dump means it should go into the trash can, but there you go.)

And then we run the code again.



We get the same error, but now we can enter into different steps of the code

We are then prompted to input which frame we wish to enter. This will take us to the point in the code, as if we stuck a bunch of print statements where we think there’s an error. The nice thing is that we can jump around, move forward and back, enter and exit without ever modifying our code. We can even run commands based on that frame point in order to check different things out.

One of the most useful of these is *type()*

As well as *dim()*

And *ls()*

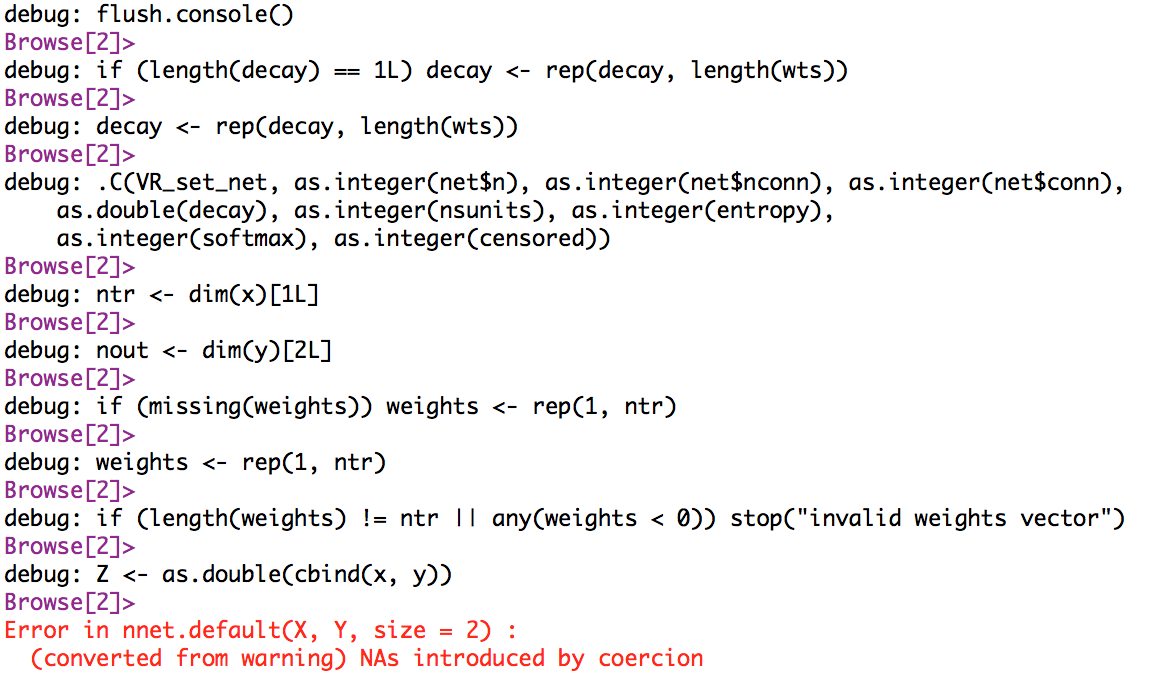
Reading the error message, it looks like our problem is in step 6: nnet.default(X,Y, size=2)

We can then use *debug(nnet.default*) and *options(warn=2)*

This will move our code into the debugger once we get to nnet.default. We can then bump forward through the code and see where the error arises

*Options(warn=2)* will actually halt the code *as* *soon* as we get to the error. One trouble that can arise is that you get the error at the *end* of the function, without being able to identify exactly where the problem is.

So, doing that, we step through the function using enter or n:



And we see that this occurs in at the step *as.double(cbind(x,y))*

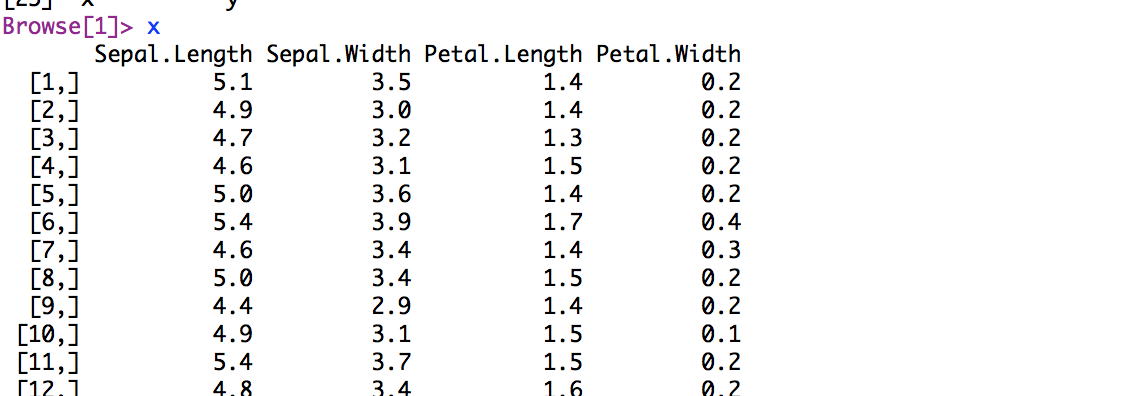
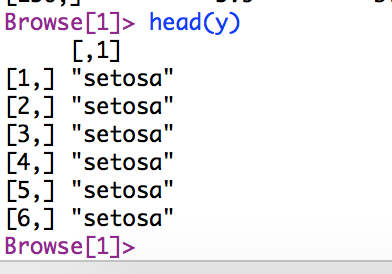
So we remove the debug, with undebug()

And we rerun the code.

We again get our six steps, and go to 6, and enter *ls()*

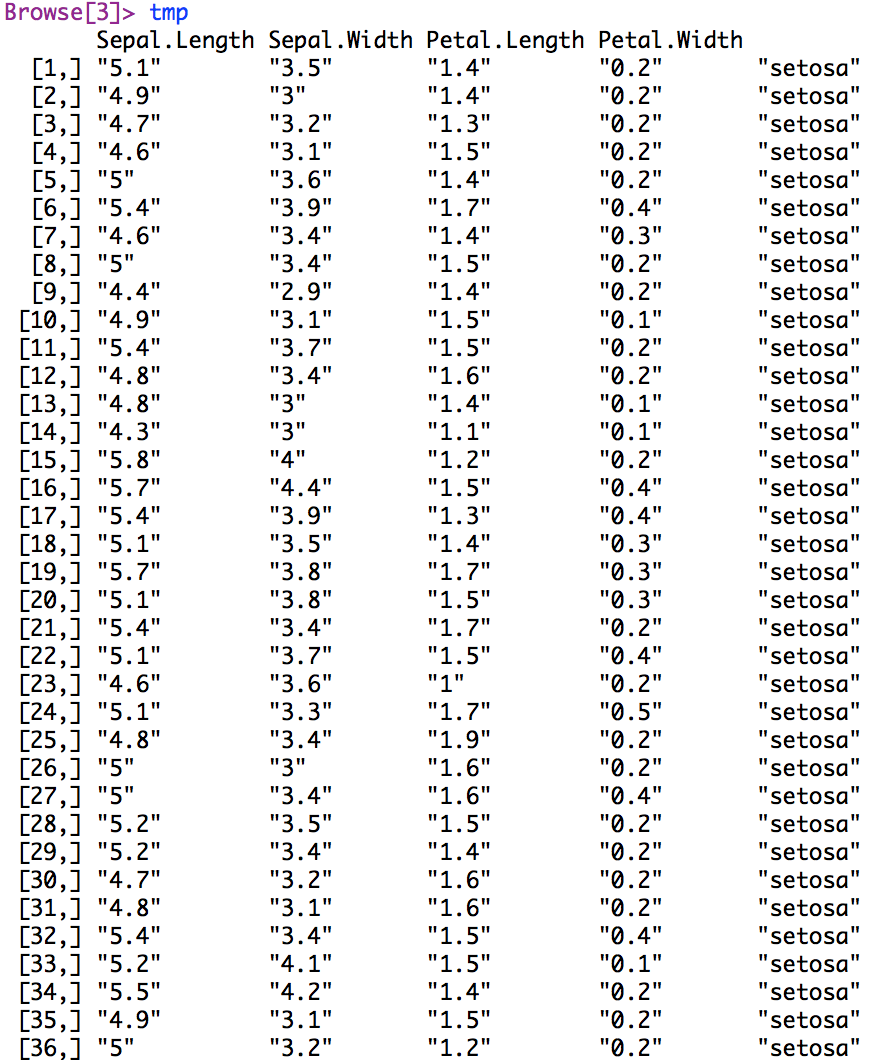
Then, we can check the type of the variable entering the function.

We see x and y will be fed into cbind.

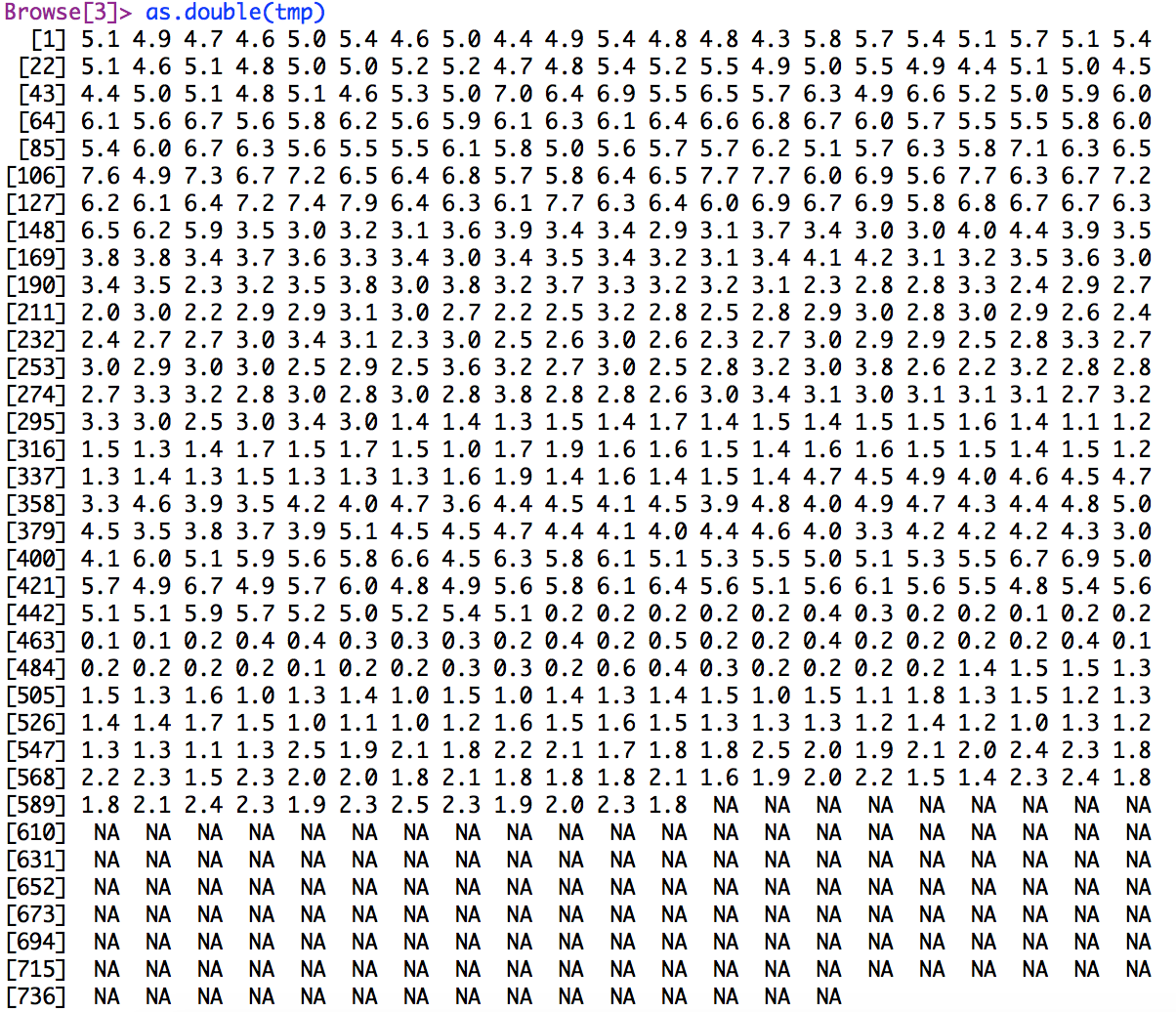


So, from the error message we can see that our error is arising from NAs. But we don’t see any NAs here.

Tmp<-cbind(x,y) gives



But if we go further,



Then, having zoomed in, and found the problem, we can fix it.

So it looks as though somehow, our as.double() function is converting our strings to NAs.

This is where we should focus in the next step, when we fix it. But that’s a post for another time