Tutorial 5 | Debugging

1. Intro to errors and debugging

Indications that something's not right:

- **message**: A generic notification/diagnostic message; execution of the function continues
- **warning**: An indication that something is wrong, but it is not fatal; execution of the function continues
- **error**: An indication that something fataly wrong happened that stop function form being executed
- **condition**: A generic concept for indicating that something unexpected can occur; can be created by programmers themselves

Below are the questions that come in handy when you are debugging a function:

- What was your input?(important: not what you **thought** you feed into that function, but what you **actually** put in here?) How did you call the function?
- What were you expecting? Output, messages, or any other results?
- What did you actually get?
- How does what you got differ from what you were expecting?
- Were your expectations correct in the first place?
- Can you reproduce the problem? (meaning to reproduce the error/warning/etc. that you got)

2. Debugging Tools in R

There are few primary tools for debugging functions in R:

- **traceback**: prints out the function call stack after an error occurs(where in the sequence of function calls the error occured); does nothing if there is 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 at specific places
- **recover**: allows you to modify the error behavior so that you can browse the function call stack

These tools are specifically designed to allow you to pick through a function. There is also a more blunt technique of inserting statements into the function.

3. Examples

**Now before we start with examples, just a little side note, you will have to execute below codes by yourself (without #'s ofc), because errors in code pieces in R MarkDown file prevent it from compiling into pdf.

3.1 Traceback

```
#new_f<-function(x){mean(x)}
#new_f(aaa)
```

```
#traceback()
```

This is a very simple example (and kinda unnecessary in this case tbh) but it is here just to give you an idea of how the traceback function works. Here is a more complicated example of how traceback can be particularly useful(don't bother understanding the function itself and al the calls it makes just look througth the general pattern of the traceback):

```
#lm(bbb~aaa)
```

```
#traceback()
```

Look whole 7 levels deep!

3.2 Debug

Now let's use the previous case to introduce the debug function:

```
#debug(lm)
#lm(bbb~aaa)
```

Well, the first thing that should happen is the **debug** function will print out the whole code of lm function and also it will open a 'browser' where it will now execute a function line by line. so just copy other the code to your console an see what happens.
To execute a line type in 'n' in the console and it wille xecute the function line by line so that you can get to the line where the error occurs.

3.3 Recover

```
#options(error=recover)
#read.csv("filethatdoesntexist")
```

This function allows you to browse through the function call error stack. You can press 1,2 or 3 to jump through the files which are the levels of the error to see where and what exactly went wrong.