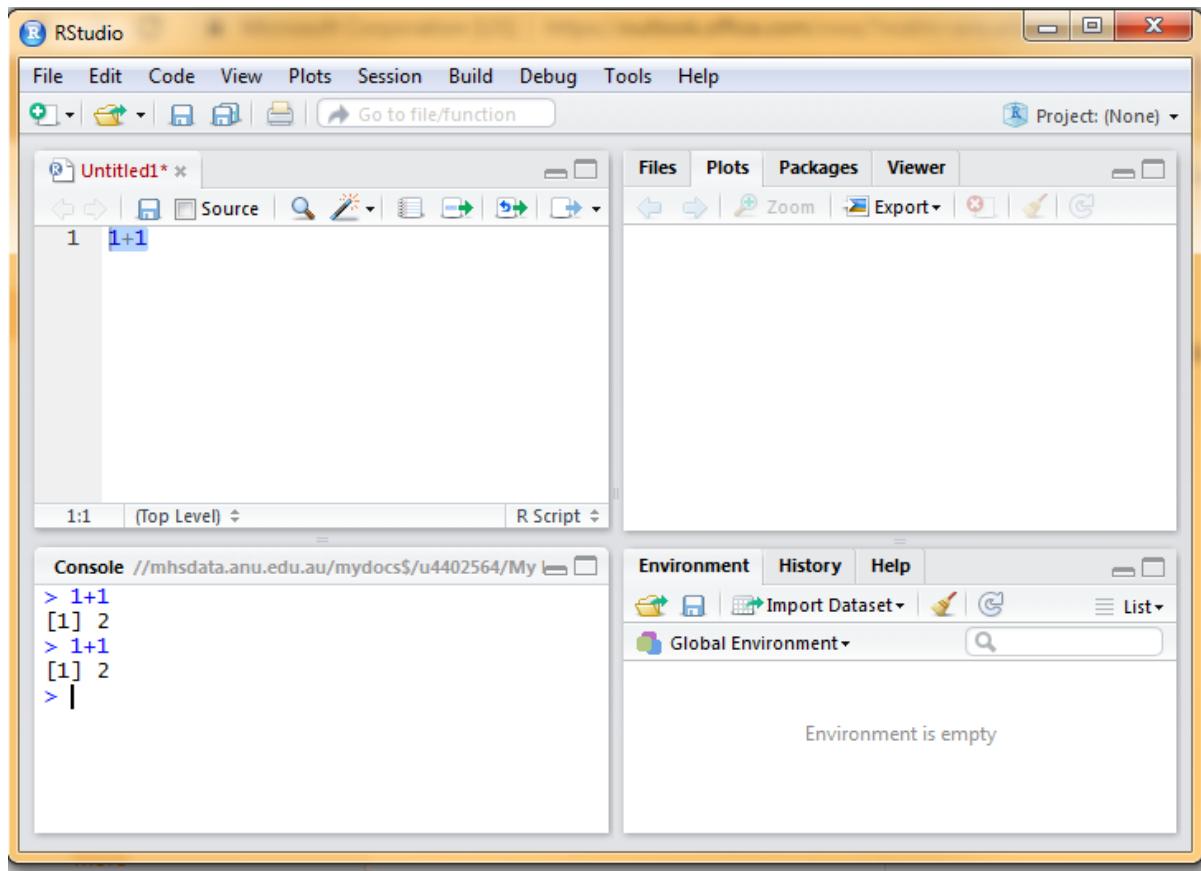
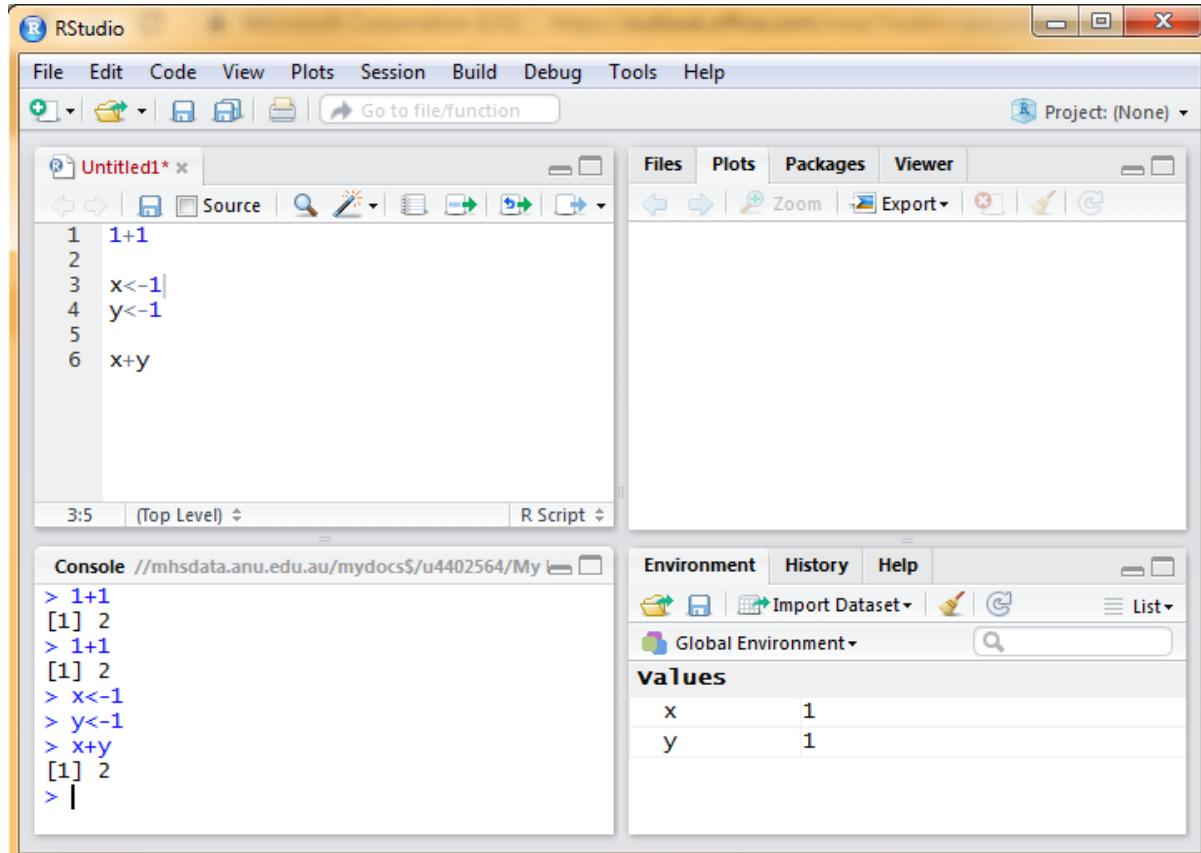


Basic R orientation

You can run the same things either directly in the console (below), or the script window (above). Anything you run in the script window (highlight it and press control + enter) will also show up in the console too.

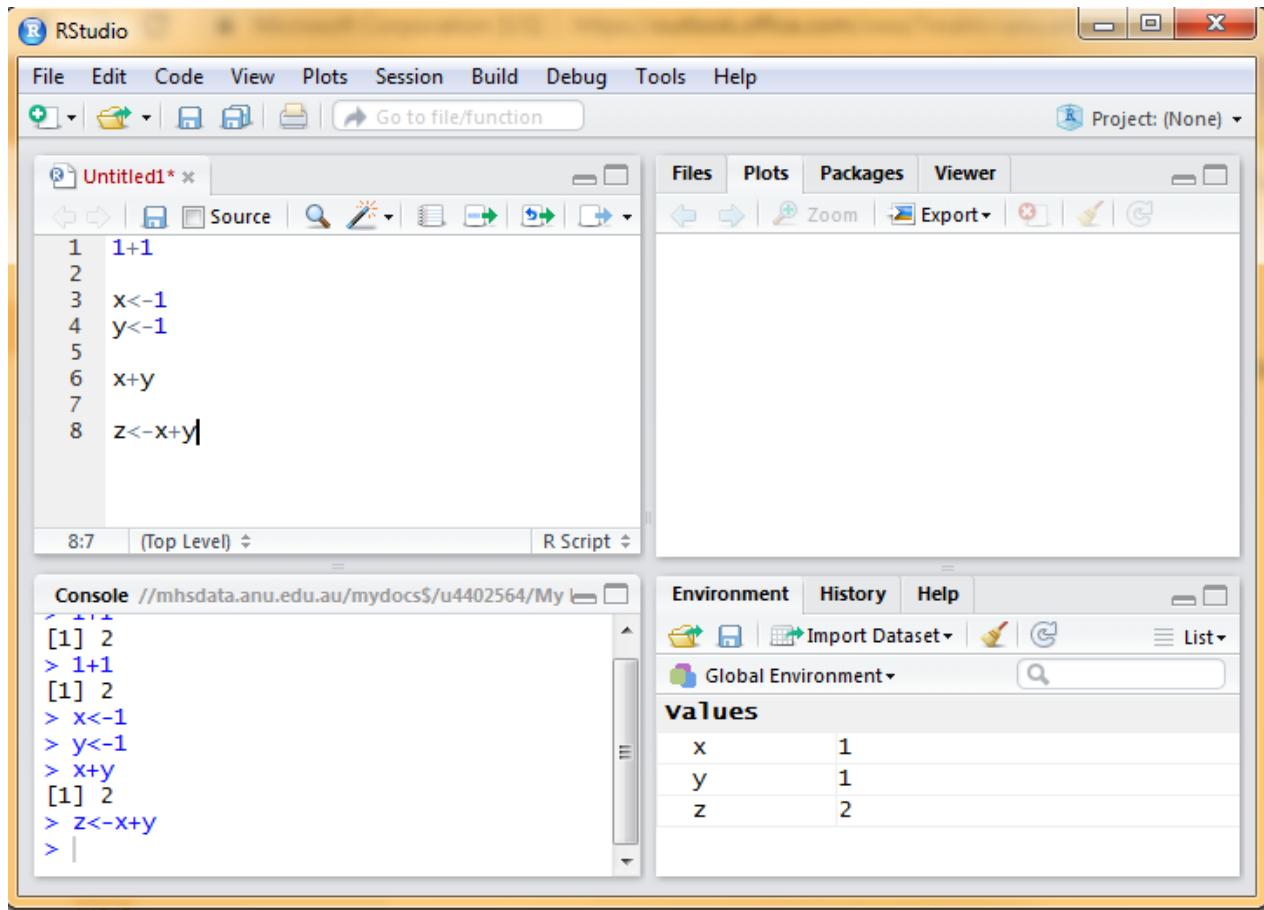


Variable assignment can be done via the `<-` operator (note they turn up in the global environment)

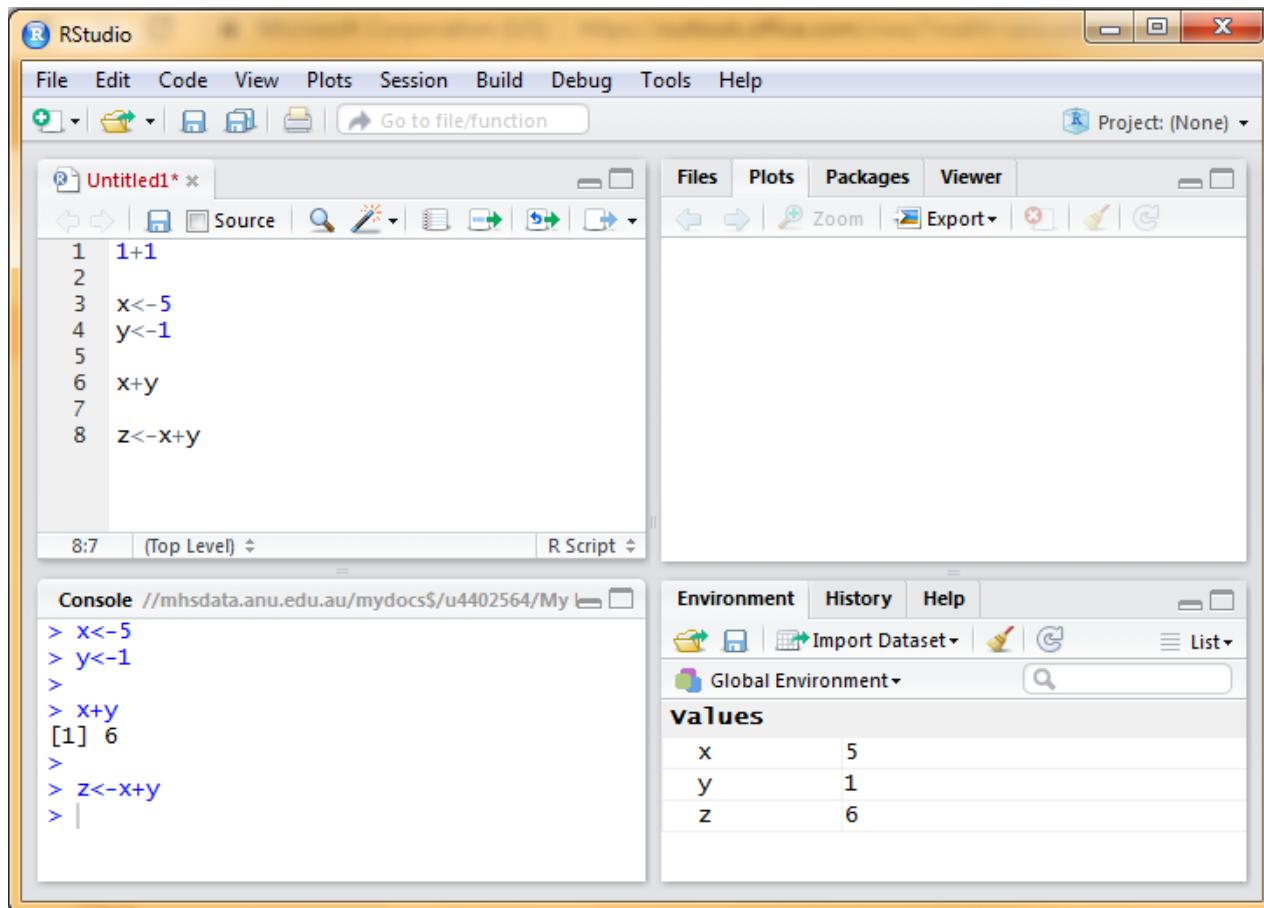


Basic introduction to R Dr. E.I. Walsh, 2019

You can add variables together like they're numbers.



This means you can easily change one value earlier in the code to update your outcomes.



Functions

All functions are basically of the form:

```
function_name <- function(input1, input2, ... ){
    statements
    return(output)
}
```

Write a simple function for adding two numbers.

The screenshot shows the RStudio interface. In the top-left pane (Script Editor), the code for a function is written:

```
1 1+1
2
3 x<-5
4 y<-1
5
6 x+y
7
8 z<-x+y
9
10 simple_function<-function(number ,anothernumber){number + anothernumber}
11
12
```

In the bottom-left pane (Console), the function is defined and called:

```
> simple_function<-function(number ,anothernumber){number + anothernumber}
> |
```

In the bottom-right pane (Environment), the variables and function are listed:

- values**

 - x 5
 - y 1
 - z 6

- Functions**

 - simple_func... function (number , anothernumber)

... and call it.

The screenshot shows the RStudio interface. In the top-left pane (Script Editor), the code is identical to the previous screenshot, but the line `simple_function(1,1)` is highlighted.

In the bottom-left pane (Console), the function is called and its result is displayed:

```
> simple_function<-function(number ,anothernumber){number + anothernumber}
> simple_function(1,1)
[1] 2
> |
```

In the bottom-right pane (Environment), the variable `z` is listed with a value of 2.

Basic introduction to R Dr. E.I. Walsh, 2019

These functions can also be used with those variables you specified before.

The screenshot shows the RStudio interface. In the top-left pane, a script named 'Untitled1.R' contains the following code:

```
4 y<-1
5
6 x+y
7
8 z<-x+y
9
10 simple_function<-function(number ,anothernumber){number + anothernumber}
11
12 simple_function(1,1)
13
14 simple_function(x,y)
15
16
```

In the bottom-left pane, the Console window shows the execution of the script:

```
> simple_function<-function(number ,anothernumber){number + anothernumber}
> simple_function(1,1)
[1] 2
> simple_function(x,y)
[1] 6
>
```

In the bottom-right pane, the Environment window displays the current variables:

Values	x	5
y	1	
z	6	

Under the Functions section, it lists the defined function:

```
simple_func... function (number , anothernumber)
```

You can save the output as a new variable, which can then be called like any other variable.

The screenshot shows the RStudio interface. In the top-left pane, a script named 'Untitled1.R' contains the following code:

```
6 x+y
7
8 z<-x+y
9
10 simple_function<-function(number ,anothernumber){number + anothernumber}
11
12 simple_function(1,1)
13
14 simple_function(x,y)
15
16 output_variable<-simple_function(x,y)
17
18 output_variable|
```

In the bottom-left pane, the Console window shows the execution of the script:

```
> simple_function<-function(number ,anothernumber){number + anothernumber}
> simple_function(1,1)
[1] 2
> simple_function(x,y)
[1] 6
> output_variable<-simple_function(x,y)
> output_variable
[1] 6
>
```

In the bottom-right pane, the Environment window displays the current variables:

Values	output_vari...	6
x	5	
y	1	
z	6	

Under the Functions section, it lists the defined function:

```
simple_func... function (number , anothernumber)
```

Basic introduction to R Dr. E.I. Walsh, 2019

You can assign the operation inside the function to a variable, but unless you tell the function to return it you won't get anything out.

The screenshot shows the RStudio interface. In the top-left pane, there is an 'Untitled1' script file containing the following R code:

```
11 simple_function(1,1)
12
13 simple_function(x,y)
14
15 output_variable<-simple_function(x,y)
16
17 output_variable
18
19
20 another_simple_function<-function(number ,anothernumber){added<-number + anothernumber}
21 another_simple_function(x,y)
22
23
```

In the bottom-left pane, the R Console shows the execution of this code:

```
> simple_function<-function(number ,anothernumber){number + anothernumber}
> simple_function(1,1)
[1] 2
> simple_function(x,y)
[1] 6
> output_variable<-simple_function(x,y)
> output_variable
[1] 6
> another_simple_function<-function(number ,anothernumber){added<-number + anothernumber}
> another_simple_function(x,y)
> |
```

The right side of the interface includes the 'Files', 'Plots', 'Packages', and 'Viewer' tabs, and the 'Environment' tab which displays the values of variables: output_variable (6), x (5), y (1), and z (6). It also shows the definition of the another_simple_function.

Need to add a return() argument.

The screenshot shows the RStudio interface. In the top-left pane, there is an 'Untitled1' script file containing the following R code:

```
15
16 output_variable<-simple_function(x,y)
17
18 output_variable
19
20
21 another_simple_function<-function(number ,anothernumber){added<-number + anothernumber}
22 another_simple_function(x,y)
23
24
25 another_simple_function<-function(number ,anothernumber){added<-number + anothernumber
26   return(added)}
27 another_simple_function(x,y)
28
29
```

In the bottom-left pane, the R Console shows the execution of this code:

```
> another_simple_function<-function(number ,anothernumber){added<-number + anothernumber
+   return(added)}
> another_simple_function(x,y)
[1] 6
> |
```

The right side of the interface includes the 'Files', 'Plots', 'Packages', and 'Viewer' tabs, and the 'Environment' tab which displays the values of variables: output_variable (6), x (5), y (1), and z (6). It also shows the definition of the another_simple_function.

Basic introduction to R Dr. E.I. Walsh, 2019

The key thing you can do in functions is adding multiple operations

RStudio interface showing a script editor with the following R code:

```
21 another_simple_function<-function(number ,anothernumber){added<-number + anothernumber}
22 another_simple_function(x,y)
23
24
25 another_simple_function<-function(number ,anothernumber){added<-number + anothernumber
26 return(added)}
27 another_simple_function(x,y)
28
29 another_simple_function<-function(number ,anothernumber){added<-number + anothernumber
30 multiplied<-added * number
31 return(multiplied)}
32 another_simple_function(x,y)
33
33:1 (Top Level) R Script
```

Console output:

```
> another_simple_function<-function(number ,anothernumber){added<-number + anothernumber
+ multiplied<-added * number
+ return(multiplied)}
> another_simple_function(x,y)
[1] 30
>
```

Global Environment viewer showing variables:

Values	x	y	z
output_v...	6		
x	5		
y	1		
z	6		

You can see how functions are the pathway to complex, reproducible operations :D

RStudio interface showing a script editor with the following R code:

```
25 another_simple_function<-function(number ,anothernumber){added<-number + anothernumber
26 return(added)}
27 another_simple_function(x,y)
28
29 another_simple_function<-function(number ,anothernumber){added<-number + anothernumber
30 multiplied<-added * number
31 return(multiplied)}
32 another_simple_function(x,y)
33
34 # Without variable assignment...
35 (5+1) * 5
36 # Have to make multiple changes if one value changes
37 (5+1) * 5
38 (6+1) * 6
39 (7+1) * 7
40
41 # Variable assignment simplifies it a bit, but not reproducible.
42 x<-5 # Change x and re-run, easy
43 (x+1) * x
44
45 # Function is much easier for re-use
46 another_simple_function(5,1)
47 another_simple_function(6,1)
48 another_simple_function(7,1)
49
50
51
52
53
54
55
48:31 (Top Level) R Script
```

Console output:

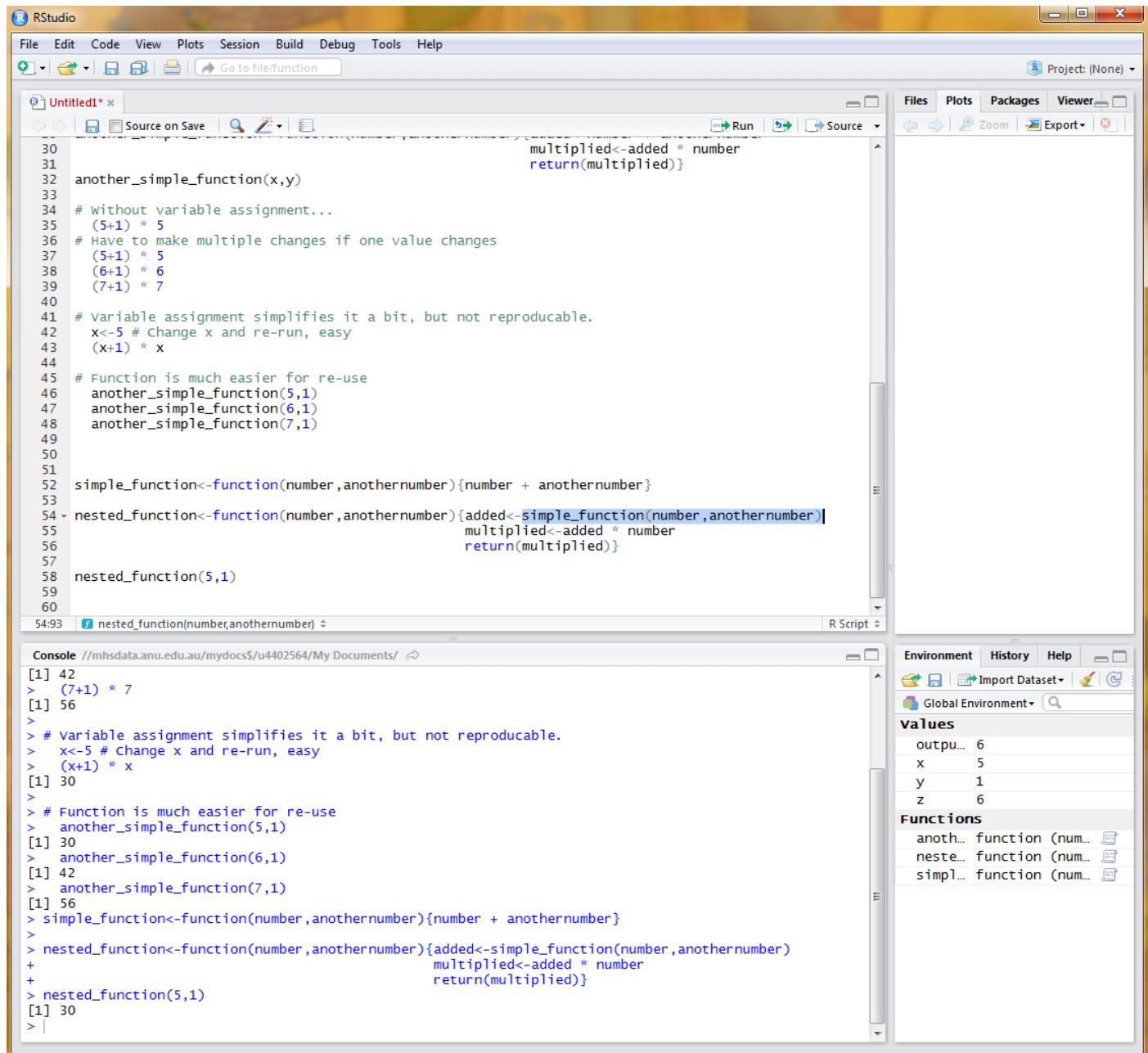
```
> # without variable assignment...
> (5+1) * 5
[1] 30
> # Have to make multiple changes if one value changes
> (5+1) * 5
[1] 30
> (6+1) * 6
[1] 42
> (7+1) * 7
[1] 56
>
> # Variable assignment simplifies it a bit, but not reproducible.
> x<-5 # Change x and re-run, easy
> (x+1) * x
[1] 30
>
> # Function is much easier for re-use
> another_simple_function(5,1)
[1] 30
> another_simple_function(6,1)
[1] 42
> another_simple_function(7,1)
[1] 56
>
```

Global Environment viewer showing variables:

Values	x	y	z
output_v...	6		
x	5		
y	1		
z	6		

Basic introduction to R Dr. E.I. Walsh, 2019

Also note, you can nest functions inside other functions for convenience.



Loops

A loop is a sequence of instructions that is continually repeated until a certain condition is reached, of the generic form `for(iteration in number of times) {do something}`. Typically, we use a lowercase `i` to refer to the specific iteration. So, if we want a loop to run 5 times, we specify `i` in `1:5`, and then allow the second argument to our `simple_function` to be `i` - this means each time the loop progresses, that `i` will become 1, 2, 3, 4, then 5.

The screenshot shows the RStudio interface. The left pane displays the script file 'example_script.R' with the following code:

```

66
67
68
69 # Challenge: add 1 to every number between 1 and 5.
70 1+1
71 1+2
72 1+3
73 1+4
74 1+5
75
76 simple_function(1,1)
77 simple_function(1,2)
78 simple_function(1,3)
79 simple_function(1,4)
80 simple_function(1,5)
81
82 for(i in 1:5){print(simple_function(1,i))}
83
84
85
86
87
88
89
90
91

```

The right pane shows the environment and global environment panes. The global environment pane contains:

values
i 5L
output_... 6
x 5
y 1
z 6

The console pane at the bottom shows the output of the script:

```

[1] 4
> simple_function(1,4)
[1] 5
> simple_function(1,5)
[1] 6
>
> for(i in 1:5){print(simple_function(1,i))}
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
>

```

Like a function, it will not return anything if you don't tell it to specify an output. So, you need to either tell it to print out using the `print()` function (this prints the output to console)

The screenshot shows the RStudio interface with the same script file 'example_script.R'. The code has been modified to include the `print` statement:

```

66
67
68
69 # Challenge: add 1 to every number between 1 and 5.
70 1+1
71 1+2
72 1+3
73 1+4
74 1+5
75
76 simple_function(1,1)
77 simple_function(1,2)
78 simple_function(1,3)
79 simple_function(1,4)
80 simple_function(1,5)
81
82 for(i in 1:5){simple_function(1,i)}
83
84 for(i in 1:5){print(simple_function(1,i))}
85
86
87
88
89
90
91

```

The right pane shows the environment and global environment panes. The global environment pane contains:

values
i 5L
numbers_... logical (empty)
output_... 6
x 5
y 1
z 6

The console pane shows the output:

```

> for(i in 1:5){simple_function(1,i)}
>
> for(i in 1:5){print(simple_function(1,i))}
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
>

```

Basic introduction to R

Dr. E.I. Walsh, 2019

Or fill up an empty variable. You build an empty variable by assigning it an empty vector value, used as `vector()`.

The screenshot shows the RStudio interface. In the top-left pane, there is a script named "example_script.R". The code in the script is as follows:

```
79 simple_function(1,4)
80 simple_function(1,5)
81
82 for(i in 1:5){simple_function(1,i)}
83
84 for(i in 1:5){print(simple_function(1,i))}
85
86
87 # Make an empty vector: there's nothing there
88 numbers_added<-vector()
89
90 numbers_added
91
92 # Fill it with loop output
93
94 for(i in 1:5){numbers_added[i]<-simple_function(1,i)}
95
96 numbers_added
97
98
99
100
101
102
103
104
```

In the bottom-left pane, the "Console" window shows the execution of the script:

```
> # Make an empty vector: there's nothing there
> numbers_added<-vector()
>
> numbers_added
logical(0)
>
> # Fill it with loop output
>
> for(i in 1:5){numbers_added[i]<-simple_function(1,i)}
>
> numbers_added
[1] 2 3 4 5 6
>
```

The right side of the interface contains the "Environment" and "Global Environment" panes, which show the variables and functions defined in the script.

Now we have basic loop, you can see how easy it is to change the parameters. Changing the number in the `for` statement changes how many times the loop runs. You can change the number you pass to the loop. And, of course, you can change the function inside the loop.

The screenshot shows the RStudio interface again. The script "example_script.R" has been modified:

```
98
99 # Can easily change how many times you do this
100 numbers_added<-vector()
101 for(i in 1:50){numbers_added[i]<-simple_function(1,i)}
102 numbers_added
103
104
105 # Can easily change the numbers involved
106 numbers_added<-vector()
107 for(i in 1:50){numbers_added[i]<-simple_function(3,i)}
108 numbers_added
109
110 # Or swap out which function we use
111 numbers_added<-vector()
112 for(i in 1:50){numbers_added[i]<-nested_function(3,i)}
113 numbers_added
114
115
```

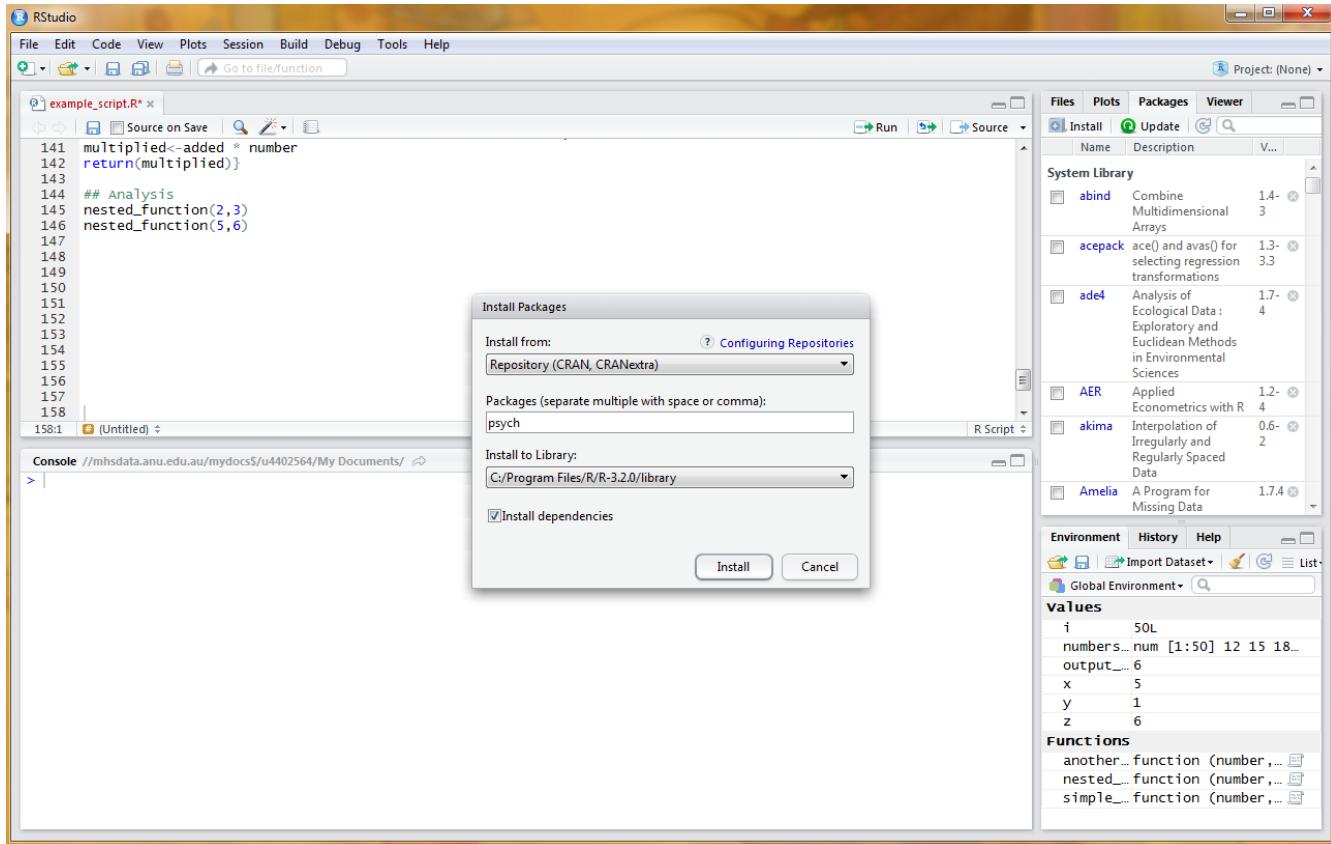
The "Console" window shows the new execution results:

```
> # Can easily change how many times you do this
> numbers_added<-vector()
> for(i in 1:50){numbers_added[i]<-simple_function(1,i)}
> numbers_added
[1] 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
[38] 39 40 41 42 43 44 45 46 47 48 49 50 51
>
> # Can easily change the numbers involved
> numbers_added<-vector()
> for(i in 1:50){numbers_added[i]<-simple_function(3,i)}
> numbers_added
[1] 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
[38] 41 42 43 44 45 46 47 48 49 50 51 52 53
>
> # Or swap out which function we use
> numbers_added<-vector()
> for(i in 1:50){numbers_added[i]<-nested_function(3,i)}
> numbers_added
[1] 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60 63 66 69 72 75 78 81 84 87 90 93
[29] 96 99 102 105 108 111 114 117 120 123 126 129 132 135 138 141 144 147 150 153 156 159
>
```

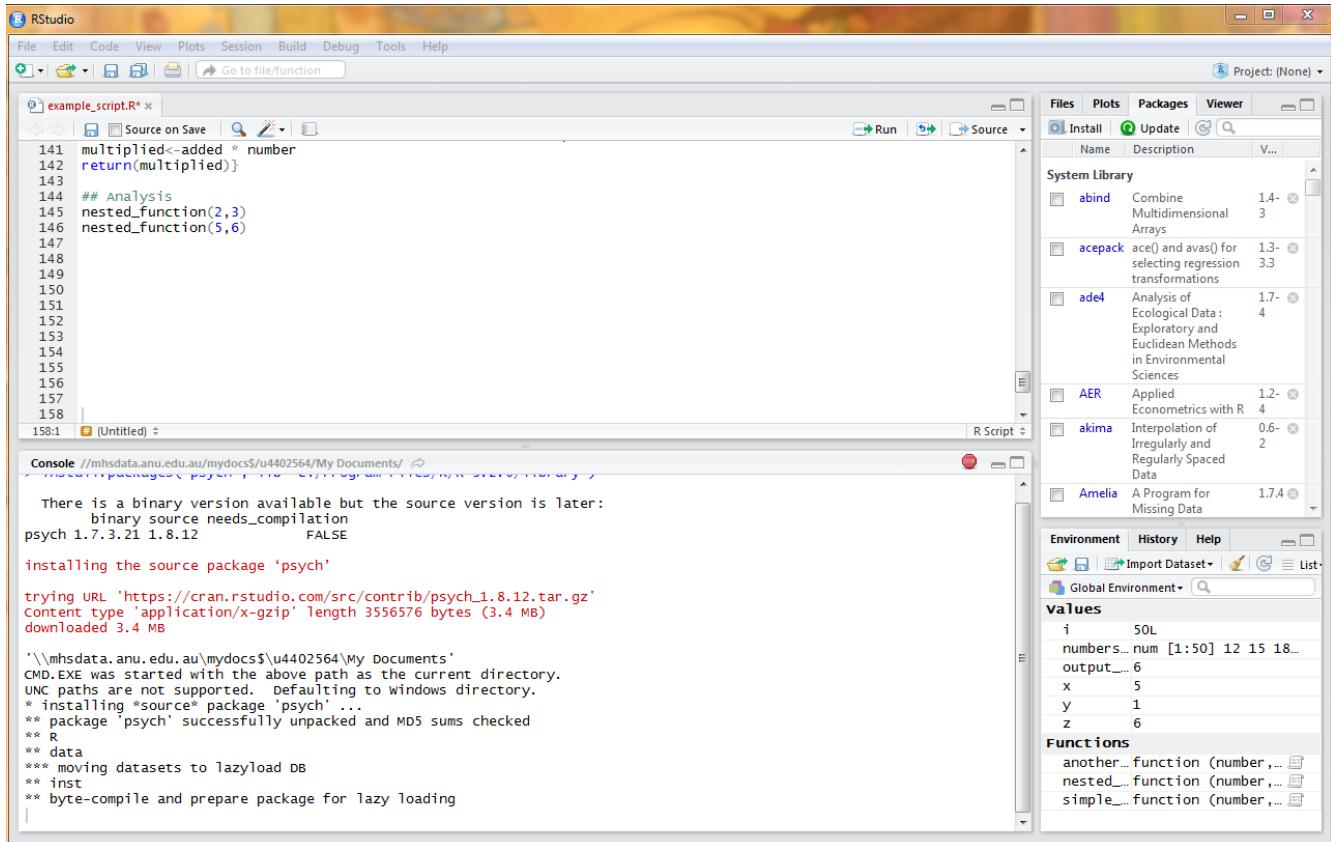
The "Environment" and "Global Environment" panes show the updated variables and functions.

Packages

You can think of packages as collated collections of functions other people have written. To install, in RStudio go to ‘packages’, ‘install’, then type the name of the package you want. For example we’re going to install ‘psych’, it has basic descriptive statistics.



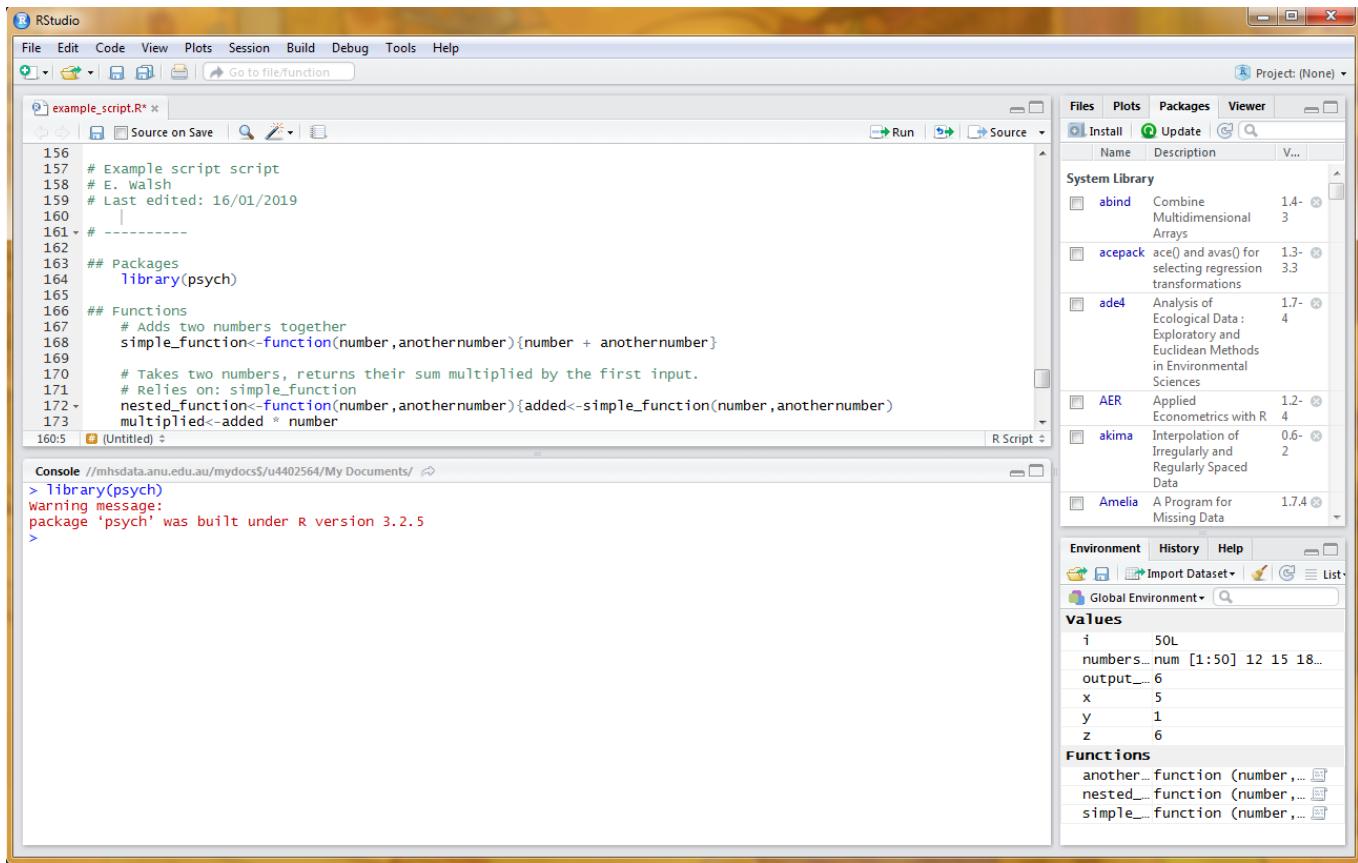
Console will show it loading.



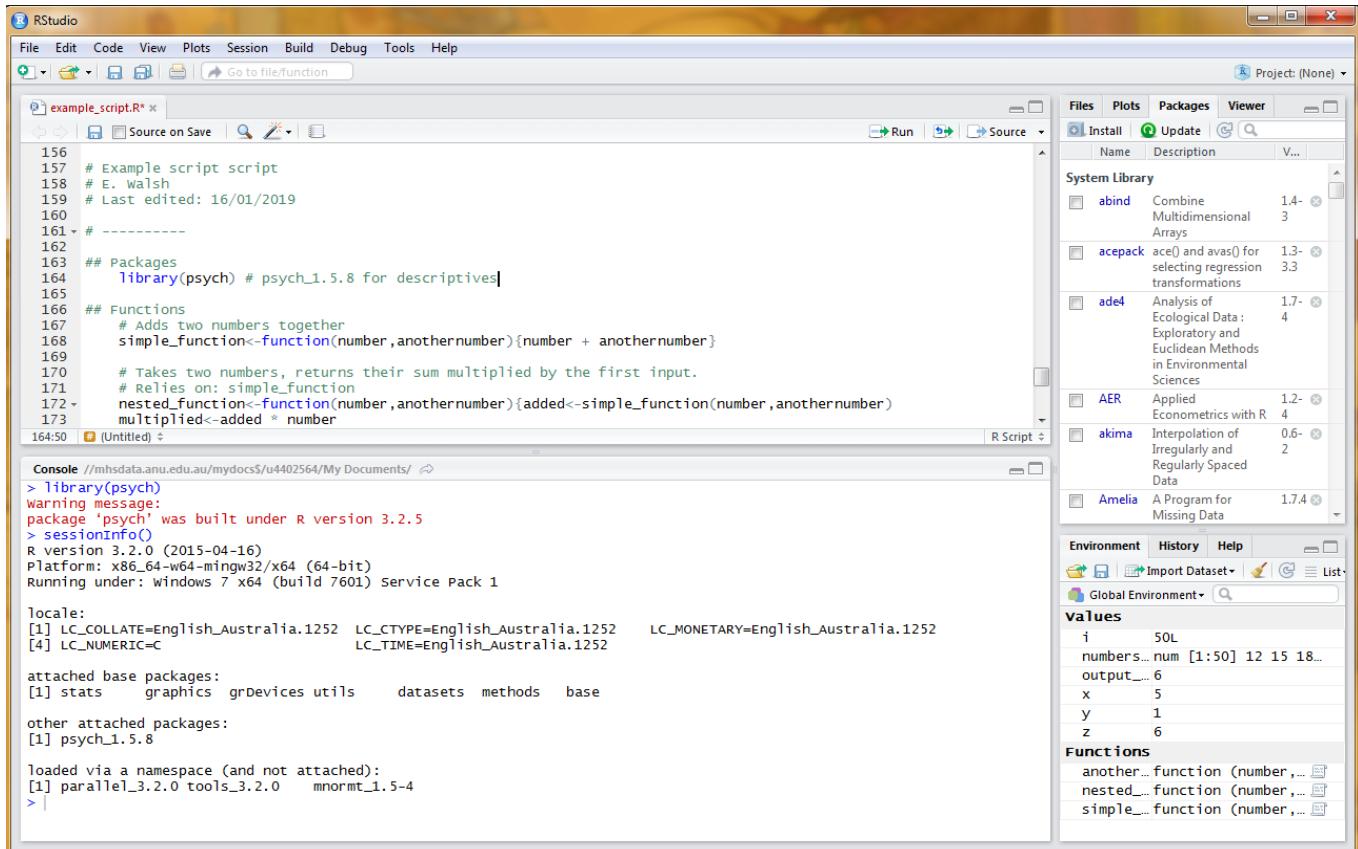
Basic introduction to R

Dr. E.I. Walsh, 2019

Once you have it installed (only need to do it once) you load it (ideally at the top of the script) using the call `library(psych)`.



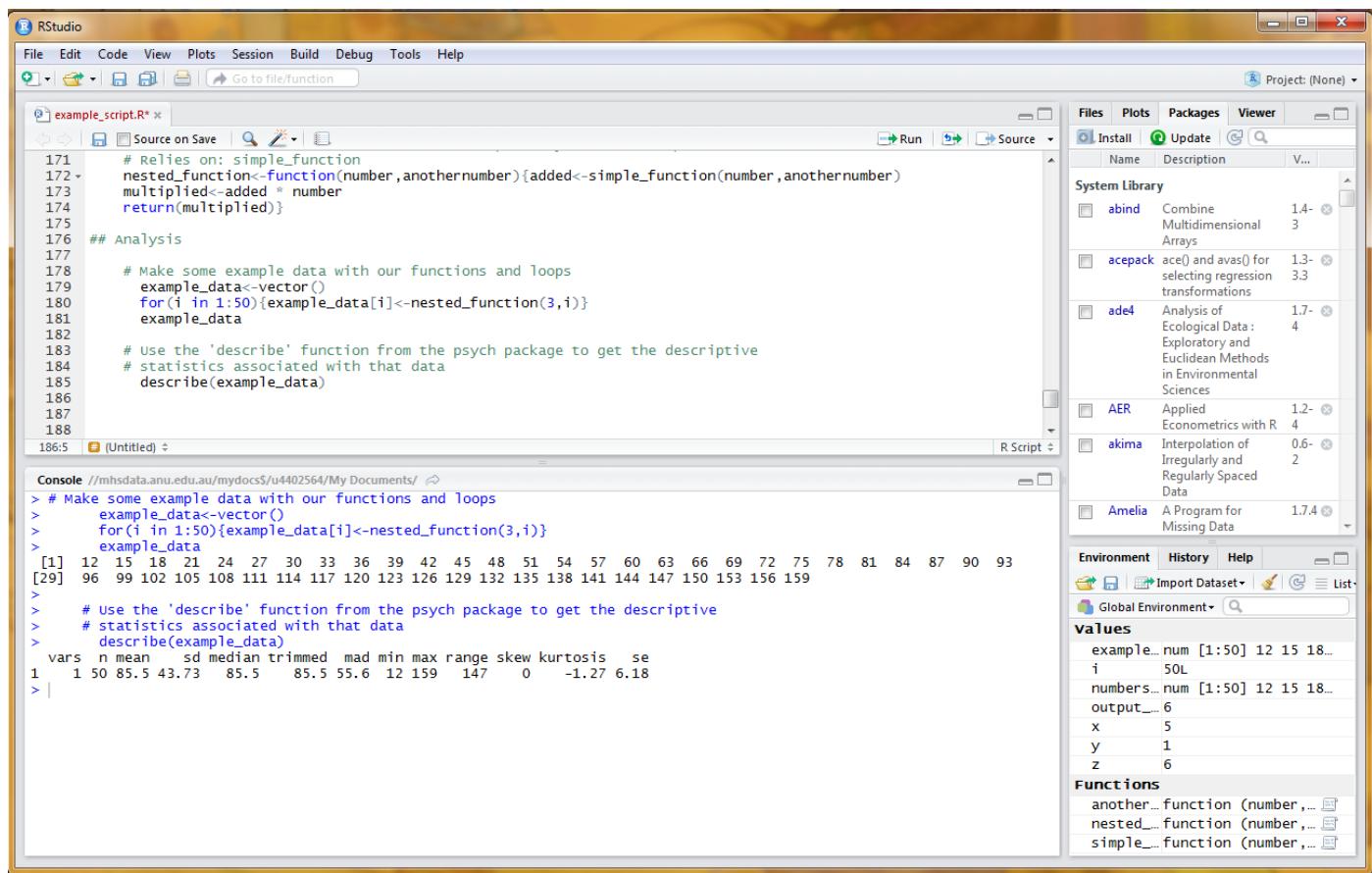
Package versions can change all the time, and you may well end up loading a lot of them, so it is best practise to include (a) what it was for, and (b) version information in the comments near the package. You can obtain this information by typing `sessionInfo()` in the console.



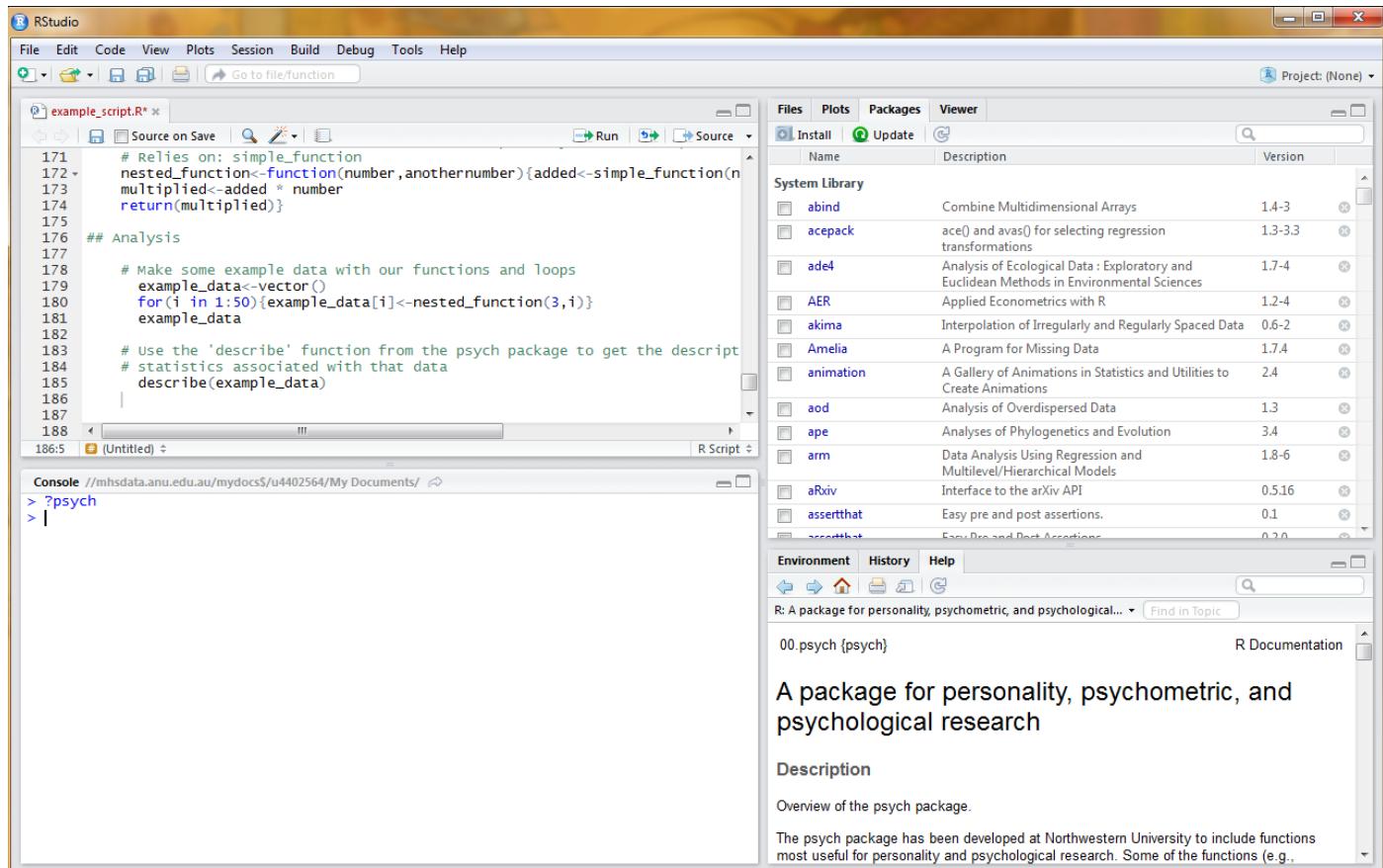
Basic introduction to R

Dr. E.I. Walsh, 2019

Now we can use all the functions that package has to offer. For example, let's use the `describe()` function to grab some descriptive statistics.



Information about packages can be found online, but also in R. To find help, use a question mark and the package's name, like the `?psych` example here. Help comes up in the 'help' window.



Basic introduction to R Dr. E.I. Walsh, 2019

You can do the same for functions within a package, such as `?describe`.

The screenshot shows the RStudio interface. In the top-left pane, the code for 'example_script.R' is displayed:

```
171 # Relies on: simple_function
172 nested_function<-function(number,anothernumber){added<-simple_function(n
173 multiplied<-added * number
174 return(multiplied)}
175
176 ## Analysis
177
178 # Make some example data with our functions and loops
179 example_data<-vector()
180 for(i in 1:50){example_data[i]<-nested_function(3,i)}
181 example_data
182
183 # use the 'describe' function from the psych package to get the descriptive
184 # statistics associated with that data
185 describe(example_data)
186
187
188
189
190
191
192
193
194
195
196
197
198
```

In the bottom-left pane, the console shows:

```
> ?psych
> ?describe
> |
```

In the top-right pane, the 'Packages' tab is selected, showing the 'System Library' with the 'psych' package listed. In the bottom-right pane, the 'describe' function documentation is shown:

R: Basic descriptive statistics useful for psychometrics

Description

There are many summary statistics available in R; this function provides the ones most useful for scale construction and item analysis in classic psychometrics. Range is most useful for the first pass in a data set, to check for coding errors.

Of course, this doesn't work with the function you made up, as you've not written any documentation for it.

The screenshot shows the RStudio interface. In the top-left pane, the code for 'example_script.R' is displayed:

```
182
183 # use the 'describe' function from the psych package to get the descriptive
184 # statistics associated with that data
185 describe(example_data)
186
187
188
189
190
191
192
193
194
195
196
197
198
```

In the bottom-left pane, the console shows:

```
> ?psych
> ?describe
> ?simple_function
No documentation for 'simple_function' in specified packages and libraries:
you could try '??simple_function'
>
```

In the top-right pane, the 'Packages' tab is selected, showing the 'System Library' with the 'psych' package listed. In the bottom-right pane, the 'describe' function documentation is shown:

R: Basic descriptive statistics useful for psychometrics

Description

There are many summary statistics available in R; this function provides the ones most useful for scale construction and item analysis in classic psychometrics. Range is most useful for the first pass in a data set, to check for coding errors.

Scripts

First - you can use the `#` to add comments. These don't produce output in the console window, and are extremely useful for keeping track of precisely what you're doing.

The screenshot shows the RStudio interface with a script file named "Untitled1.R" open. The code in the script is:

```

1 # Use the hash to add comments, these will not run.
2 # 1+1
3 # 1+1
4
5
6
7
8
9
10
11
12
13
14
15

```

In the console, the following output is shown:

```

> # use the hash to add comments, these will not run.
> 1+1
[1] 2
> # 1+1
>

```

The right panel shows the Global Environment with variables `x`, `y`, and `z` defined with values 5, 1, and 6 respectively.

Good practise: lead with who wrote the script it, what it is for, and when you last meaningfully edited it. Then setup (functions up the top), and then your analysis. Use comments to remind you what you were doing, and indentation for readability. ... Future you will be very grateful to past you.

The screenshot shows the RStudio interface with a script file named "example_script.R" open. The code in the script is:

```

1 # Example script script
2 # E. Walsh
3 # Last edited: 16/01/2019
4 # -----
5
6
7 # use the hash to add comments, these will not run.
8 # 1+1
9 # 1+1
10
11 ## Functions
12 # Adds two numbers together
13 simple_function<-function(number,anothernumber){number + anothernumber}
14
15 # Takes two numbers, returns their sum multiplied by the first input.
16 # Relies on: simple_function
17 nested_function<-function(number,anothernumber){added<-simple_function(number,anothernumber)
18 multiplied<-added * number
19 return(multiplied)}
20
21 ## Analysis
22 nested_function(2,3)
23 nested_function(5,6)
24

```

In the console, the following output is shown:

```

>

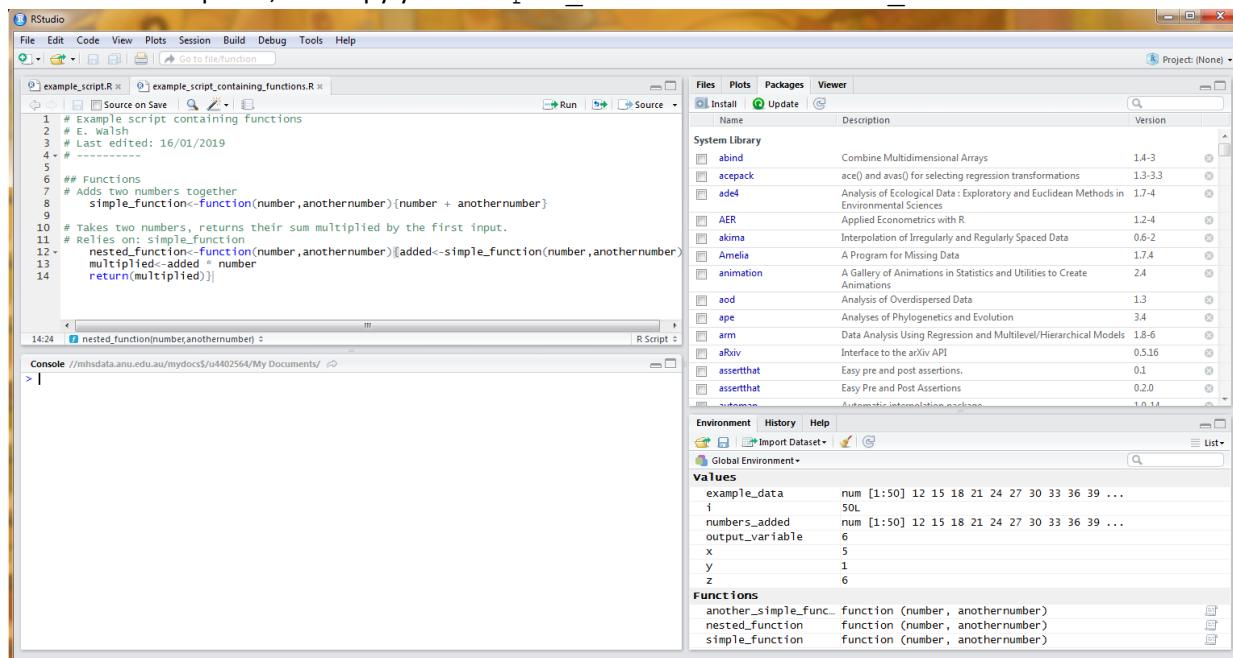
```

The right panel shows the Global Environment with variables `x`, `y`, and `z` defined with values 5, 1, and 6 respectively, and three user-defined functions: `simple_function`, `nested_function`, and `another_function`.

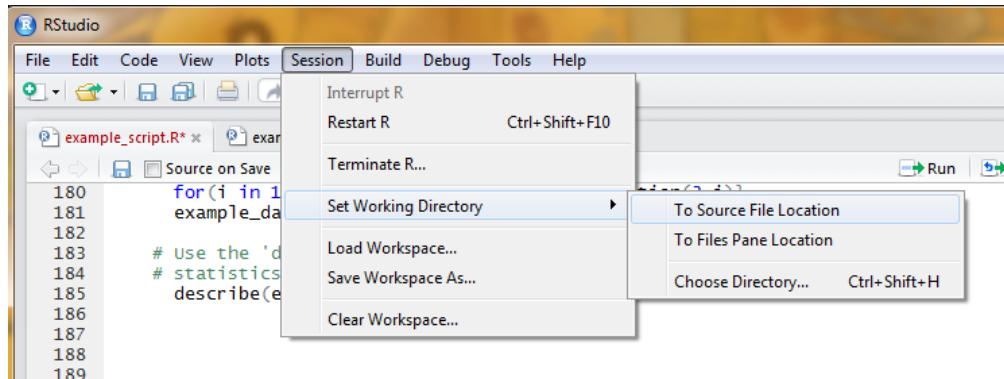
Basic introduction to R Dr. E.I. Walsh, 2019

A common usage scenario is saving functions in a script.

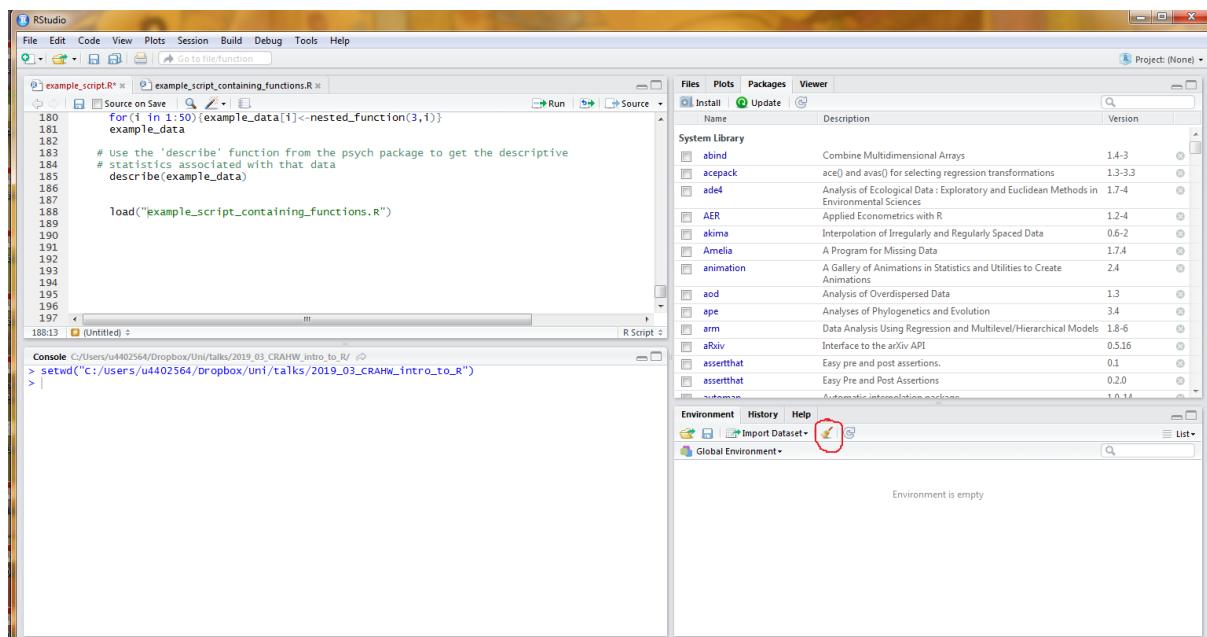
Make a new script file, and copy your `simple_function` and `nested_function` functions into it.



Go back to your first script, set the working directory.



To show this works, we're going to clear the environment. This tells R to forget all of the values and functions it currently knows: this does not clear away functions belonging to packages.



Basic introduction to R Dr. E.I. Walsh, 2019

Now, we 'source' our script. See how the functions now show up in the environment.

The screenshot shows the RStudio interface. In the top-left pane, there are two tabs: 'example_script.R' and 'example_script_containing_functions.R'. The code in 'example_script.R' is as follows:

```
180 for(i in 1:50) {example_data[i] <- nested_function(3,i)}
181 example_data
182
183 # Use the 'describe' function from the psych package to get the descriptive
184 # statistics associated with that data
185 describe(example_data)
186
187
188 source("example_script_containing_functions.R")
189
190
191
192
193
194
195
196
197
```

In the bottom-left pane, the console shows:

```
Console C:/Users/u4402564/Dropbox/Uni/talks/2019_03_CRAHW_intro_to_R/ ↵
> source("example_script_containing_functions.R")
>
```

In the top-right pane, the 'Packages' tab of the 'Session' menu is selected, showing the installed packages:

Name	Description	Version
abind	Combine Multidimensional Arrays	1.4-3
acepack	ace() and avas() for selecting regression transformations	1.3-3.3
ade4	Analysis of Ecological Data : Exploratory and Euclidean Methods in Environmental Sciences	1.7-4
AER	Applied Econometrics with R	1.2-4
akima	Interpolation of Irregularly and Regularly Spaced Data	0.6-2
Amelia	A Program for Missing Data	1.7.4
animation	A Gallery of Animations in Statistics and Utilities to Create Animations	2.4
aod	Analysis of Overdispersed Data	1.3
ape	Analyses of Phylogenetics and Evolution	3.4
arm	Data Analysis Using Regression and Multilevel/Hierarchical Models	1.8-6
aRiv	Interface to the arXiv API	0.516
assertthat	Easy pre and post assertions.	0.1
assertthat	Easy Pre and Post Assertions	0.2.0
automap	Automatic interpolation package	1.0-1.4

In the bottom-right pane, the 'Environment' tab is selected, showing the defined functions:

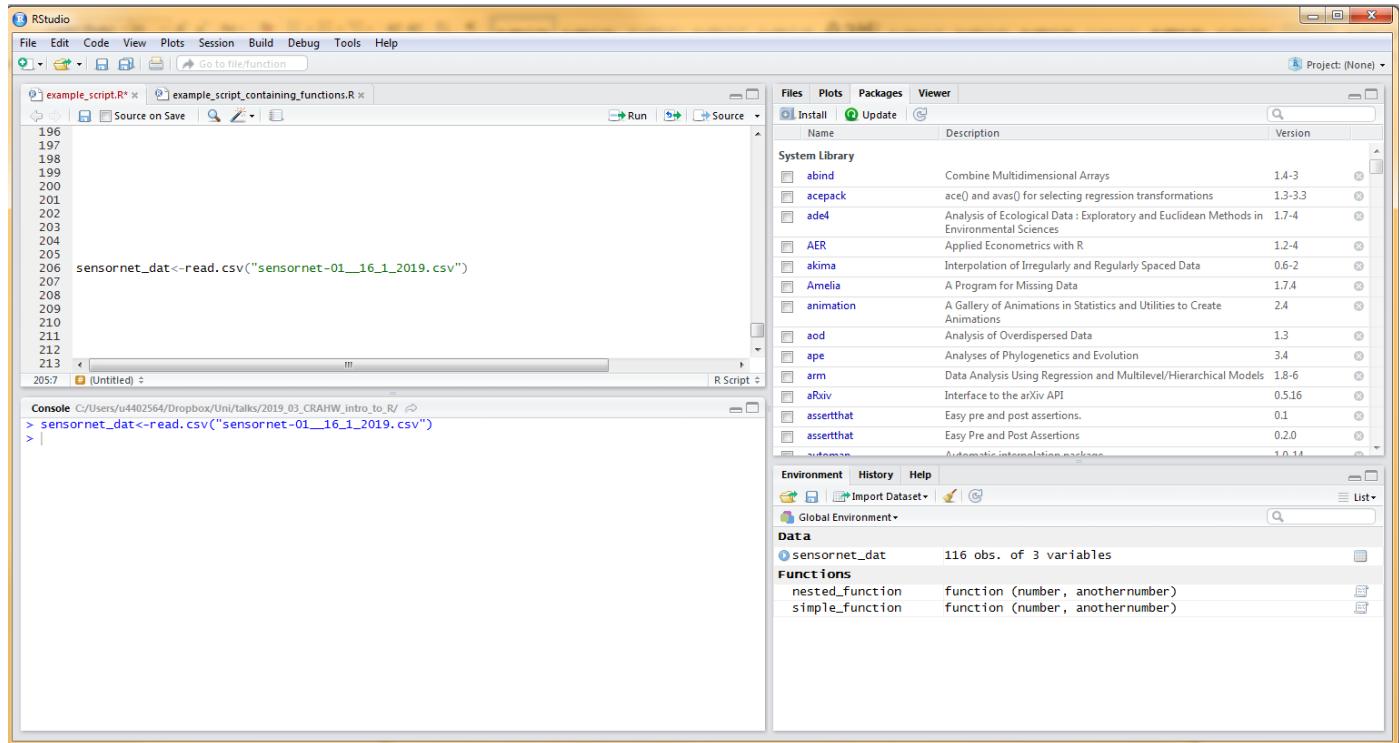
Functions	
nested_function	function (number, anothernumber)
simple_function	function (number, anothernumber)

Importing and working with datafiles

Very much like scripts: make sure you have set your working directory so R knows where to look for your data files.

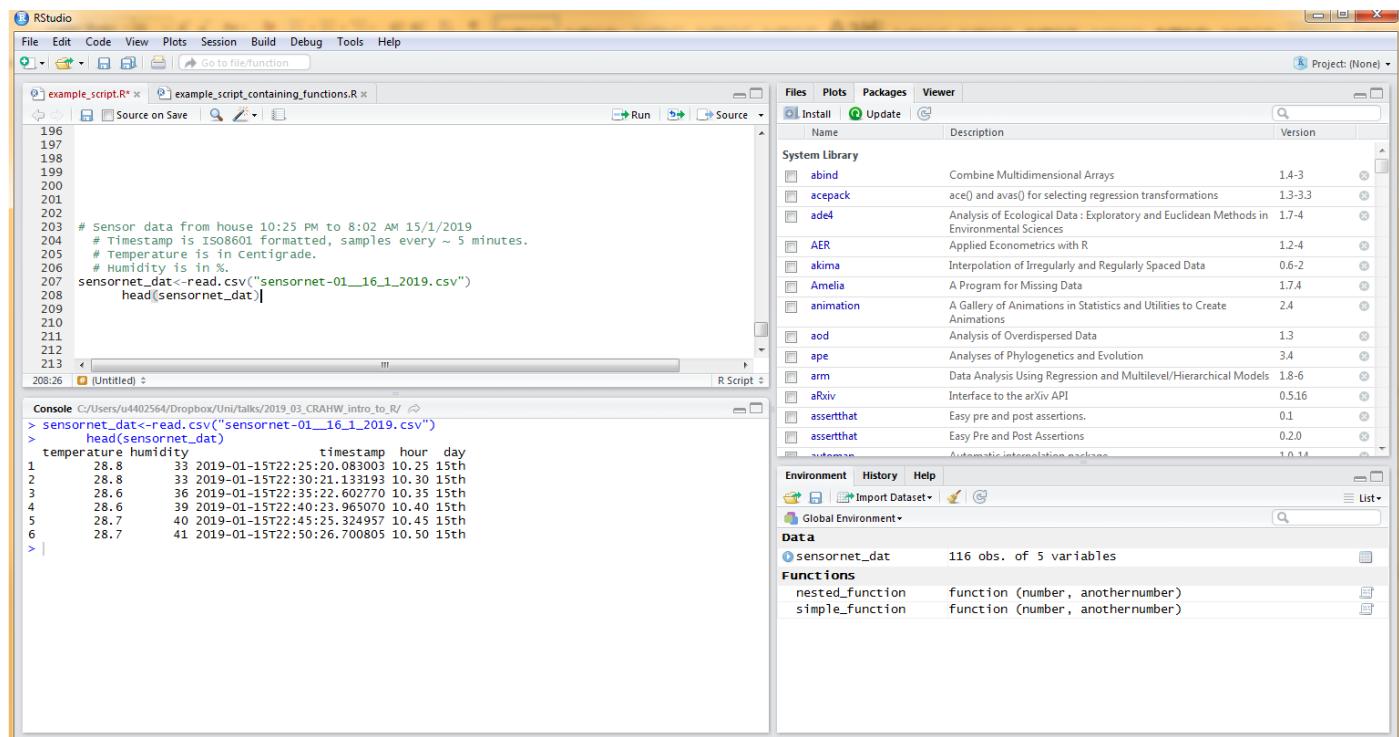
Assign the data to a variable: if you don't it just prints in console. Here it is a .csv so we're using 'read_csv'; can also read excel and SPSS files with the help of a package (too much detail for today).

It is best practise to append _dat to the variable name, so you always know which variables are data files.



ALWAYS use the 'head' function to look at the first few rows of the data file to ensure it has loaded correctly.

Also, it is best practise at this point to include descriptions of the dataset and each variable in comments for future reference.



Basic introduction to R Dr. E.I. Walsh, 2019

There are two ways to view a variable in a dataset; either using the \$ sign, or as a named index. They're both the same, for now we'll use the \$ sign because it is easiest.

To get a feel for the data, let's start by looking at the descriptive statistics for temperature and humidity using the `describe()` function.

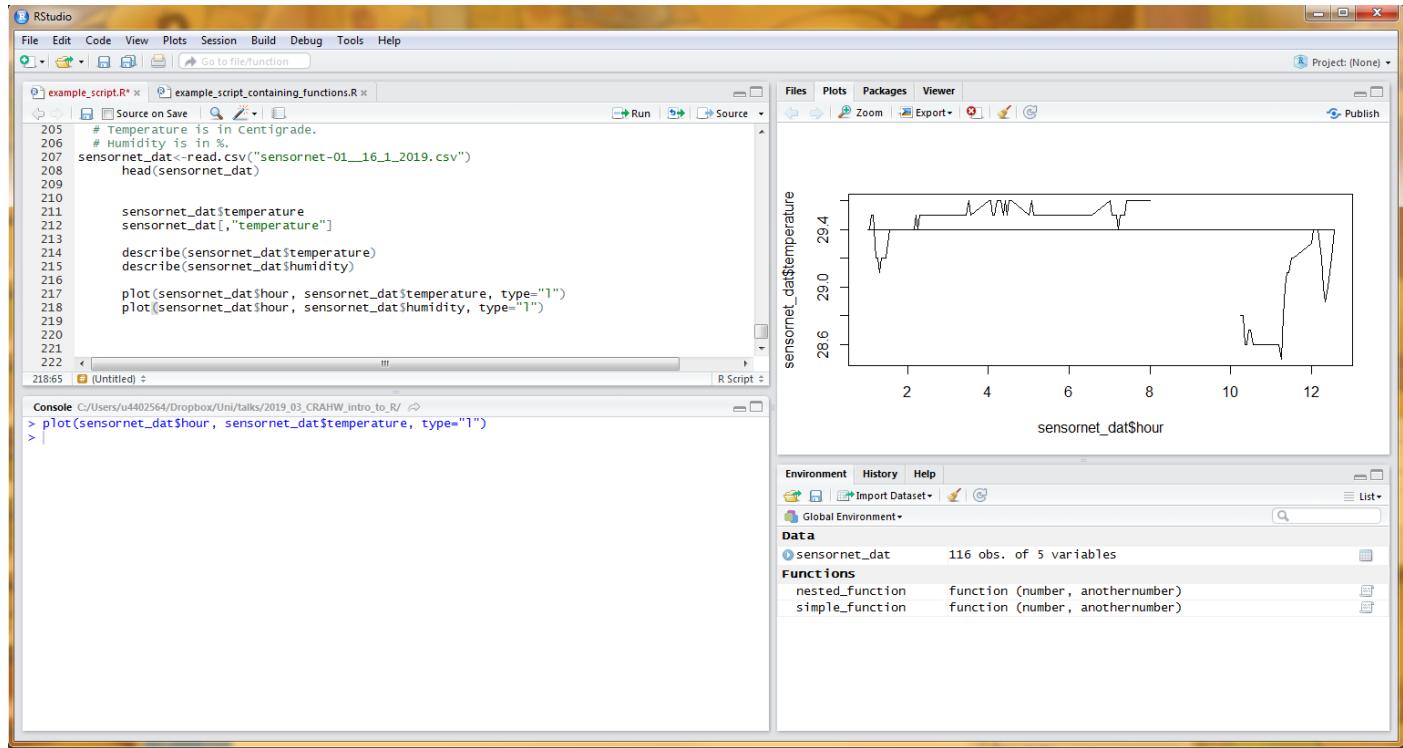
The screenshot shows the RStudio interface with the following components:

- File Menu:** File, Edit, Code, View, Plots, Session, Build, Debug, Tools, Help.
- Toolbar:** Includes icons for file operations like Open, Save, Run, and Source.
- Project Bar:** Shows "Project: (None)".
- Code Editor:** Displays an R script with code for reading a CSV file and calculating statistics for temperature and humidity. The line `describe(sensornet_dat\$temperature)` is highlighted.
- Console:** Shows the output of running the script, including the results of `describe` for temperature and humidity.
- Packages Tab:** Shows the "System Library" with a list of packages and their descriptions. Packages listed include abind, acepack, ade4, AER, akima, Amelia, animation, aod, ape, arm, arXiv, assertthat, and automap.
- Environment Tab:** Shows the global environment with objects sensornet_dat, nested_function, and simple_function.

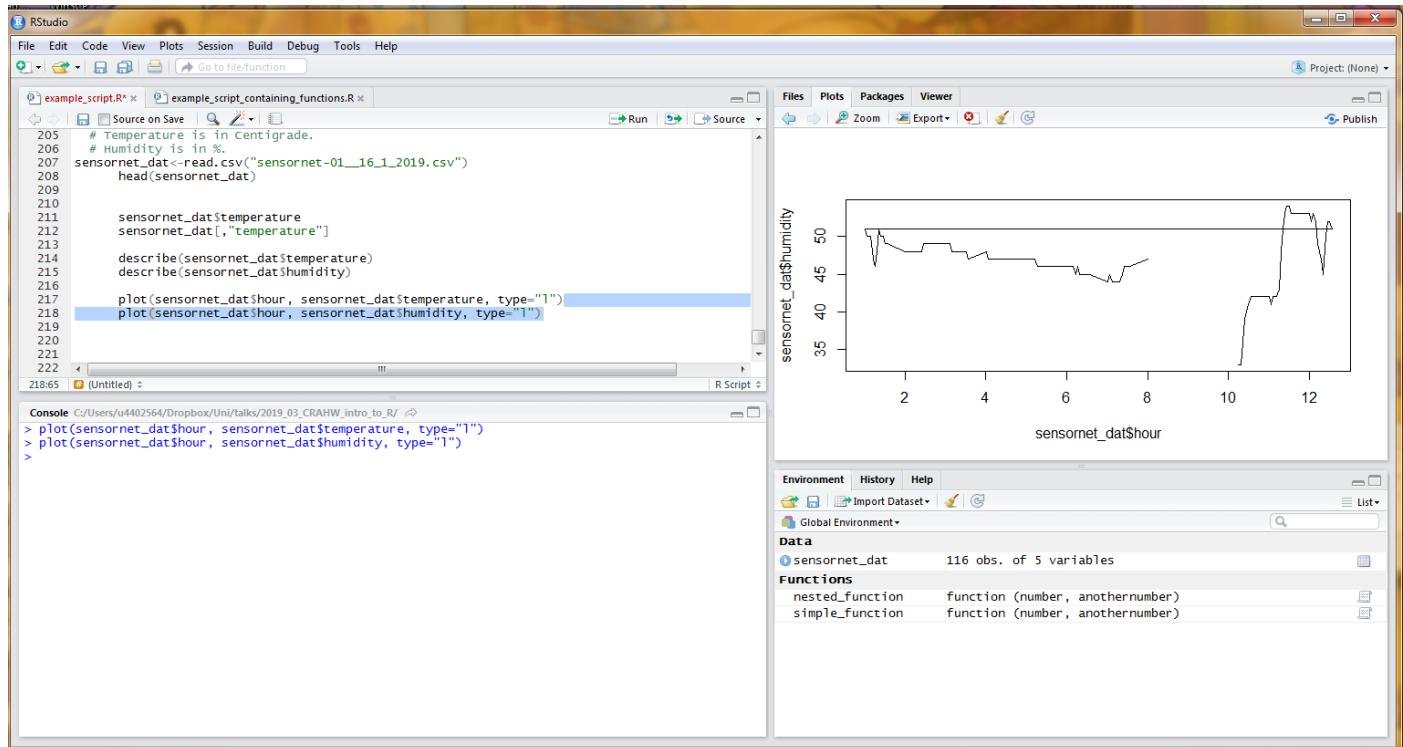
Basic introduction to R

Dr. E.I. Walsh, 2019

Now, let's plot temperature and humidity by hour. There are many amazing plotting packages in R, such as ggplots, but for now let's just use base graphics. We're using the `plot()` function, with the two variables we want to plot, then the additional argument `type="l"` to get a joined up line. Here is the plot for temperature...

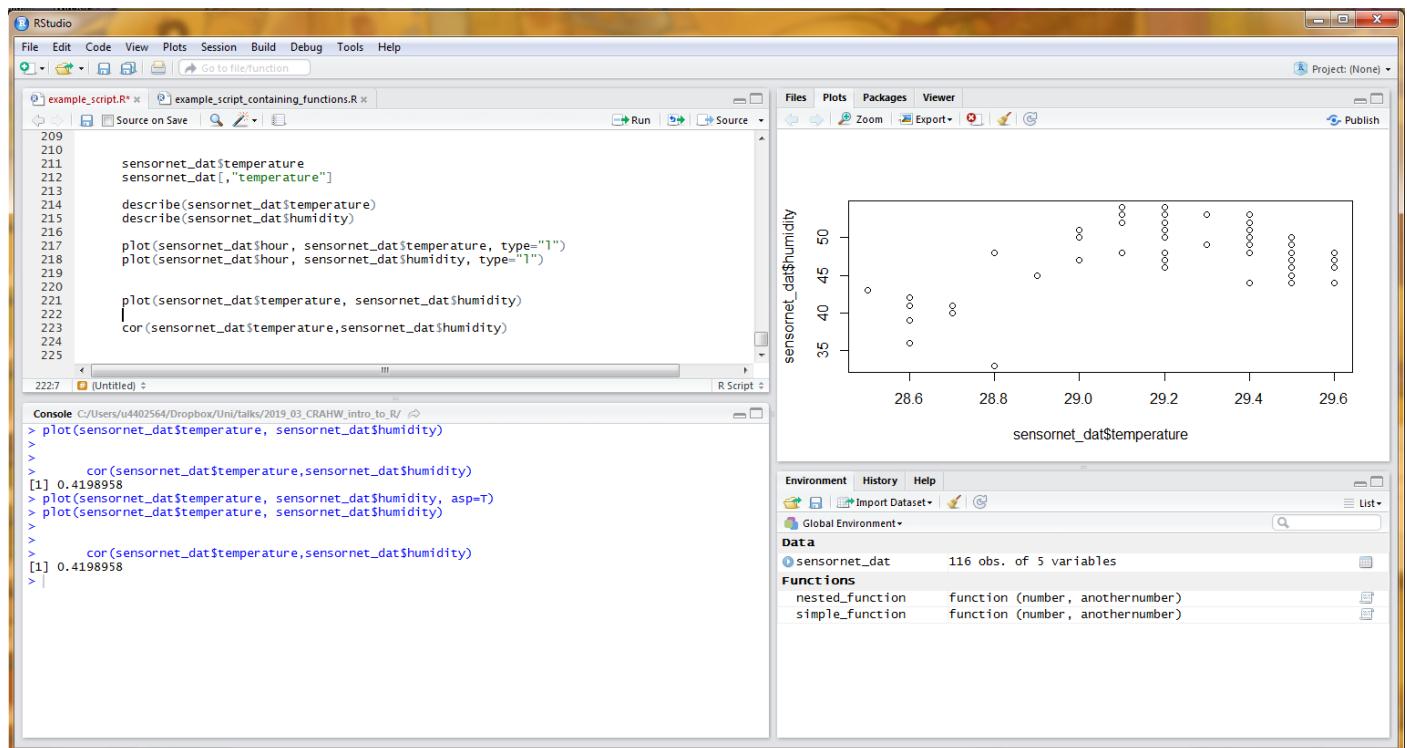


And here is the one for humidity.

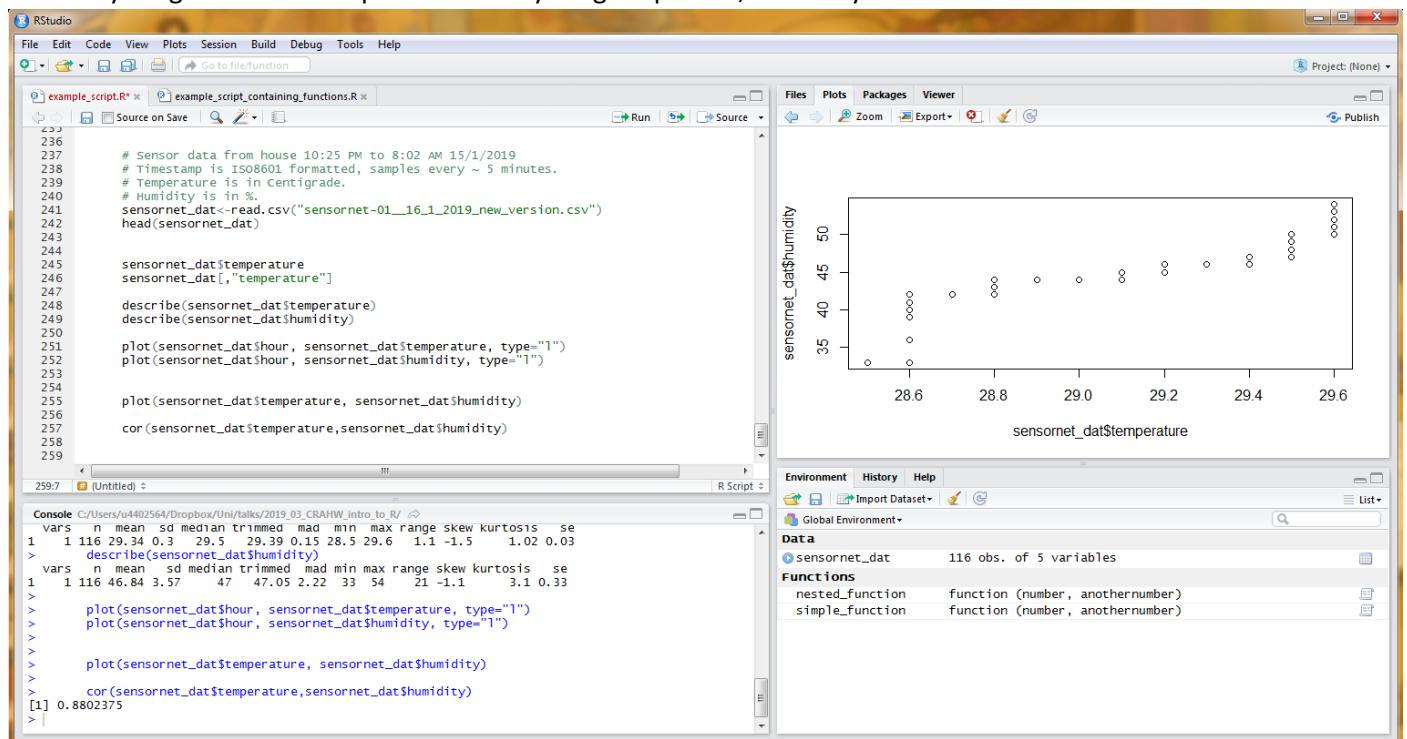


Basic introduction to R Dr. E.I. Walsh, 2019

Then we can run whatever analysis we want. For example, we might look at the correlation here using `cor()`.

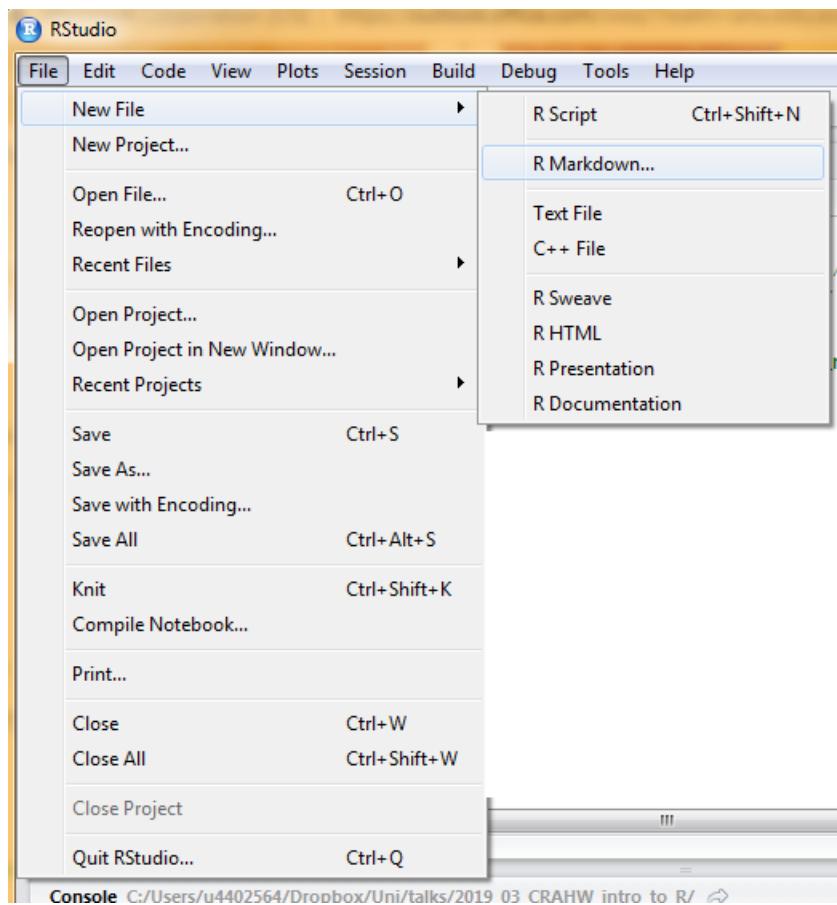


But, say we made a mistake with the sensornet data; turns out there's an updated datafile. Here's where the reproducibility comes in: we can just go back to where we loaded the file, change the name to the new one, then re-run everything else in our script below. Everything is updated, with very minimal effort!

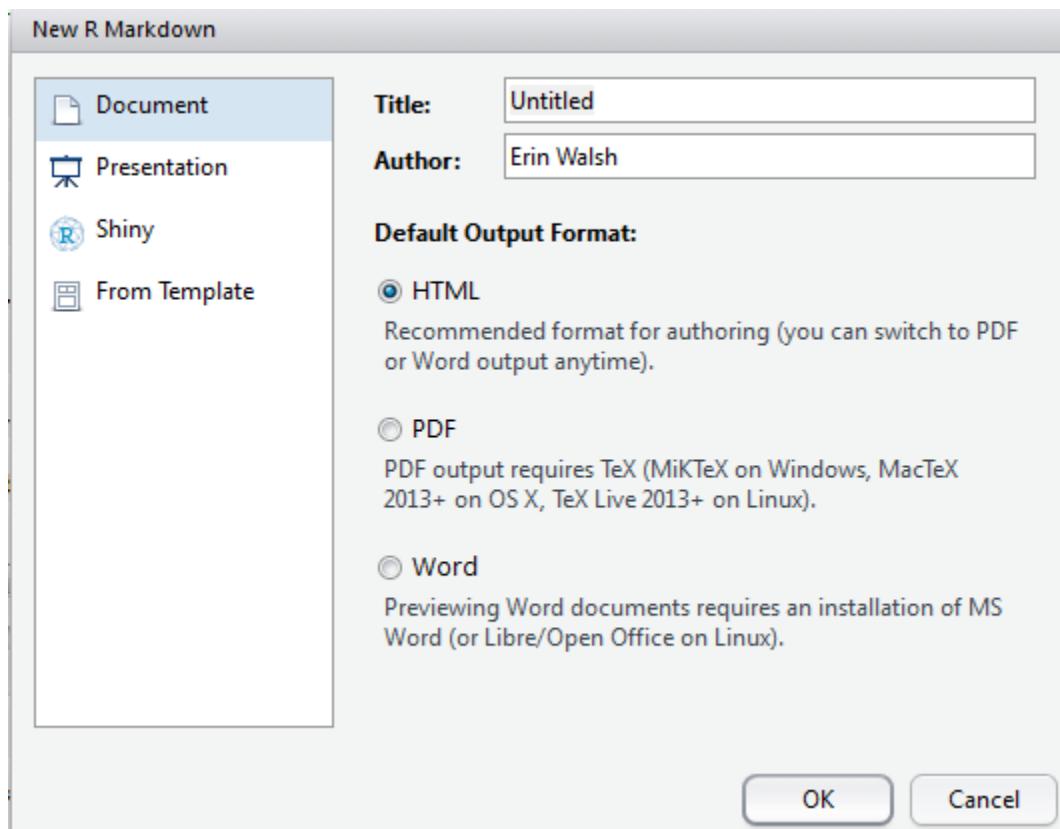


RMarkdown

We'll start with a new RMarkdown file.



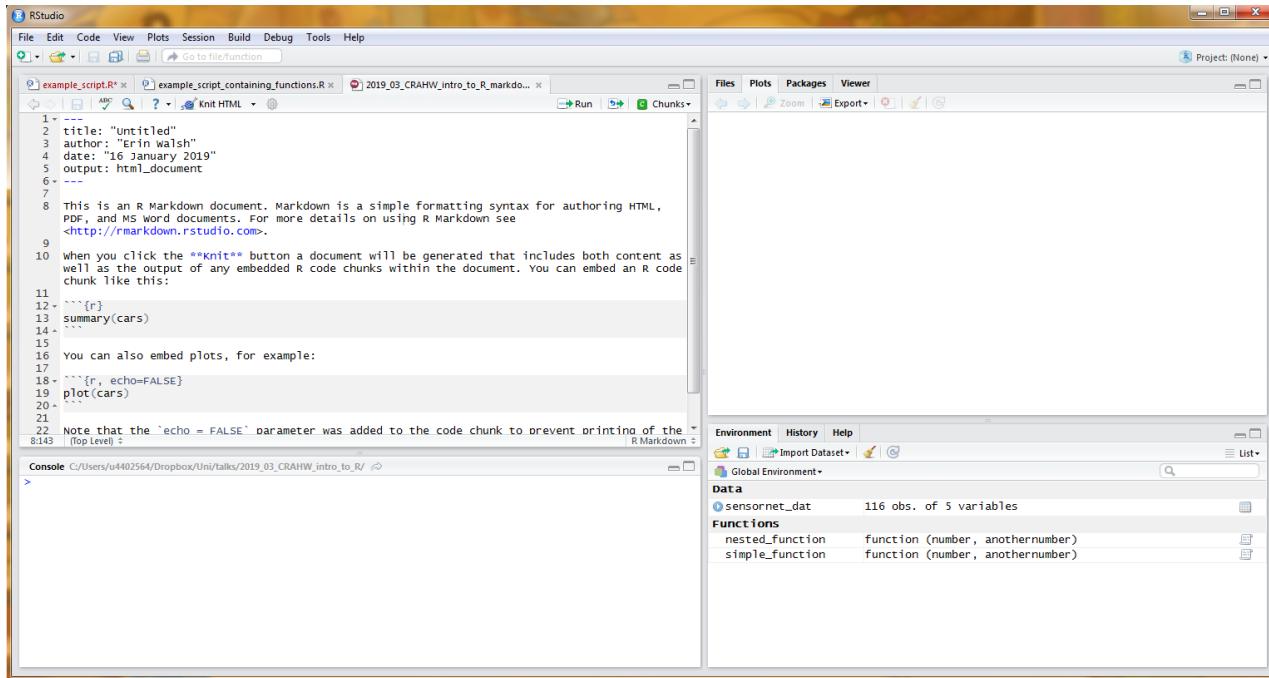
The next step should look something like this – should be fine to leave it on defaults.



Basic introduction to R

Dr. E.I. Walsh, 2019

They helpfully include information in the default document. Save this document with any name you'd like.



Click 'Knit html', and wait a moment. The console will show some code, and then a new window will pop up. It will look something like this:

The screenshot shows a browser window displaying the generated HTML document. The title is "Untitled". The content includes the author's name ("Erin Walsh") and the date ("16 January 2019"). A note explains what the R Markdown document is and how it works. Below this, there is a code block for the "summary(cars)" command, followed by the resulting output:

```
##      speed         dist
## Min.   : 4.0   Min.   : 2.00
## 1st Qu.:12.0   1st Qu.: 26.00
## Median :15.0   Median : 36.00
## Mean   :15.4   Mean   : 42.98
## 3rd Qu.:19.0   3rd Qu.: 56.00
## Max.   :25.0   Max.   :120.00
```

Below the code block, there is a note about embedding plots, followed by a scatter plot of "dist" vs "speed". The plot shows a positive correlation between the two variables.

dist

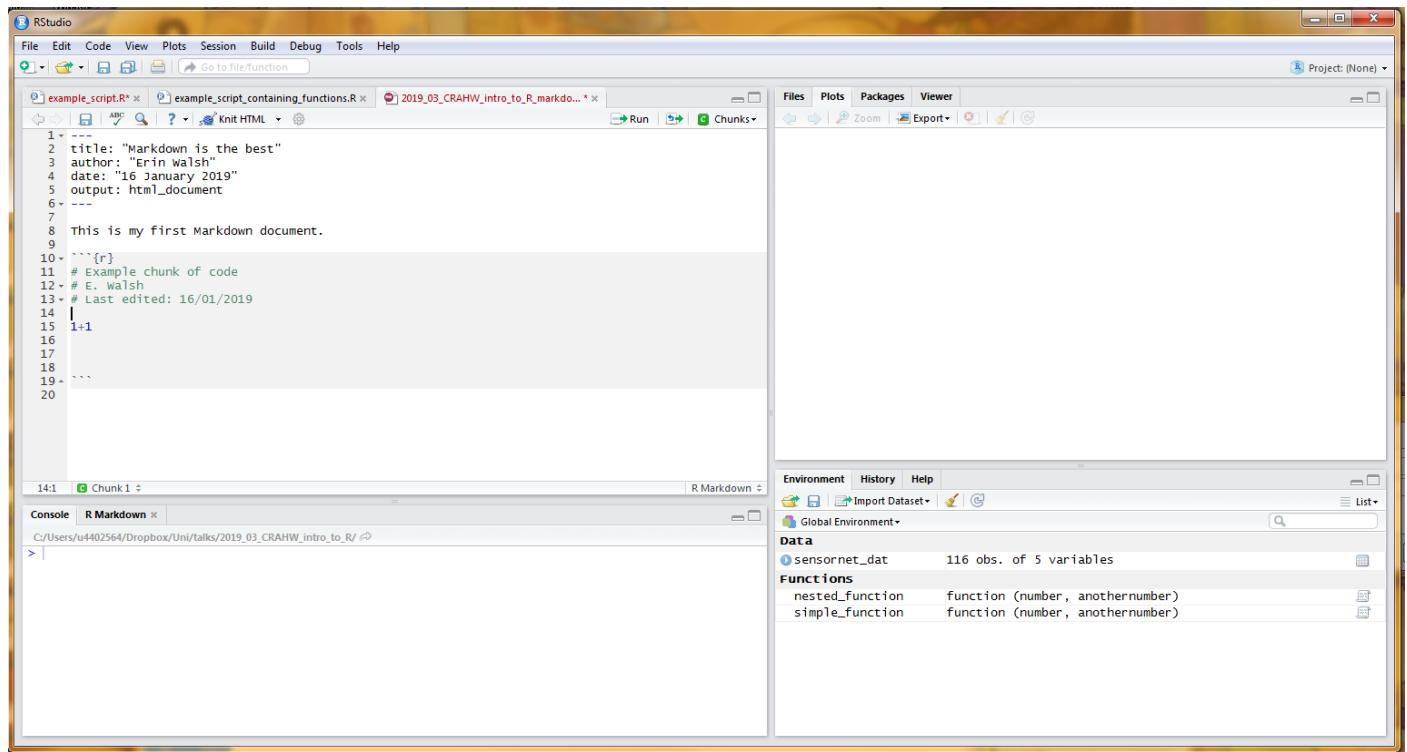
speed

Basic introduction to R

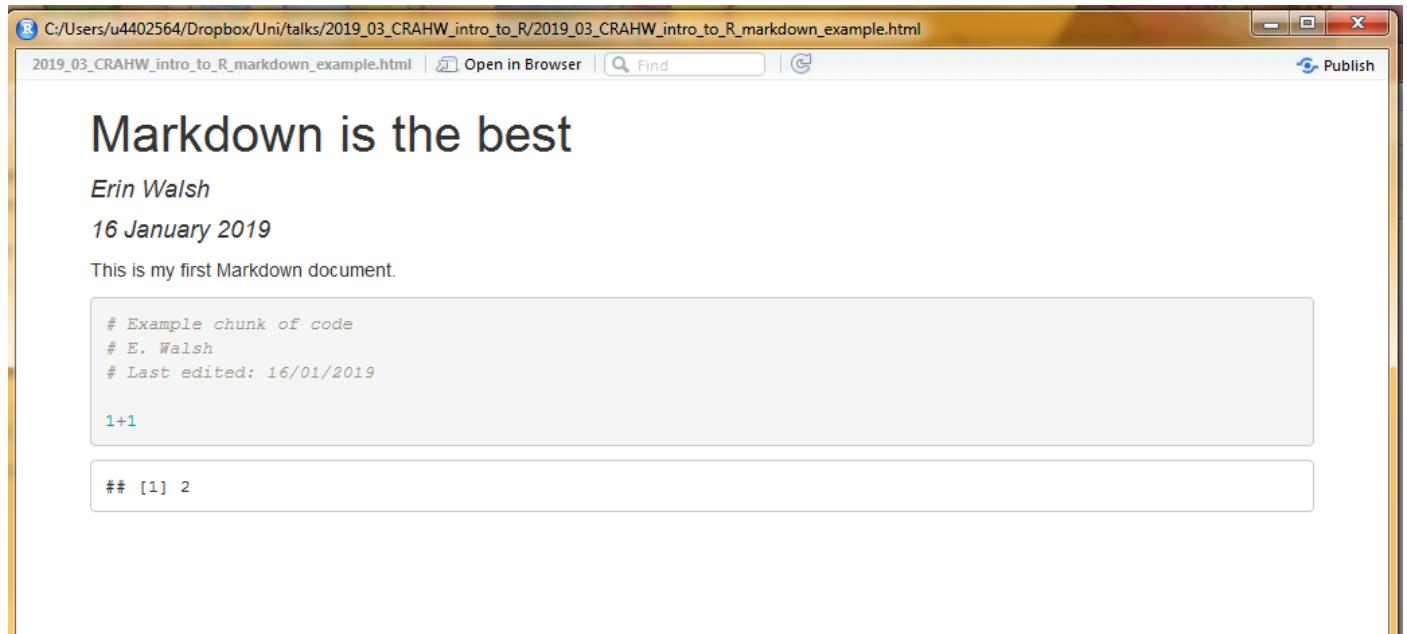
Dr. E.I. Walsh, 2019

You will also see it as an html file in the same folder as the underlying script.

Markdown has two parts: the plain text area (in white) and the code chunk (in grey). Let's edit the document by removing most of the example text, changing the text at the top, and adding something simple to the code chunk. It should look something like this:



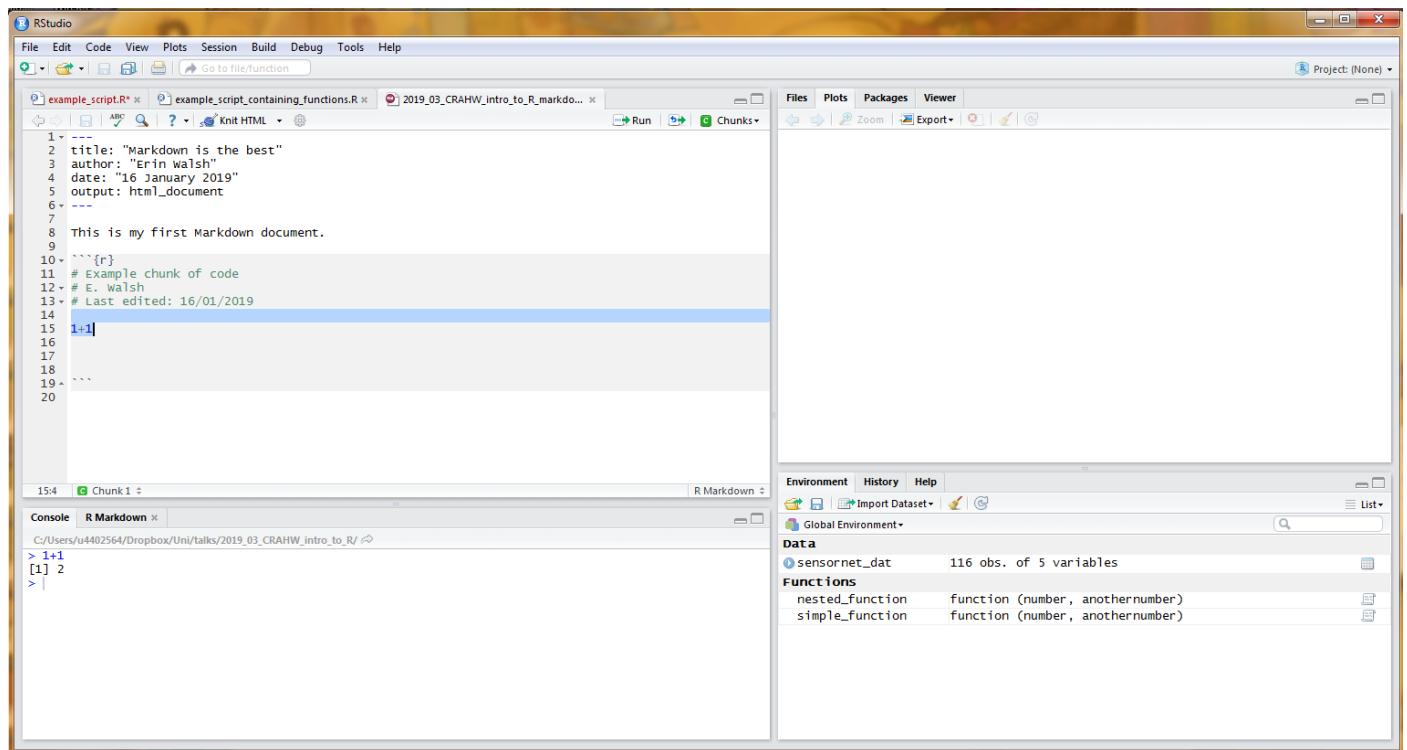
Knit this document and you should end up with something like this:



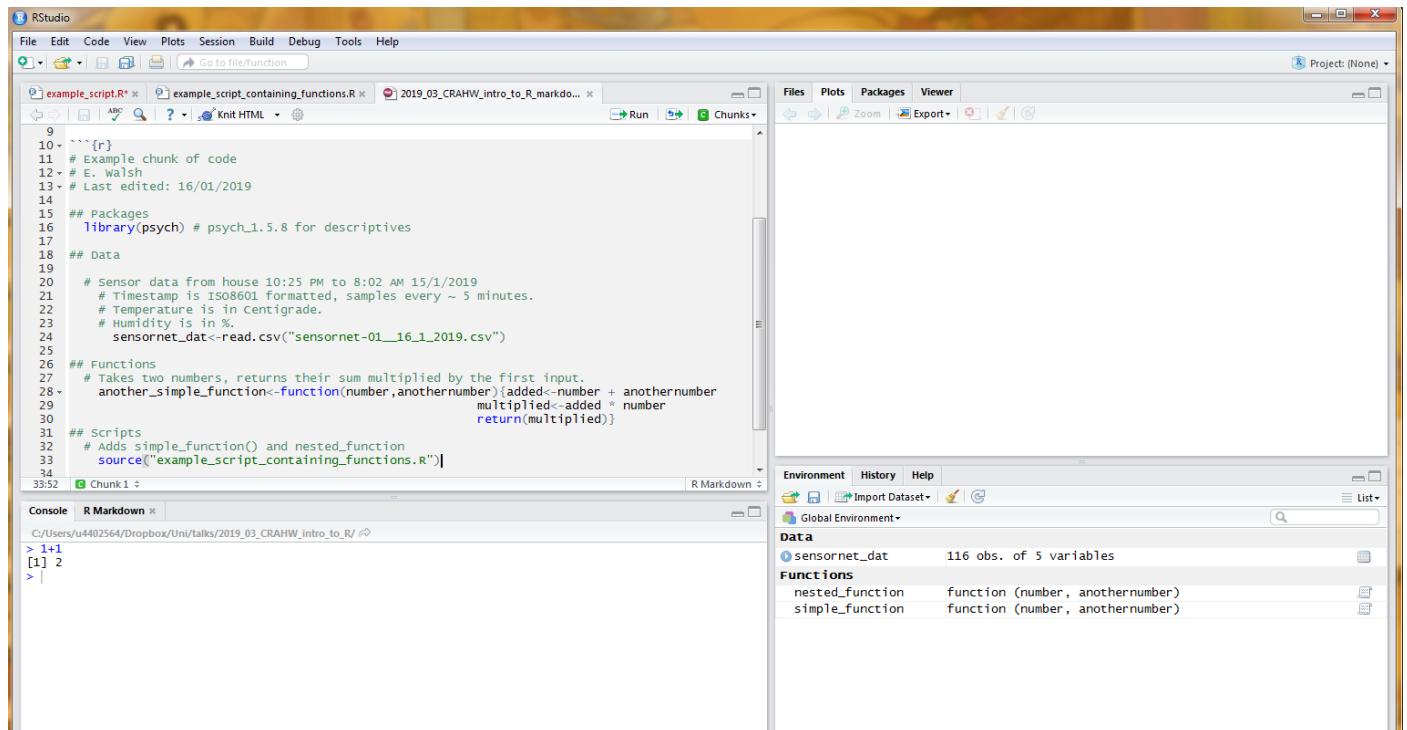
Basic introduction to R

Dr. E.I. Walsh, 2019

You don't need to compile it every time just to see the results. The grey code chunks work just like the normal script window does, so you can run things as you work (thanks to R's hybrid nature).



Now for the fun part: we can do anything and everything in the code chunks that we did for the scripts. Even importing scripts! Let's load some packages, data and scripts...



Basic introduction to R Dr. E.I. Walsh, 2019

...And run one of our functions from the script we saved over temperature

The screenshot shows the RStudio interface. In the top-left pane, there are two tabs: "example_script.R" and "example_scriptContainingFunctions.R". The "example_scriptContainingFunctions.R" tab is active, displaying the following R code:

```
22 # Temperature is in °Celsius
23 # Humidity is in %
24 sensornet_dat<-read.csv("sensornet-01_16_1_2019.csv")
25
26 ## Functions
27 # Takes two numbers, returns their sum multiplied by the first input.
28 another_simple_function<-function(number,anothernumber){added<-number + anothernumber
29 multiplied<-added * number
30 return(multiplied)}
31
32 ## Scripts
33 # Adds simple_function() and nested_function()
34 source("example_scriptContainingFunctions.R")
35
36 ## Analysis
37
38 # Loop over temperature to perform some operation
39 temperature_transformed<-vector()
40 for(i in 1:nrow(sensornet_dat)){
41   temperature_transformed[i]<-nested_function(2,sensornet_dat$temperature[i])
42 }
43
44 temperature_transformed
45
46 ...
47
```

In the bottom-left pane, the "Console" tab is active, showing the output of the R code. It includes a header and several lines of numerical data:

```
C:/Users/u4402564/Dropbox/Uni/talks/2019_03_CRAHW_intro_to_R/
> # Loop over temperature to perform some operation
> temperature_transformed<-vector()
> for(i in 1:nrow(sensornet_dat)){
+   temperature_transformed[i]<-nested_function(2,sensornet_dat$temperature[i])
+ }
>
> temperature_transformed
[1] 61.0 61.2 61.2 61.2 61.2 61.2 61.4 61.4 61.6 61.6 61.6 61.6 61.8 62.0 62.0 62.0 62.2 62.2 62.2 62.2 62.4
[24] 62.4 62.4 62.4 62.4 62.4 62.4 62.4 62.4 62.4 62.4 62.4 62.6 62.6 62.8 62.8 62.8 62.8 62.8 62.8 62.8 62.8
[47] 62.8 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0
[70] 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0
[93] 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.2 63.2 63.2 63.2 63.2 63.2 63.2 63.2 63.2 63.2 63.2 63.2 63.2 63.2
[116] 63.2
>
```

The right-hand side of the interface shows the "Environment" and "Global Environment" panes, which display the variables and functions defined in the script.

Let's see what happens to the descriptive statistics before and after:

This screenshot shows the same RStudio session after adding `describe()` calls to the script. The "example_scriptContainingFunctions.R" tab is active, and the "Console" tab shows the following additional R code and its output:

```
48:7  # Loop over temperature to perform some operation
49:7 for(i in 1:nrow(sensornet_dat)){
50:7   temperature_transformed[i]<-nested_function(2,sensornet_dat$temperature[i])
51:7 }
52:7
53:7 temperature_transformed
54:7
55:7 describe(temperature_transformed)
56:7
57:7 vars n mean sd median trimmed mad min max range skew kurtosis se
58:7 1 116 62.68 0.6 63 62.79 0.3 61 63.2 2.2 -1.5 1.02 0.06
59:7
60:7 describe(sensornet_dat$temperature)
61:7 vars n mean sd median trimmed mad min max range skew kurtosis se
62:7 1 116 29.34 0.3 29.5 29.39 0.15 28.5 29.6 1.1 -1.5 1.02 0.03
63:7
```

The "Environment" and "Global Environment" panes remain visible on the right.

Basic introduction to R Dr. E.I. Walsh, 2019

Ok, let's save the mean.

The screenshot shows the RStudio interface. In the top-left pane, there are three tabs: 'example_script.R*', 'example_script_containing_functions.R*', and '2019_03_CRAHW_intro_to_R_markdown.Rmd'. The 'example_script.R*' tab is active. The code in the editor is as follows:

```
38
39 # Loop over temperature to perform some operation
40 temperature_transformed<-vector()
41 for(i in 1:nrow(sensornet_dat)){
42   temperature_transformed[i]<-nested_function(2,sensornet_dat$temperature[i])
43 }
44
temperature_transformed
45
| describe(temperature_transformed)
46
47 describe(sensornet_dat$temperature)
48
49 mean_temp_before<-describe(sensornet_dat$temperature)$mean
50
51 mean_temp_after<-describe(temperature_transformed)$mean
52
53 mean_temp_before
54
55 mean_temp_after
56
57
58 ...
59
60
61
62
```

In the bottom-left 'Console' pane, the output of the R code is shown:

```
C:/Users/u4402564/Dropbox/Uni/talks/2019_03_CRAHW_intro_to_R/
> mean_temp_before<-describe(sensornet_dat$temperature)$mean
>
> mean_temp_after<-describe(temperature_transformed)$mean
>
> mean_temp_before
[1] 29.33966
> mean_temp_after
[1] 62.67931
>
```

The right-hand side of the interface includes the 'Files', 'Plots', 'Packages', and 'Viewer' tabs, along with a 'Project' dropdown set to '(None)'. Below these are the 'Environment', 'History', and 'Help' tabs, and a 'Global Environment' sidebar listing datasets like 'sensornet_dat', 'temp_humidity_1', and 'temp_humidity_2' with their respective structures and values.

Right now, knitting this is rather underwhelming – you just end up getting a (nicer) depiction of the analysis.

The screenshot shows a web browser window displaying the knitted R Markdown document at 'C:/Users/u4402564/Dropbox/Uni/talks/2019_03_CRAHW_intro_to_R/2019_03_CRAHW_intro_to_R_markdown_example.html'. The page content is identical to the RStudio console output above, showing the R code and its results. The browser has a standard toolbar with 'Open in Browser' and 'Find' buttons.

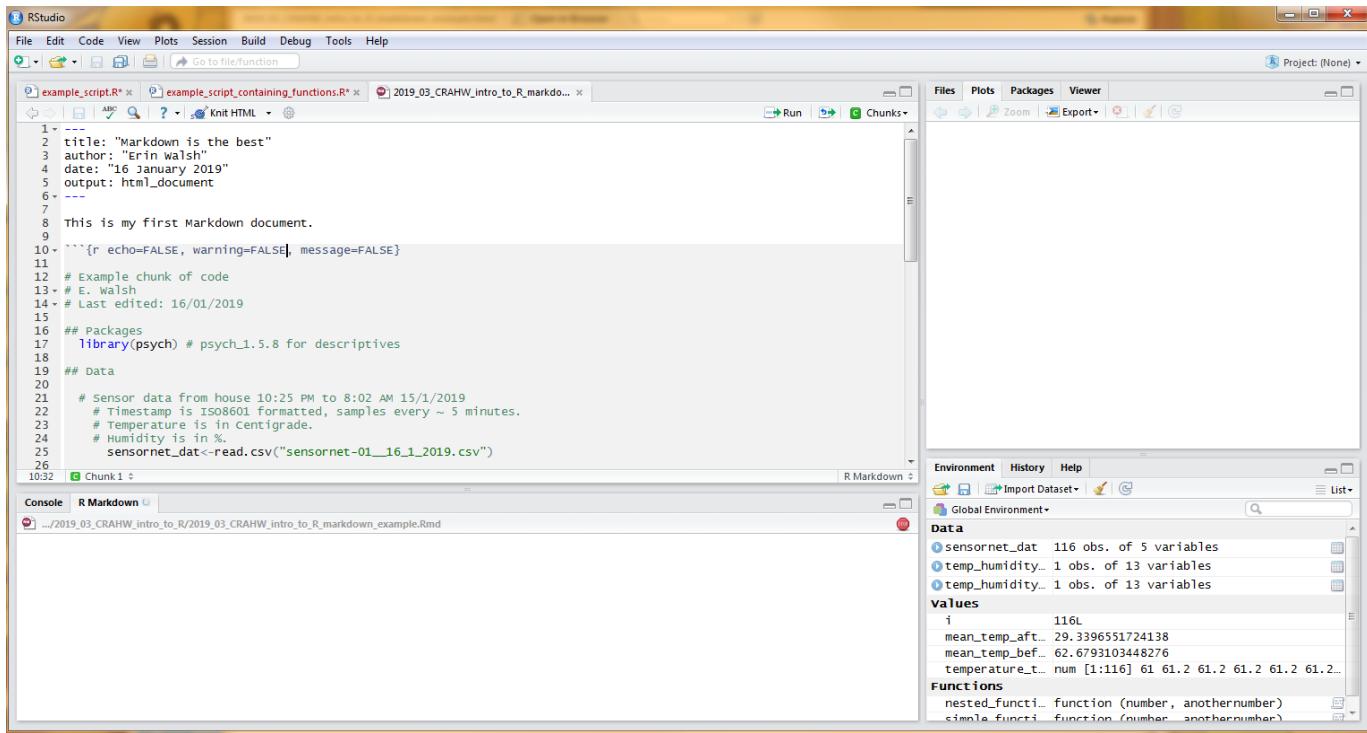
One night (15th-16th January 2019) we measured the temperature and humidity in our house every five minutes or so, overnight. The average overnight temperature was 29.34 degrees.

Then I brought the data into R and did some nefarious things with loops and functions.

Basic introduction to R

Dr. E.I. Walsh, 2019

So, let's tidy things up. First, let's hide the underlying code, and just show the output. It will still be run every time markdown is run, but doesn't show up. We do this by adding some arguments to the chunk header.



This gives us... Just the output.

The screenshot shows a web browser window displaying the generated HTML output. The title of the page is "2019_03_CRAHW_intro_to_R_markdown_example.html". The content of the page is as follows:

Markdown is the best

Erin Walsh
16 January 2019

This is my first Markdown document.

```
## [1] 61.6 61.6 61.2 61.2 61.4 61.4 61.2 61.2 61.2 61.2 61.2 61.0 61.6  
## [15] 62.0 62.2 62.2 62.4 62.4 62.6 62.8 62.8 62.6 62.4 62.0 61.8 62.0  
## [29] 62.2 62.4 62.8 62.8 62.8 63.0 63.0 62.4 62.4 62.2 62.4 62.4 62.6  
## [43] 62.8 62.8 62.8 62.8 62.8 63.0 62.0 62.8 63.0 63.0 63.0 63.0 63.0  
## [57] 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.2 63.0 63.2 63.0  
## [71] 63.0 63.2 63.2 63.2 63.0 63.2 63.0 63.2 63.2 63.0 63.0 63.0 63.0  
## [85] 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0 63.0  
## [99] 63.0 63.0 63.0 63.0 63.0 63.0 63.2 63.0 63.0 63.0 62.8 63.0 63.0 63.2  
## [113] 63.2 63.2 63.2 63.2
```

```
##   vars   n  mean   sd median trimmed mad min  max range skew kurtosis    se  
## 1     1 116 62.68 0.6      63   62.79 0.3   61 63.2    2.2 -1.5     1.02 0.06
```

```
##   vars   n  mean   sd median trimmed mad min  max range skew kurtosis    se  
## 1     1 116 29.34 0.3     29.5   29.39 0.15 28.5 29.6    1.1 -1.5     1.02  
##     se  
## 1  0.03
```

```
## [1] 29.33966
```

```
## [1] 62.67931
```

One night (15th-16th January 2019) we measured the temperature and humidity in our house ever five minutes or so, overnight. The average overnight temperature was 29.34 degrees.

Then I brought the data into R and did some nefarious things with loops and functions.

Now it appears the average overnight temperature was 62.68 degrees. Deadly.

Basic introduction to R Dr. E.I. Walsh, 2019

We can also turn off the output to just show the code:

The screenshot shows the RStudio interface with the following components:

- Top Bar:** File, Edit, Code, View, Plots, Session, Build, Debug, Tools, Help.
- Left Panel:** Shows the R Markdown file content. The code includes a title block, a note about hiding results, and a code chunk for reading a CSV file named "sensornet-01_16_1_2019.csv".
- Console:** Displays the R command `mean_temp_before` and its output [1] 62.67931, followed by `mean_temp_after` and its output [1] 29.33966.
- Output Area:** Shows the R Markdown code again, with syntax highlighting for functions like `nested_function` and `simple_function`.
- Environment Tab:** Shows the global environment with variables like `sensornet_dat`, `temp_humidity`, and `temperature_t`.
- Help Tab:** Shows help documentation for `nested_function` and `simple_function`.

Basic introduction to R Dr. E.I. Walsh, 2019

Or hide everything:

The screenshot shows the RStudio interface. In the top-left pane, there are two files open: 'example_script.R' and 'example_script_containing_functions.R'. The code in 'example_script.R' includes a YAML header and a code chunk with 'results='hide''. The code in 'example_script_containing_functions.R' reads a CSV file named 'sensornet_dat.csv'. In the bottom-left pane, the 'Console' tab is active, showing R code and its output. The output shows the mean temperature before and after reading the CSV file. In the bottom-right pane, the 'Environment' tab is selected, displaying the global environment with variables like 'sensornet_dat', 'temp_humidity...', and 'temperature_t...'. The 'Global Environment' section also lists functions like 'nested_functi...' and 'simple_functi...'.

```
1 ---  
2 title: "Markdown is the best"  
3 author: "Erin walsh"  
4 date: "16 January 2019"  
5 output: html_document  
6 ---  
7  
8 This is my first Markdown document.  
9 ```{r echo=FALSE, warning=FALSE, message=FALSE, results='hide'}```  
10 # Example chunk of code  
11 # E. walsh  
12 # Last edited: 16/01/2019  
13  
14 ## Packages  
15 library(psych) # psych_1.5.8 for descriptives  
16  
17 ## Data  
18  
19 # Sensor data from house 10:25 PM to 8:02 AM 15/1/2019  
20 # Timestamp is ISO8601 formatted, samples every ~ 5 minutes.  
21 # Temperature is in Centigrade.  
22 # Humidity is in %  
23 sensornet_dat<-read.csv("sensornet-01_16_1_2019.csv")  
24  
25  
26  
1063 [1] "Chunk 1" R Markdown
```

```
C:/Users/u4402564/Dropbox/Uni/talks/2019_03_CRAHW_intro_to_R/
```

```
> mean_temp_before<-describe(temperature_transformed)$mean  
>  
> mean_temp_after<-describe(sensornet_dat$temperature)$mean  
>  
> mean_temp_before  
[1] 62.67931  
>  
> mean_temp_after  
[1] 29.33966  
>
```

Environment | History | Help |

Data

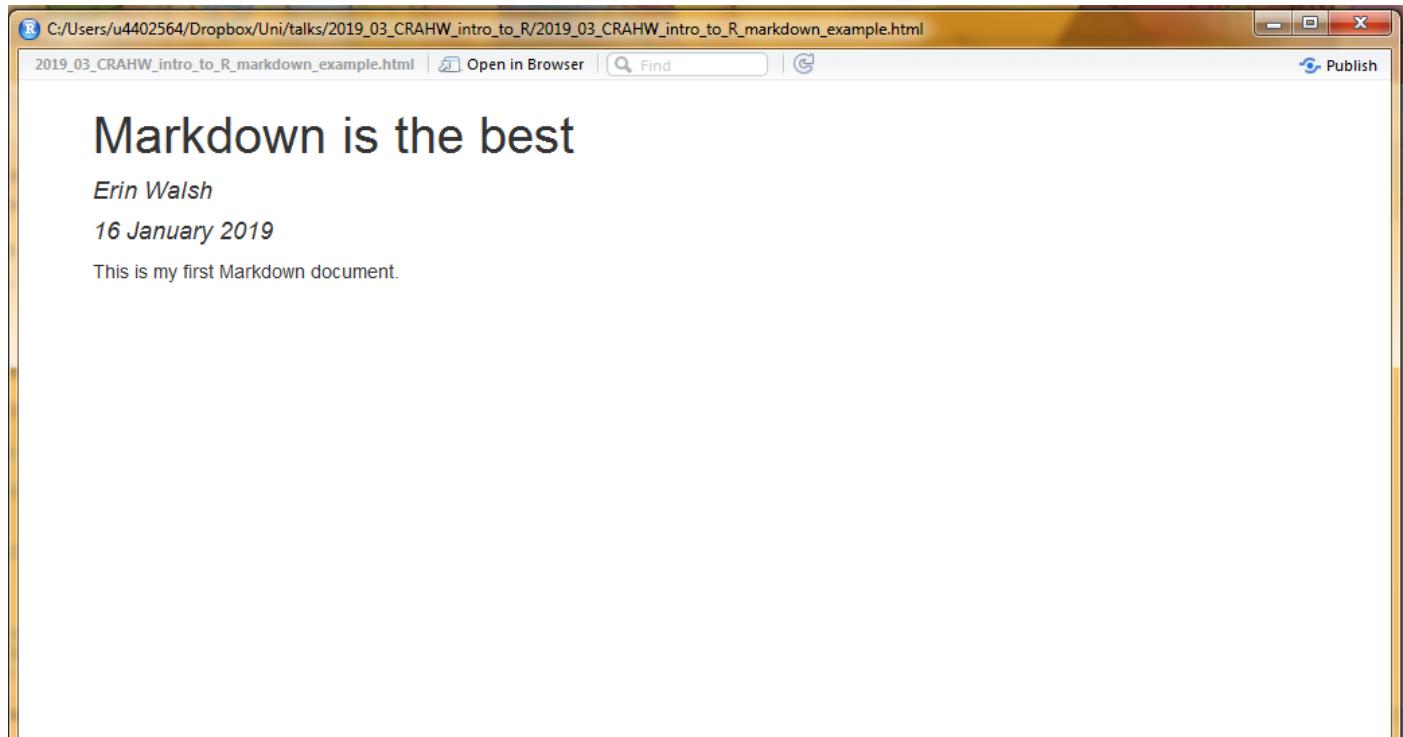
- sensornet_dat 116 obs. of 5 variables
- temp_humidity... 1 obs. of 13 variables
- temp_humidity... 1 obs. of 13 variables

Values

- i 116L
- mean_temp_aft... 29.3396551724138
- mean_temp_bef... 62.6793103448276
- temperature_t... num [1:116] 61 61.2 61.2 61.2 61.2...

Functions

- nested_functi... function (number, anothernumber)
- simple_functi... function (number, anothernumber)



Basic introduction to R

Dr. E.I. Walsh, 2019

Why would we do this? Well, you can also pipe output directly into text. This essentially means you can write your results section in R, with the results dynamically piped in.

The screenshot shows the RStudio interface. In the top-left pane, there are three tabs: 'example_script.R*', 'example_script-containing_functions.R*', and '2019_03_CRAHW_intro_to_R_markdown.Rmd'. The 'example_script-containing_functions.R*' tab is active. It contains R code that reads data from 'sensornet_dat' and performs calculations to find mean temperatures before and after a specific event. The 'R Markdown' tab at the bottom indicates the code is being processed. In the bottom-left pane, the 'Console' tab shows the execution of the R code, displaying the calculated mean temperatures: 62.67931 and 29.33966. The right-hand side of the interface features the 'Global Environment' pane, which lists variables like 'sensornet_dat', 'temp_humidity..', and 'temperature_t..', along with their types and values.

The screenshot shows a web browser displaying the generated HTML file. The title bar reads 'C:/Users/u4402564/Dropbox/Uni/talks/2019_03_CRAHW_intro_to_R/2019_03_CRAHW_intro_to_R_markdown_example.html'. The page content is as follows:

Markdown is the best

Erin Walsh

16 January 2019

This is my first Markdown document.

One night (15th-16th January 2019) we measured the temperature and humidity in our house ever five minutes or so, overnight. The average overnight temperature was 29.3396552 degrees.

Then I brought the data into R and did some nefarious things with loops and functions.

Now it appears the average overnight temperature was 62.6793103 degrees. Deadly.

Basic introduction to R

Dr. E.I. Walsh, 2019

You can also do operations within the in-text `r` tags, such as fixing the rounding in the above text. In general I'd recommend doing this mostly in the code proper, as you don't want your text getting too messy.

The screenshot shows the RStudio interface. The code editor contains R code that reads data from 'sensornet_dat' and performs calculations on 'temperature_transformed'. The console shows the execution of these commands and their results. The global environment pane shows several objects: 'sensornet_dat' (116 obs. of 5 variables), 'temp_humidity_' (1 obs. of 13 variables), and 'temp_humidity_1' (1 obs. of 13 variables). The values pane shows numerical values for 'i', 'mean_temp_aft...', 'mean_temp_bef...', and 'temperature_...'. The functions pane lists 'nested_funct_i...' and 'simple_funct_i...'.

```
43
46     describe(temperature_transformed)
47
48     describe(sensornet_dat$temperature)
49
50     mean_temp_before<-describe(sensornet_dat$temperature)$mean
51
52     mean_temp_after<-describe(temperature_transformed)$mean
53
54     mean_temp_before
55
56     mean_temp_after
57
58
59     ``
60
61
62 one night (15th-16th January 2019) we measured the temperature and humidity in our house ever five minutes or so, overnight. The average overnight temperature was `r round(mean_temp_before,2)` degrees.
64
65 Then I brought the data into R and did some nefarious things with loops and functions.
66
67 Now it appears the average overnight temperature was `r round(mean_temp_after,2)` degrees. Deadly.
68
69
```

```
C:/Users/u4402564/Dropbox/Uni/talks/2019_03_CRAHW_intro_to_R.R
> mean_temp_before<-describe(sensornet_dat$temperature)$mean
>
>     mean_temp_after<-describe(temperature_transformed)$mean
>
>     mean_temp_before
[1] 29.33966
>
>     mean_temp_after
[1] 62.67931
>
```

The screenshot shows a browser displaying a Markdown document. The title is 'Markdown is the best'. Below the title, there is author information ('Erin Walsh' and '16 January 2019'). The text content includes a paragraph about measuring temperature and humidity, followed by a note about bringing the data into R and performing calculations. The final sentence is 'Now it appears the average overnight temperature was 62.68 degrees. Deadly.'

Markdown is the best

Erin Walsh

16 January 2019

This is my first Markdown document.

One night (15th-16th January 2019) we measured the temperature and humidity in our house ever five minutes or so, overnight. The average overnight temperature was 29.34 degrees.

Then I brought the data into R and did some nefarious things with loops and functions.

Now it appears the average overnight temperature was 62.68 degrees. Deadly.

You can add as many code chunks with as many different behaviours and things in them as you would like. Just remember that R runs them from top to bottom, one after another, just like it is a script.

Basic introduction to R

Dr. E.I. Walsh, 2019

Let's finish the exercise by importing a couple more handy packages, and running a linear regression to describe the association between temperature and humidity. Stargazer is a great package for neatly summarising most linear models in a table.

The screenshot shows the RStudio interface. In the top-left pane, there are two tabs: "example_script.R" and "example_script_containing_functions.R". The "example_script_containing_functions.R" tab is active. The code in the editor is as follows:

```
40
41 temperature_transformed<-vector()
42 for(i in 1:nrow(sensornet_dat)){
43   temperature_transformed[i]<-nested_function(2,sensornet_dat$temperature[i])
44 }
45
46 temperature_transformed
47
48 describe(temperature_transformed)
49
50 describe(sensornet_dat$temperature)
51
52 mean_temp_before<-describe(sensornet_dat$temperature)$mean
53
54 mean_temp_after<-describe(temperature_transformed)$mean
55
56 mean_temp_before
57
58 mean_temp_after
59
60 library(stargazer) # Normally this would go up the top but just putting it here for demonstration
61
62 linear_model<-glm(humidity ~ temperature, data=sensornet_dat)
63
64 summary(linear_model)
65
66 ...
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
```

In the bottom-left pane, the "Console" tab is selected, showing the output of the R code. The output includes:

```
C:/Users/4402564/Dropbox/Uni/talks/2019_03_CRAHW_intro_to_R.R
Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) -262.289   15.609  -16.8 <2e-16 ***
temperature  10.536    0.532   19.8 <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Dispersion parameter for gaussian family taken to be 2.891588)

Null deviance: 1463.89 on 115 degrees of freedom
Residual deviance: 329.64 on 114 degrees of freedom
AIC: 456.35

Number of Fisher Scoring iterations: 2
```

The right-hand side of the interface shows the "Global Environment" pane, which lists variables and functions defined in the current session. Variables include "sensornet_dat", "temp_humidity..", "temp_humidity..", "linear_model", "mean_temp_aft...", "mean_temp_bef...", and "temperature_t...". Functions include "nested_function" and "stargazer".

We've made a new chunk below, and told it to render '`asis`' (as well as not showing constituent code). This is because Stargazer produces really lovely html tables; you can see we have asked stargazer to report our linear model with html output. It looks unreadable in the console window, but when we knit...

The screenshot shows the RStudio interface. In the top-left pane, the "example_script_containing_functions.R" tab is active. The code in the editor is as follows:

```
59
60
61 library(stargazer) # Normally this would go up the top but just putting it here for demonstration
62
63 linear_model<-glm(humidity ~ temperature, data=sensornet_dat)
64
65 summary(linear_model)
66 ...
67
68
69
70 One night (15th-16th January 2019) we measured the temperature and humidity in our house every five minutes or so, overnight. The average overnight temperature was `r round(mean_temp_before,2)` degrees.
71
72 Then I brought the data into R and did some nefarious things with loops and functions.
73
74 Now it appears the average overnight temperature was `r round(mean_temp_after,2)` degrees. Deadly.
75
76 Fine, but what is the actual relationship between temperature and humidity? A table can help us find out.
77
78 ``{r echo=FALSE, results='asis'}
79
80 stargazer(linear_model, type="html")
81
82
83
84
```

In the bottom-left pane, the "Console" tab is selected, showing the output of the R code. The output includes:

```
C:/Users/4402564/Dropbox/Uni/talks/2019_03_CRAHW_intro_to_R.R
<tr><td style="text-align:left"></td><td>(0.532)</td></tr>
<tr><td style="text-align:left"></td><td></td></tr>
<tr><td style="text-align:left">constant</td><td>-262.289<sup>**</sup></td></tr>
<tr><td style="text-align:left"></td><td>(15.609)</td></tr>
<tr><td style="text-align:left"></td><td></td></tr>
<tr><td colspan="2" style="border-bottom: 1px solid black;"></td></tr><tr><td style="text-align:left">Observations</td><td>116</td></tr>
<tr><td style="text-align:left">Log Likelihood</td><td>-226.173</td></tr>
<tr><td style="text-align:left">Akaike Inf. Crit.</td><td>456.346</td></tr>
<tr><td colspan="2" style="border-bottom: 1px solid black;"></td></tr><tr><td style="text-align:right"><sup>p</sup><sup><0.05;</sup></td><td><sup>**</sup><sup><0.01;</sup></td></tr>
</table>
|
```

The right-hand side of the interface shows the "Global Environment" pane, which lists variables and functions defined in the current session. Variables include "sensornet_dat", "temp_humidity..", "temp_humidity..", "linear_model", "mean_temp_aft...", "mean_temp_bef...", and "temperature_t...". Functions include "nested_function" and "stargazer".

We get a beautiful document.

2019_03_CRAHW_intro_to_R_markdown_example.html | Open in Browser | Find | Publish

Markdown is the best

Erin Walsh

16 January 2019

This is my first Markdown document.

One night (15th-16th January 2019) we measured the temperature and humidity in our house every five minutes or so, overnight. The average overnight temperature was 29.34 degrees.

Then I brought the data into R and did some nefarious things with loops and functions.

Now it appears the average overnight temperature was 62.68 degrees. Deadly.

Fine, but what is the actual relationship between temperature and humidity? A table can help us find out.

<i>Dependent variable:</i>	
	humidity
temperature	5.026*** (1.017)
Constant	-100.624*** (29.853)
Observations	116
Log Likelihood	-301.392
Akaike Inf. Crit.	606.784

Note: p<0.1; p<0.05; p<0.01

The formatting here can be customised substantially (lots of good resources online; this is a great start:

<https://www.markdowntutorial.com/>).