Day 2

Workflow and Github

Solveig Bjørkholt

28. June 2022

Plan for today

What we will learn today:

- What R and RStudio is.
- Writing and running code.
- Calling functions.
- Introduction to Rmarkdown and Jupyter Notebooks.
- Paths and directories.
- Github

What we will do today:

- Download R and RStudio (if you haven't already).
- Set up a shared repository in Github.
- Work with version control in Github.

What is R and RStudio?

R is a programming language, along with many other programming languages. Perhaps you've heard of some of them? Python, Java, Javascript, PHP, C, C++... We use these programming languages to communicate with the computer. Typically, there are lots of trade-offs with using one language over another, for example whether you want your code to be easily readable by a human, or whether it should make the requests run fast. Also, some programming languages are good for web development (e.g. Javascript and PHP), some work great for background processes (e.g. C and C++), and some are favorites for mathematics and statistical computing (e.g. python and R). We will be focusing on R in these sessions, though we have a long-term hope of integrating some python as well.

Learning R is a journey - and a quite frustrating journey at times, especially if it's the first programming language you learn. So why bother learning it? Why not just use excel or SPSS or another personal drag-and-drop favorite? Well...

- R is free. It's non-proprietary and requires no license. Down with the big corporations, knowledge for all!
- R enhances reproducibility. Anybody can re-run your code to use for their own applications, or to replicate your studies.

- R makes it easier to spot errors. Although errors are pain, we would rather have an error message early in the process than spotting a mistake two days before you think you've finished the project.
- R offers a large and supportive community to help you, both on the web and in real life. R-Meetups, RLadies, RStudio conference, you name it.

We often say that we "code in R", but we "work in RStudio". The reason for this is that R is the machine doing the calculations, while RStudio is the framework we work within. So if programming was a car, then R would be the engine and RStudio would be the bodywork.

Installing R and RStudio (adapted from Louisa Boulaziz)

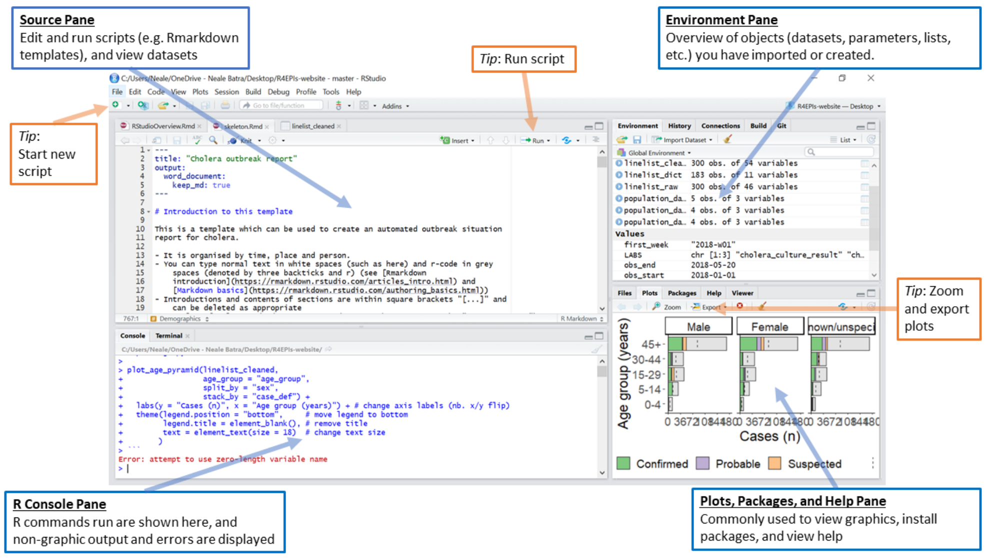
R

1. Open an internet browser and go to www.r-project.org
2. Click the "download R" link in the middle of the page under "Getting Started".
3. Select a CRAN location (a mirror site) and click the corresponding link. Here you should go to the Norway click on the link.
4. Click the "Download R for Windows" or the version for your computer. Link at the top of the page. If you have an old Mac, you need to read the next page carefully.
5. Click on the "install R for the first time" link at the top of the page (Windows). Mac users have to choose the version of R that is suitable for their software system. Old Macbook-users need to make sure that they click on the link corresponding to their software. If you are unsure about your software, click the apple in the left hand corner and go to "about this Mac". There it will say MacOS followed by the name of the software.
6. Click "Download R for Windows" and save the executable file somewhere on your computer. Run the .exe file and follow the installation instructions.
7. Now that R is installed, you need to download and install RStudio.

RStudio

1. Go to www.rstudio.com and click on the "Download RStudio" button.
2. Click on the "Download RStudio Desktop".
3. Click on the version recommended for your system, or the latest Windows version, and save the executable file. Run the .exe file and follow the installation instructions.

How RStudio works


Source Pane
Edit and run scripts (e.g. Rmarkdown templates), and view datasets

Tip: Run script

Environment Pane
Overview of objects (datasets, parameters, lists, etc.) you have imported or created.

Tip:
Start new script

Tip: Zoom and export plots

R Console Pane
R commands run are shown here, and non-graphic output and errors are displayed

Plots, Packages, and Help Pane
Commonly used to view graphics, install packages, and view help

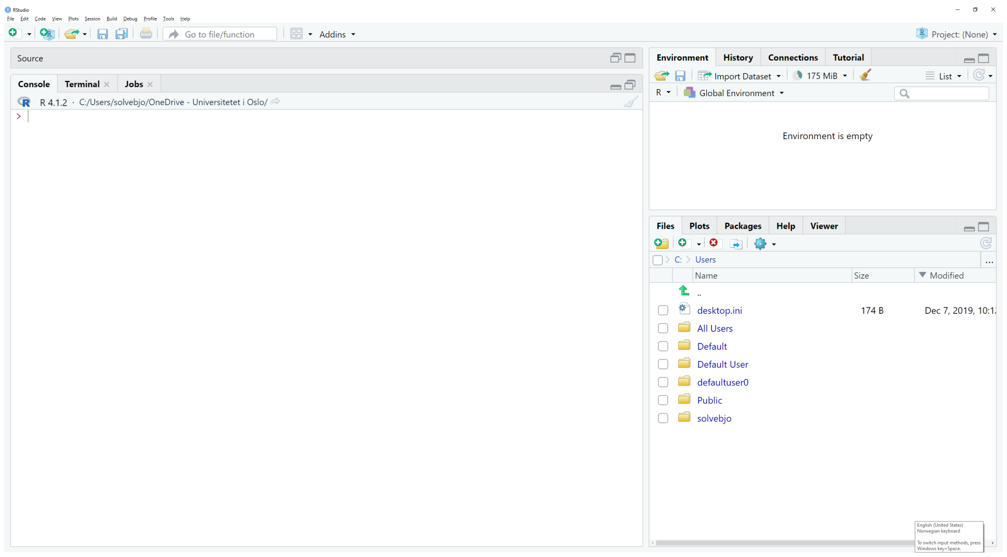
```
---
title: "Cholera outbreak report"
output:
  word_document:
    keep_md: true
---

# Introduction to this template

This is a template which can be used to create an automated outbreak situation
report for cholera.

- It is organised by time, place and person.
- You can type normal text in white spaces (such as here) and r-code in grey
    spaces (denoted by three backticks and r) (see [Rmarkdown
    introduction](https://rmarkdown.rstudio.com/articles_intro.html) and
    [Markdown basics](https://rmarkdown.rstudio.com/authoring_basics.html))
- Introductions and contents of sections are within square brackets "[...]" and
    can be deleted as appropriate
```

```
>
> plot_age_pyramid(linelist_cleaned,
+                  age_group = "age_group",
+                  split_by = "sex",
+                  stack_by = "case_def") +
+   labs(y = "Cases (n)", x = "Age group (years)") + # change axis labels (nb. x/y flip)
+   theme(legend.position = "bottom",     # move legend to bottom
+         legend.title = element_blank(), # remove title
+         text = element_text(size = 18)  # change text size
+       )
+ ...
>
Error: attempt to use zero-length variable name
> |
```

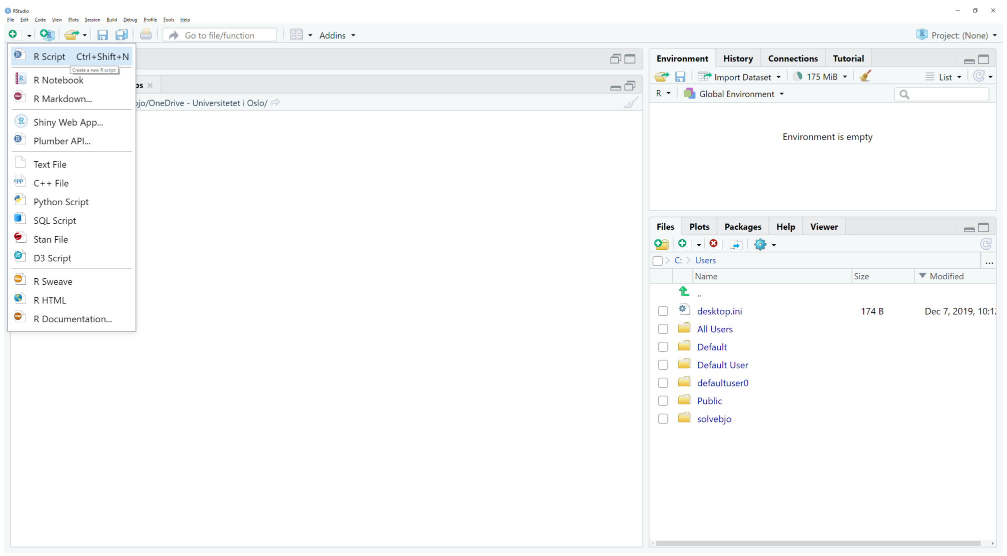
```
linelist_clea.. 300 obs. of 54 variables
linelist_dict   183 obs. of 11 variables
linelist_raw    300 obs. of 46 variables
population_da.. 5 obs. of 3 variables
population_da.. 4 obs. of 3 variables
population_da.. 4 obs. of 3 variables
Values
  first_week    "2018-W01"
  LABS          chr [1:3] "cholera_culture_result" "ch..
  obs_end       2018-05-20
  obs_start     2018-01-01
```

Writing and running code

When you have R and RStudio installed, open RStudio. Remember that we code in R, but work in RStudio, so this is the program you typically open. Your screen should look something like this:



We want to write our code in a script. A script is kind of like a word document, just for code. Scripts can be saved, stored and shared with others, which is preferable to the alternative - to write your code, execute it and then lose it once you close your programs and go home for the day. To start a script in RStudio, go to "File" and choose "New file", "R Script". Alternatively, click on the document with a green plus-sign in the top left hand corner and choose "R Script" - as shown below.

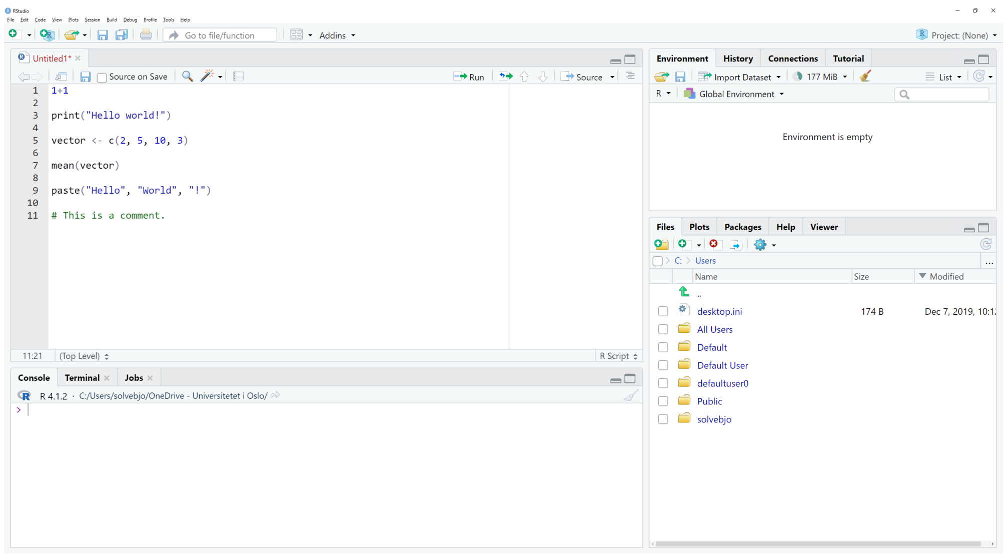


```
Environment is empty
```

Name	Size	Modified
..		
desktop.ini	174 B	Dec 7, 2019, 10:1
All Users		
Default		
Default User		
defaultuser0		
Public		
solvebjo		

The script emerges on the top left hand corner of your RStudio interface. Try typing the following into your script:

```r
1 + 1

print("Hello world!")

vector <- c(2, 5, 10, 3)

mean(vector)

paste("Hello", "World", "!")

# This is a comment.
```



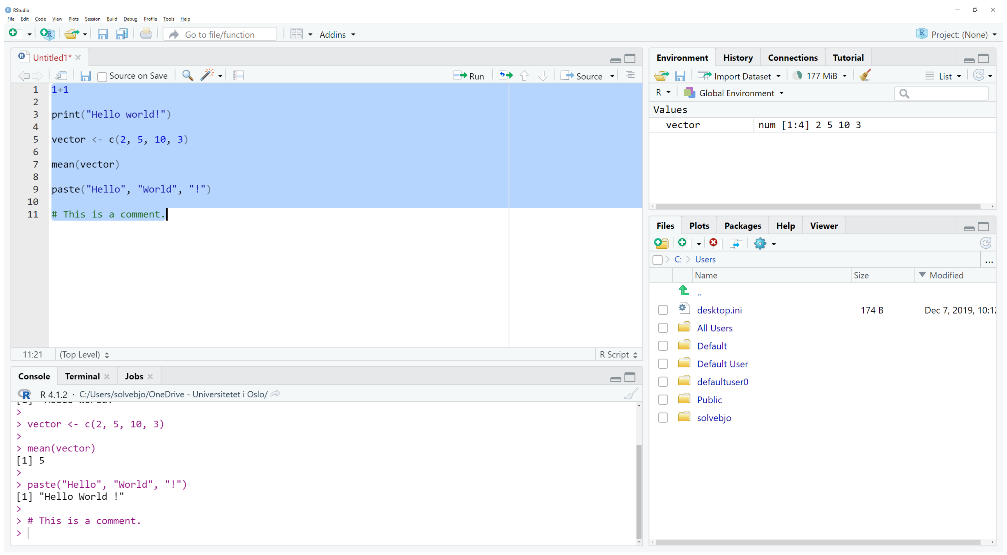
```r
1+1

print("Hello world!")

vector <- c(2, 5, 10, 3)

mean(vector)

paste("Hello", "World", "!")

# This is a comment.
```

```
R 4.1.2 · C:/Users/solvebjo/OneDrive - Universitetet i Oslo/
>
```

```
Environment is empty
```

Name	Size	Modified
..		
desktop.ini	174 B	Dec 7, 2019, 10:1
All Users		
Default		
Default User		
defaultuser0		
Public		
solvebjo		

Then mark the code and click CNTR+ENTER (i.e. run the code!). The output from your code appears in the bottom left hand corner of your RStudio interface. This is called the "Console". It's actually where R works, so you can view it as the engine of the car. All output from your code will appear here.

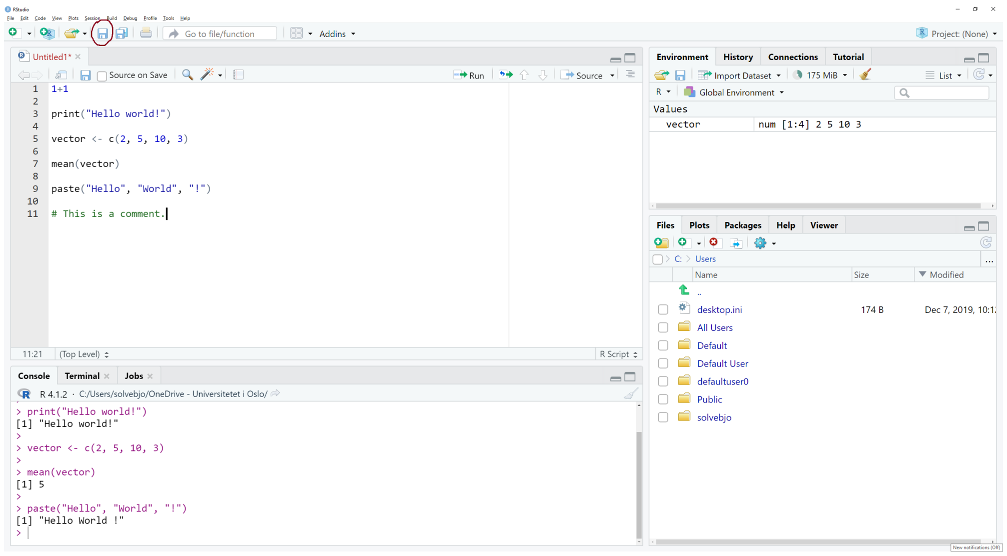


```
1+1

print("Hello world!")

vector <- c(2, 5, 10, 3)

mean(vector)

paste("Hello", "World", "!")

# This is a comment.
```

```
vector        num [1:4] 2 5 10 3
```

```
>
> vector <- c(2, 5, 10, 3)
>
> mean(vector)
[1] 5
>
> paste("Hello", "World", "!")
[1] "Hello World !"
>
> # This is a comment.
>
```

To save the script, hit the blue disc-symbol as shown in the picture below.



```
1+1

print("Hello world!")

vector <- c(2, 5, 10, 3)

mean(vector)

paste("Hello", "World", "!")

# This is a comment.
```

```
vector        num [1:4] 2 5 10 3
```

```
> print("Hello world!")
[1] "Hello world!"
>
> vector <- c(2, 5, 10, 3)
>
> mean(vector)
[1] 5
>
> paste("Hello", "World", "!")
[1] "Hello World !"
>
```

Objects

Notice that something has appeared in the upper right hand corner of your RStudio interface. This is an object. It has emerged because of this line in our script:

```
vector <- c(2, 5, 10, 3)
```

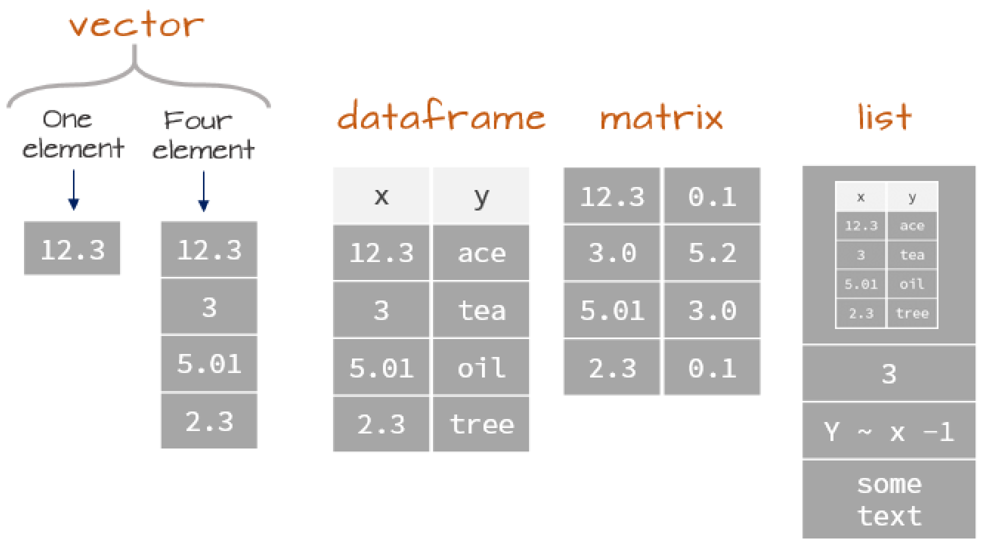
This line, because of the arrow (`<-`), assigns values to an object and stores it in the short-term memory of R. The box with the object is the short-term memory, also known as "Environment". We frequently assign values to objects in R, so it's good to familiarize yourself with this now. Think: Whenever you want to use an object later, make an object of it.

In general, to make an object, choose whichever name you like, add an arrow towards the name of the object, and then add the operation you want done. If you want to combine several elements, e.g. several numbers or several words, you need to put a c in front of your parentheses. `c` stands for "combine". For example:

```r
numberone <- 1

numbers <- c(1, 2, 3, 4, 5)

meanofnumbers <- mean(c(1, 2, 3, 4, 5))

names <- c("Billy", "Joe", "Vera")
```

Why did we call the object "vector" in the first place? Because it *is* a vector. You can think of vectors as a sequence of information, for example a bunch of numbers or a bunch of names. `numberone`, `meanofnumbers`, `numbers` and `names` are all vectors. There are other types of objects in R. The ones we will learn are: vector, dataframe, matrix and list. The difference between them is this:

- Vectors have the same data type and amount to *one* variable.
- Dataframes can have different data types and amount to *several* variables.
- Matrices have the same data type and amount to *several* variables.
- Lists have several types of data types and amount to *one* or *several* variables.[1]



To know what an object contains, try writing the name of the object and run it[2].

Data types

What does "data type" mean, then? Well, R has several data types and we will focus on four[3]. These are `numeric`, `integer`, `character`, `factor` and `logical`, and they are also called *classes*. They differ from each other in the types of data they store, be it strings, numbers or logical values.

Data type	Properties
`numeric`	contains decimal and whole numbers, also called "double"
`integer`	contains only whole numbers
`character`	contains character strings ("words")

[1]Some illustrations of data objects in R also operate with arrays, but we will not focus on that here.
[2]To "run a code" means to mark it and hit CNTR+ENTER
[3]Some people also add "complex" as a data type, but this is outside our scope.

Data type	Properties
`factor`	contains categories (either ranked or unranked)
`logical`	contains only two values, TRUE and FALSE

Numbers and integers can be one number, sequences of numbers, or mathematical expressions. For example:

```
100

1 + 100 + 20

1.4353  # commas are written as dots (.)

1.5 + 2.9 - 8.3

2 * 8  # multiply is written as star (*)

9/3  # divide is written as slash (/)

c(3, 100, 203.4, 15000)
```

Strings are sequences of characters, often words. We need to surround them in quotation marks. For example:

```
"A"

c("A", "B")  # If 'A' and 'B' were two categories, this object could also be a factor.

"Hello"

"We are learning R!"

c("Hello.", "We are learning R!")
```

Logical vectors take two values; TRUE and FALSE.

```
TRUE

FALSE

c(TRUE, FALSE, FALSE, FALSE, TRUE)
```

To check the data type, use the function `class()`.

```
class(100)
```

```
## [1] "numeric"
```

```
class("A")
```

```
## [1] "character"
```

```
class(FALSE)
```

```
## [1] "logical"
```

Logical operators

At last, I leave this here because it might be useful later.

Operator	Meaning
`==`	equals
`<`	less than
`>`	bigger than
`<=`	less than or equals
`>=`	bigger than or equals
`!=`	does not equal
`!x`	does not equal x
`|`	or
`&`	and

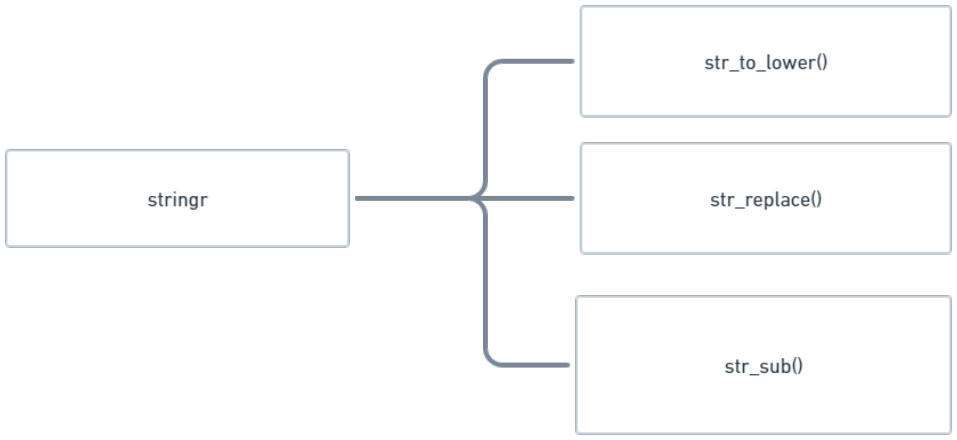
Calling functions

As you can see, we can use R as a calculator if we want to. Writing `1+1` gives `2` in the Console. But the real power of R comes from running functions. A function is a command that tells R what to with something. We've already seen a few functions so far, for example `print()`, `mean()` and `class()`. `print()` tells R to give us the output of an object, `mean()` tells R to calculate the average of an object, and `class()` tells R to give us the data type of an object.

Learning functions is a bit like learning vocabulary in a language, so don't fret if you find yourself grasping for the name of a function. It's the same as when you somewhat know a language and can't really recall a word. The solution? Ask someone or look it up on the internet.

Packages

Most functions we get from packages. Packages are bundles of functions that we import to R, and they are made by the wide community of people out there who work on R-content. For example, `stringr` is a package that gives us a lot of functions we can use to work with strings (i.e. character things). Some of these functions are `str_to_lower()`, which make all characters into lower case, `str_repalce`, which takes part of a string and replaces it with something else, and `sub_str()`, which takes out a part of a string.



To get a package into R, we first have to install it (from the world wide web), then import it (telling R that we want to use it in this script).

To install a package, use the function `install.packages()` and put the name of the package in quotation marks. You only need to do this *once* (unless you uninstall and reinstall R, or want to update your packages to newer versions). So no need to populate your script with `install.packages()`.

```
install.packages("string")
```

To import a package, use the function `library()`. Here, you do not need quotation marks.

```
library(stringr)
```

Once this is in order, we can start using function from the package `stringr`. A full overview of all the functions in a package is available on the internet through a document that all package-makers have to make for their package. In the case of `stringr`, a quick internet search leads us to this document.

Let's try some of the functions!

```
string <- "This is a string, meaning a sequence of characters, typically words. It is also a vector, si

print(string)
```

```
## [1] "This is a string, meaning a sequence of characters, typically words. It is also a vector, since
```

```
str_to_lower(string)  # Sets all characters to lower case.
```

```
## [1] "this is a string, meaning a sequence of characters, typically words. it is also a vector, since
```

```
str_replace(string, "This is a string, meaning a", "A string is a")  # Replaces some parts of a string
```

```
## [1] "A string is a sequence of characters, typically words. It is also a vector, since it contains i
```

```
str_sub(string, 1, 16)  # Picks out characters from place 1 to place 16.
```

```
## [1] "This is a string"
```

Your own functions

You can make your own functions too.

```r
make_lower_and_pick_start <- function(x) {

    x <- str_to_lower(x)
    x <- str_sub(x, 1, 16)

    return(x)

}

make_lower_and_pick_start(string)
```

```
## [1] "this is a string"
```

Installing LaTex

We're going to write in RMarkdown. We need LaTex to make RMarkdown reports in PDF, so we need to install this as well.

LaTex is a document preparation system, much like word, except it is more code-heavy and allows you to create beautiful documents. TinyTex is a custom LaTeX distribution, so this is what we'll install. See https://yihui.org/tinytex/ for more informaton.

To install TinyTex from R, we need to install the package `tinytex` and use the function `install_tinytex()`.

```r
install.packages("tinytex")
install_tinytex()
# to uninstall TinyTeX, run tinytex::uninstall_tinytex()
```

Introduction to RMarkdown and Jupyter Notebooks