

Reproducible Research in R

Contents

0.1	Learning outcomes	8
0.2	Workshop description	8
0.3	Prerequisite	8
0.4	Keywords	9
0.5	Schedule	9
0.6	Authors and copyright	9
1	Introduction to Rmarkdown	11
1.1	Rmarkdown core parts	11
1.1.1	YAML header	12
1.1.2	The R chunks	12
1.1.3	Everything else is plain old markdown	12
1.2	Challenge: Introduction 1	12
1.3	Rmarkdown ecosystem	13
1.3.1	Pandoc	13
1.3.2	YAML	13
1.3.3	LaTeX	13
1.3.4	Knitr	14
1.3.5	Rmarkdown	14
1.3.6	Bookdown	14
1.3.7	Others	14
1.4	Setup	14
1.5	Challenge: Introduction 2	15
1.6	Useful tips	15

2	Vanilla Markdown	17
2.1	Vanilla tags	17
2.2	Practice vanilla markdown	18
2.3	Challenge: Markdown 101	19
3	Rmarkdown	21
3.1	Embedding R code	21
3.1.1	echo and eval	24
3.1.2	include	25
3.1.3	results	25
3.2	Challenge: code chunks	27
4	Git and GitHub introduction	29
4.1	Github setup	31
4.2	RStudio setup	35
4.3	GitHub things	35
4.4	More git	35
4.5	git no nos	36
4.6	Starting with git	36
4.6.1	Configuring git	36
4.6.2	Intiating git repository	37
4.6.3	First commit	38
4.7	Which files to commit?	42
5	The R chunk 2	43
5.1	Working with code chunks	43
5.2	Figures related chunk options	47
5.3	Challenge: code chunks	49
5.4	More useful chunk options	50
5.4.1	Stand alone Rmarkdown file	51
5.5	Challenge: more code chunks	52
5.6	References	52

<i>CONTENTS</i>	5
6 More Rmarkdown	53
6.1 Setup	53
6.1.1 Setting global chunk options	54
6.1.2 Loading libraries	54
6.1.3 Downloading the data	54
6.2 Challenge: 1	55
6.3 Exploring the data	56
6.4 Challenge: 1	57
6.5 Visualising the data	57
7 YAML header	59
7.1 Tables	61
7.2 R slides - ioslides	61
7.3 Extras	61
8 Bookdown	63
9 Bibliographies	65
10 Work in progress	67
10.1 Tabbed sections	67
10.2 How documents looks	67
10.3 Figure options via yaml	68
10.4 tables Rmarkdown	68

Reproducible Research in R

- Level: beginner-intermediate
- Duration: 6 hours
- Student numbers: 25-30

Welcome to the Reproducible Research in R (RRR)¹ workshop. The main aim of this workshop is to set you on the right path of making your research more reproducible and shariaable. Reproducible research means that future you and anyone else will be able to pick up your analysis and reproduce the same results including figures and tables. Reproducible research also implies well documented reasearch, your code should be well commented and the reasons behind functions and methods should be well explained through out the analysis. The communication aspect should not be after thought, is should stay with your analysis as you are going through it. Rmarkdown is a way of literal programing² that keeps code and words and sentences together. The other important aspect that goes hand and hand with reproducibility is ability to easily collaborate and share your analysis. We are going to repurpose git version cotrol tool and leverage GitHub remote hosting provider for managing and sharining our work. Git + GitHub will provide very powerful way for global collaboration and exposure of your work. In this workshop we are going to verstion control our work and push it to github, which can then be accessible by your collaborators and supervisors. Git + GitHub should become integral part of your workflow.

The RRR course given by the Monash Bioinformatics Platform³ for the Monash Data Fluency⁴ initiative. Our teaching style is based on the style of The Carpentries⁵.

- HTML versoin⁶
- PDF version⁷

¹<https://github.com/MonashDataFluency/r-rep-res>

²<http://www.literateprogramming.com/knuthweb.pdf>

³<https://www.monash.edu/researchinfrastructure/bioinformatics>

⁴<https://monashdatafluency.github.io/>

⁵<https://carpentries.org/>

⁶<https://monashdatafluency.github.io/r-rep-res/>

⁷<https://monashdatafluency.github.io/r-rep-res/Reproducible-Research-in-R.pdf>

0.1 Learning outcomes

Attendees will learn how to:

- write vanilla markdown, Rmarkdown and bookdown documents
- use `knitr`, `rmarkdown` and `bookdown` R packages to build various document types including PDF, HTML and DOCX
- create reproducible Rmarkdown documents leveraging `.Rproj` and `.RData`
- include inline citation and full references list in to Rmarkdown document using `.bib` files
- create presentations from Rmarkdown documents that include R code
- work with `git` version control tool
- create reproducible and “backed up” analysis via remote repositories (e.g github)

0.2 Workshop description

This workshop is an introduction to writing and communicating research using Rmarkdown. Rmarkdown is an easy way to create documents that include your R code and its output such figure and tables. Rmarkdown is a single documents that can be “knitted” and shared as various document types such as PDF and HTML. Rmarkdown supports scientific writing such as use of citations and figure cross-referencing. Rmarkdown can also be used to create presentations that include your R code and its output. We will also cover bookdown, which is an extension to Rmarkdown that allows creation of larger documents such as books with multiple chapters.

In this workshop we will also cover git version control tool⁸ that can help with organising and “checkpointing” Rmarkdown documents, associated R code and data. Git is not a back up system, but it does allow one to retrieve older versions of your work. Git together with remote repositories like GitHub⁹ can provide centralised location for your research. All together Rmarkdown, git and github can enable reproducible research that is visible and accessible by greater public including supervisors and management.

0.3 Prerequisite

This is an introductory level workshop, however some prior exposure to R and familiarity with RStudio is assumed. It is highly recommended that you read this article in full¹⁰, if you have to prioritise, read at least these section (1,2,6,10).

⁸<https://git-scm.com/book/en/v1/Getting-Started-About-Version-Control>

⁹<https://github.com>

¹⁰<https://peerj.com/preprints/3159/>

0.4 Keywords

- R
- Rmarkdown
- communication
- reproducibility
- git and github

0.5 Schedule

- 10:00-10:30am (30 minutes) Welcome and warm up
- 10:30-12:00pm (1.5 hours) Rmarkdown
- 12:00-1:00pm (1 hour) lunch
- 1:00-3:00pm (2 hours) More Rmarkdown
- 3:00-3:15pm (15 minutes) Tea break
- 3:15-4:45pm (1.5 hours) Even more Rmarkdown
- 4:45-5:00pm (15 minutes) Warm down

0.6 Authors and copyright

This course is developed for the Monash Bioinformatics Platform by:

- Paul Harrison¹¹
- Adele Barugahare¹²
- Kirill Tsyganov¹³

This work is licensed under a CC BY-4: Creative Commons Attribution 4.0 International License¹⁴. The attribution is “Monash Bioinformatics Platform” if copying or modifying these notes.



¹¹<mailto:paul.harrison@monash.edu>

¹²<mailto:Adele.Barugahare@monash.edu>

¹³<mailto:kirill.tsyganov@monash.edu>

¹⁴<http://creativecommons.org/licenses/by/4.0/>

Chapter 1

Introduction to Rmarkdown

Rmarkdown has become much more than just embedding R code into a document. It enables construction of very sophisticated document types from plain text files. Rmarkdown file can become pdf documents and a static website at the same time. It can turn your analysis scattered across several different Rmarkdown documents into a single multi-paged books with cross-referencing and citations, let's call it a thesis or a paper or a course book. And with essentially no effort you can change from “rendering” your Rmarkdown into presentation slides instead of web-page, ready for a conference in little time. Rmarkdown is a natural evolution of vanilla markdown, backed and extended by the Rmrkdown ecosystem discussed shortly. Rmarkdown¹ isn't the only other flavour of markdown. There are other initiatives such as GitHub Flavored Markdown (GFM)² which mainly enhances content appears on github site and CommonMark³ that tries to unify all the different flavours of markdown syntax, but all will converge to pandoc tool.

1.1 Rmarkdown core parts

Any Rmarkdown documents is broken into three main parts:

- YAML header
- The R chunks
- markdown - plain text

Those are different parts of the document that all work together to form - render

¹<https://rmarkdown.rstudio.com/>

²<https://guides.github.com/features/mastering-markdown/>

³<http://commonmark.org/>

a final document. Each one of those parts can be customised with further options, which will cover later in the book.

1.1.1 YAML header

YAML header will always seat at the very top of your Rmarkdown document and it starts and ends with triple dash symbols, ---. Note that YAML is indentation and space sensitive, meaning you need to be rather strict about amount of indentation you use and text strings will need to be quoted.

```
---
title: "Hello world"
author: "Kirill"
date: "17 June 2019"
output: html_document
---
```

1.1.2 The R chunks

These are special parts of the document that hold code that can be executed inline of the Rmarkdown document. R chunks are highly customisable via chunk options. We will spending a lot of time in the course working with code chunks and different options types.

```
```{r}
plot(mtcars)
```
```

1.1.3 Everything else is plain old markdown

```
# Have I been Marked Down?
```

1.2 Challenge: Introduction 1

5 minutes

1. What file extension should we typically use for saving our **R**markdown files?
answer link⁴
2. What document types can be produced (compiled) from Rmarkdown?

⁴<https://superuser.com/questions/249436/file-extension-for-markdown-files>

3. Will I have to learn more “languages” to use Rmarkdown (discussion question)?

The short answer is no. Learning and writing Rmarkdown will take you a very long way.

The longer answer is yes. At some point in the future you might want to very sophisticated documents and for that you’ll most certainly will need at least tiny amount of html + css knowledge and maybe some knowledge about LaTeX (I’ve yet to learn a single thing about LaTeX - so far so good :D)

check out this bit of Rmarkdown⁵

1.3 Rmarkdown ecosystem

Rmarkdown has relatively complicated ecosystem. It includes several different R packages. Most of those packages wrap other existing tools, written by different people, thereby providing an “easy” way to interface with the tools via R language. A large part of the ecosystem exists thanks to pandoc⁶ tool.

1.3.1 Pandoc

Pandoc is a stand alone tool (command line tool) that one can run in the terminal to convert markdown documents to other documents types including html, pdf and MS docs. Since vanilla markdown is pretty simple in what it can produce, pandoc added whole lot of “features”, additional marking tags, that one can use to build more elaborate documents from plain text.

1.3.2 YAML

This is stand along language that is used in variety of places, with main advantage to it is that it can be easily ready by humans as well easily parsed by computer. A lot of the time YAML can be used ad a configuration file. This is example how it is used with Rmarkdown. We will talk about YAML in more depth in a different section. In brief we will use YAML to set documents appearance and link additional files with the documents, such as bibliographies.

1.3.3 LaTeX

~_(\)_/-

⁵[link%20to%20github%20that%20the%20line%20of%20code%20above](#)

⁶<https://pandoc.org/>

1.3.4 Knitr

As was mentioned before we are using `pandoc`⁷ to convert markdown to other document types. `knitr`⁸ provides function to convert Rmarkdown files into vanilla markdown, which then in turn can be converted by `pandoc` into html document for example. Some of the things that `knitr`⁹ does includes R code execution and assembling the results into markdown.

1.3.5 Rmarkdown

An `rmarkdown` R package¹⁰ will convert `.Rmd` files into other format types. Under the hood it will use `pandoc`¹¹ to do so. The main function that we are concerned with is `rmarkdown::render()` which will call `knitr::knit()` when required.

1.3.6 Bookdown

A `bookdown` R package¹² enhances `rmarkdown`¹³ by enabling multi-page documents e.g books and easy cross-referencing.

1.3.7 Others

These are more R packages that enable more things via Rmarkdown.

- `xaringan`¹⁴
- `blogdown`¹⁵
- `thesisdown`¹⁶

1.4 Setup

We will need to install these packages. Let's install these packages

⁷<https://pandoc.org/>

⁸<https://yihui.name/knitr/>

⁹<https://yihui.name/knitr/>

¹⁰<https://github.com/rstudio/rmarkdown>

¹¹<https://pandoc.org/>

¹²<https://github.com/rstudio/bookdown>

¹³<https://github.com/rstudio/rmarkdown>

¹⁴<https://github.com/yihui/xaringan>

¹⁵<https://github.com/rstudio/blogdown>

¹⁶<https://github.com/ismayc/thesisdown>

```
packages <- c("tidyverse",  
             "rmarkdown",  
             "knitr",  
             "bookdown",  
             "tinytex",  
             "citr")  
  
keep <- packages[!(packages %in% installed.packages()[,"Package"])]  
  
if(length(keep)) {  
  install.packages(keep)  
}
```

1.5 Challenge: Introduction 2

2 minutes

1. Now is a good time to tweak your RStudio to your needs.
 - change font size
 - change themes and background color
 - rearrange panels
2. Please turn off “Restore .RData into workspace at startup” in “Tools -> Global Options”.

1.6 Useful tips

- don’t attempt to compile to `pdf_document` until absolutely necessary. `LaTeX` engine that is used by Rmarkdown to pdf conversion known to have issues with aligning figures and tables. This typically causes figures and tables overflow to next pages and general text misalignment. Get your content written first, intermediate compilation to `html_documents` are totally fine, before worrying about technical issues
- don’t save data into `.RData`, this will make your work less reproducible

Chapter 2

Vanilla Markdown

The original (vanilla) version of Markdown invented by John Gruber¹ defines a handful of tags, discussed next. Markdown is relatively small and simple language for writing plain text documents that are easy-to-write and easy-to-read, but it is greatly enhanced and extended by pandoc tool.

2.1 Vanilla tags

Let's open our first Markdown file.

File

New File

R Markdown

```
title = "Learning Rmarkdown"
```

```
author = "Me"
```

- select document type **HTML**
- to build (compile) the document press **knitr** button or **ctrl+alt+k**
- to save as **.Rmd** file

Since we are using RStudio and R it is inevitable that will be using Rmarkdown flavour, but we can still write only vanilla markdown and remember that under the hood Rmarkdown will always be converted to vanilla markdown.

From now onward we are going to start using **knitr** to compile Rmarkdown into html. Remember from the Rmarkdown ecosystem² that **knitr** will convert Rmarkdown to markdown and **rmarkdown** R package will convert - render

¹<https://en.wikipedia.org/wiki/Markdown>

²

markdown file into html. By pressing that blue button both things will happen automatically and we don't need to think about, but I wanted you to know that.

These are essentially all core (vanilla) markdown tags. Let's practice writing them.

```
# Header1
## Header2
### Header3

Paragraphs are separated
by a blank line.

Two spaces at the end of a line
produces a line break.

Text attributes _italic_,
**bold**, `monospace`.

Horizontal rule:

---

Bullet list:

* apples
* oranges
* pears

Numbered list:

1. wash
2. rinse
3. repeat

A [link](http://example.com).

![Image](Image_icon.png)

> Markdown uses email-style > characters for blockquoting.
```

2.2 Practice vanilla markdown

Now it's just a matter of learning some of the markdown syntax. Let's delete all current text from the opened document except the YAML header and type

this new text in `Hello world, I'm learning R markdown !` and pressing the `knit HTML` button.

```
Hello world, I'm learning R markdown !
```

Not much happened. This is because we didn't mark our text in any way. You can put as much text as you want and it will appear as is, unless "specially" marked to look differently.

Now add the `#` symbol at the start of the line and press the `knit HTML` button again. We'll be pressing this button a lot! For those who like keyboard short cuts use `ctrl+shift+k` instead.

```
# Hello world, I'm learning R markdown !
```

How about now? A single hash symbol made it whole lot bigger didn't it? We've marked this whole line to be the header line.

Now make three new lines with the same text, but different numbers of `#` symbols, one, two and three respectively and keep pressing the `Knit HTML` button

```
# Hello world, I'm learning R markdown !
## Hello world, I'm learning R markdown !
### Hello world, I'm learning R markdown !
```

This is how you can specify different headers type using markdown.

Remember that vanilla markdown³ is comprised entirely of punctuation characters.

2.3 Challenge: Markdown 101

5 minutes

1. How to mark text so that it appears underlined? answer link⁴
2. Can markdown replace html⁵ (discussion question)? It has replaced html and latex in documentation and communication of results. My feeling is that data science ecosystem heavily rotates around markdown. But html, pdf and latex in this context are simply communication and sharing medium. One would not want to replace html + css for large website project

³<https://daringfireball.net/projects/markdown/syntax>

⁴<https://softwareengineering.stackexchange.com/questions/207727/why-there-is-no-markdown-for-underline>

⁵<https://en.wikipedia.org/wiki/HTML>

```
library(tidyverse)
```

```
## -- Attaching packages ----- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.1      v purrr   0.3.2
```

```
## v tibble  2.1.2      v dplyr  0.8.1
```

```
## v tidyr   0.8.2      v stringr 1.4.0
```

```
## v readr   1.3.1      v forcats 0.3.0
```

```
## -- Conflicts ----- tidyverse_conflicts() --
```

```
## x dplyr::filter() masks stats::filter()
```

```
## x dplyr::lag()    masks stats::lag()
```

```
library(knitr)
```

Chapter 3

Rmarkdown

The reason that we are learning Rmarkdown¹ is because it gives us very straight forward way of writing plain text documents with inline R code that will become a very sophisticated document types. The bonus points also come from the fact that Rmarkdown files are easy to version control (git) and see the difference between versions. This approach of interleaving analysis code, commentary and description is very explicit, which has direct implication in reproducibility, shearability and collaboration.

In Markdown section I've showed you how to start new Rmarkdown document in RStudio, but lets briefly recap how we do that again.

File

New File

R Markdown

```
title = "Learning Rmarkdown"
```

```
author = "Me"
```

- select document type **HTML**
- to build (compile) the document press **knitr** button or **ctrl+alt+k**
- to save as **.Rmd** file

3.1 Embedding R code

RStudio templates **.Rmd** file for us. However lets delete all the text after the **yml** header.

¹<https://rmarkdown.rstudio.com/>

```

---
title: 'Learning Rmarkdown'
author: 'Kirill'
date: '21/06/2019'
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

```

I'm going to explain `knitr::opts_chunk` in later section. An R chunk is a “special” block with in the document that will be read and evaluated by `knitr`, ultimately converting everything into plain markdown. But for us it means that we can focus on our analysis and embed R code without having to worry about it. Additionally there are large number of chunk options that helps with different aspects of the document including code decoration and evaluation, results and plots rendering and display.

This is an R code chunk.

```

```{r}

```

```

The little `r` there specifies the “engine”, basically telling Rmarkdown how to evaluate the code inside the chunk. Here we are saying use R engine (language) to evaluate the code. The list of languages² is rather long, hence why earlier comments about Rmarkdown spanning much greater area than one might think. In this workshop we are only going to focus on R language.

Let's write our first bit of R code inside the Rmarkdown document. First we need to start a new R chunk, which can be done in either of three ways:

- simply type it out
- press insert button at the top of the window
- `ctrl+alt+i`

Let's start with a simple `print()` statement and print `Hello world, I'm learning Rmarkdown !` string, except we are going to split it between two variable

```

```{r}
a <- 'Hello world, Im learning Rmarkdown !'
a
```

```

²<https://bookdown.org/yihui/rmarkdown/language-engines.html>

Note that each chunk can be run independently in the console by pressing `ctrl+enter` or little green arrow.

Each code chunk is highly customisable via chunk options³. We are going to learn a few today, but we won't be able to cover all of them. You probably never going to use some of them, but as long as you know what to look for you'll be able to search for then. Note that all chunk options have a default value. Not specifying an options means you are using the default value. These are chunk options that we are going to cover today.

| name | value | type | description |
|------------|-----------|-----------------|---|
| child | NULL | code_evaluation | A character vector of filenames. Knitr will knit the files and place them |
| engine | 'R' | code_evaluation | Knitr will evaluate the chunk in the named language, e.g. engine = 'py' |
| eval | TRUE | code_evaluation | If FALSE, knitr will not run the code in the code chunk. |
| include | TRUE | code_evaluation | If FALSE, knitr will run the chunk but not include the chunk in the fin |
| fig.align | 'default' | plots | How to align graphics in the final document. One of 'left', 'right', or 'ce |
| fig.cap | NULL | plots | A character string to be used as a figure caption in LaTeX. |
| fig.height | 7 | plots | The height to use in R for plots created by the chunk (in inches). |
| fig.width | 7 | plots | The width to use in R for plots created by the chunk (in inches). |
| echo | TRUE | results | If FALSE, knitr will not display the code in the code chunk above it's r |
| results | 'markup' | results | If 'hide', knitr will not display the code's results in the final document. |
| message | TRUE | results | If FALSE, knitr will not display any messages generated by the code. |
| warning | TRUE | results | If FALSE, knitr will not display any warning messages generated by the |

General layout of any chunk is

```
```{r chunk_name, options}
...
```
```

Note a couple of things, there isn't a comma between `r` and `chunk_name`. Not sure why this is.. Also note that `chunk_name` is optional, you can skip it, as we have in earlier examples. Naming chunks is good idea to conceptually label the chunk as to what it does, but also we you are going to build more sophisticated documents you'll be able to selectively include chunks by refer to them by the chunk name.

Lets start off with these four chunk options:

- `echo` show what has been typed in i.e show the code
- `eval` evaluate or execute that code
- `include` include execute code into the document, relies on `eval = TRUE`
- `results` hide resulting output

These allow us fine level control over the final document. Think about who are generating the document for and what type of information you need to share.

³<https://bookdown.org/yihui/rmarkdown/r-code.html>

Sometimes we might want to show the code, but not execute it and other times we might just want to execute it and share the results, e.g plot, without actually showing the code.

3.1.1 echo and eval

Let's start with `echo = TRUE` and `eval = TRUE`.

```
```{r echo = T, eval = T}
a <- 'Hello world,'
b <- 'Im learning Rmarkdown !'
ab <- paste(a, b)
print(ab)
```
```

```
a <- 'Hello world,'
b <- 'Im learning Rmarkdown !'
ab <- paste(a, b)
print(ab)
```

```
## [1] "Hello world, Im learning Rmarkdown !"
```

Now let's turn echo off, `echo=FALSE`.

```
```{r echo = F}
a <- 'Hello world,'
b <- 'Im learning Rmarkdown !'
ab <- paste(a, b)
print(ab)
```
```

```
## [1] "Hello world, Im learning Rmarkdown !"
```

Okay, we don't see our original `print()` statement. And now let's pass `eval=FALSE` options instead

```
```{r eval = F}
a <- 'Hello world,'
b <- 'Im learning Rmarkdown !'
ab <- paste(a, b)
print(ab)
```
```



```
a <- 'Hello world,'
b <- 'Im learning Rmarkdown !'
ab <- paste(a, b)
print(ab)
```

3.1.2 include

This option also helps to manipulate your final document look. This option dictates whether the output of the executed code will be included into the final document. Sometimes you can simply trigger `eval` flag to achieve similar result of code now being included, but other times you might want the code to actually be executed but not included. For example when future R chunk relies on the output of this intermediate chunk, but there is now need to include that into the document.

```
```{r echo = T, eval = T, include = F}
a <- 'Hello world,'
b <- 'Im learning Rmarkdown !'
ab <- paste(a, b)
print(ab)
```
```

```
```{r, echo = T, include = T}
ab
```
```

```
a <- 'Hello world,'
b <- 'Im learning Rmarkdown !'
ab <- paste(a, b)
print(ab)
```

```
ab
```

```
## [1] "Hello world, Im learning Rmarkdown !"
```

3.1.3 results

Now we see the code, but not the output. The difference between `echo` and `results` is subtle, at least in my head. Let's consider the following example.

```
```{r echo = T, eval = F, results = 'asis'}
a <- 'Hello world,'
```

```
b <- 'Im learning Rmarkdown !'
ab <- paste(a, b)
print(ab)
```

```

```
```{r}
ab
```

```

```
a <- 'Hello world,'
b <- 'Im learning Rmarkdown !'
ab <- paste(a, b)
print(ab)
```

```
ab
```

```
## [1] "Hello world, Im learning Rmarkdown !"
```

Let's turn results = 'hide'

```
```{r echo = T, eval = F, results = 'hide'}
a <- 'Hello world,'
b <- 'Im learning Rmarkdown !'
ab <- paste(a, b)
print(ab)
```
```{r}
ab
```

```

```
a <- 'Hello world,'
b <- 'Im learning Rmarkdown !'
ab <- paste(a, b)
print(ab)
```

```
[1] "Hello world, Im learning Rmarkdown !"
```

```
ab
```

```
## [1] "Hello world, Im learning Rmarkdown !"
```

And now we only see `print()` statement and no output.

3.2 Challenge: code chunks

3 minutes

1. Go through all of your code so far and give each chunk a name

```
```\n{r chunk_name, options}\n```
```



## Chapter 4

# Git and GitHub introduction

When you are rock climbing you want to set your anchors often  
How often will depend on your experience and desire not to fall  
Git commit like you are vertically hanging off 70 feet rock

I going to break to you right at the start that (unfortunately) doing git and GitHub is like rock climbing, but nonetheless it has great benefits for your research and analysis including making it more visible, reproducible and potentially collaborative.

Git<sup>1</sup> is one of many tool, but very popular, that was design for **tracking versions** of software development - a.k.a version control tool. While it hasn't been strickly design with scientific research projects in mind we will happily repurpose git to help us stay on top of our research projects. In git world everything rotates around git repository, which is "special" folder. Inside that folder every file and folder is "tracked" for changes. Git repositories often are synonymous with project folder. It is a "bucket" or "container" that holds everything related to a particular project.

GitHub on the another hand is a place where people can deposit, store and share their git repositories. GitHub also one of many different places that people can choose to use to store and share their git repositories. Image below illustrates some of other common place one can choose to store they git repositories a.k.a projects

For this workshop we are going to go with GitHub, mainly it is very popular and it has a lot of useful features, some of which I'll share with you later in the workshop.

---

<sup>1</sup><https://git-scm.com/doc>

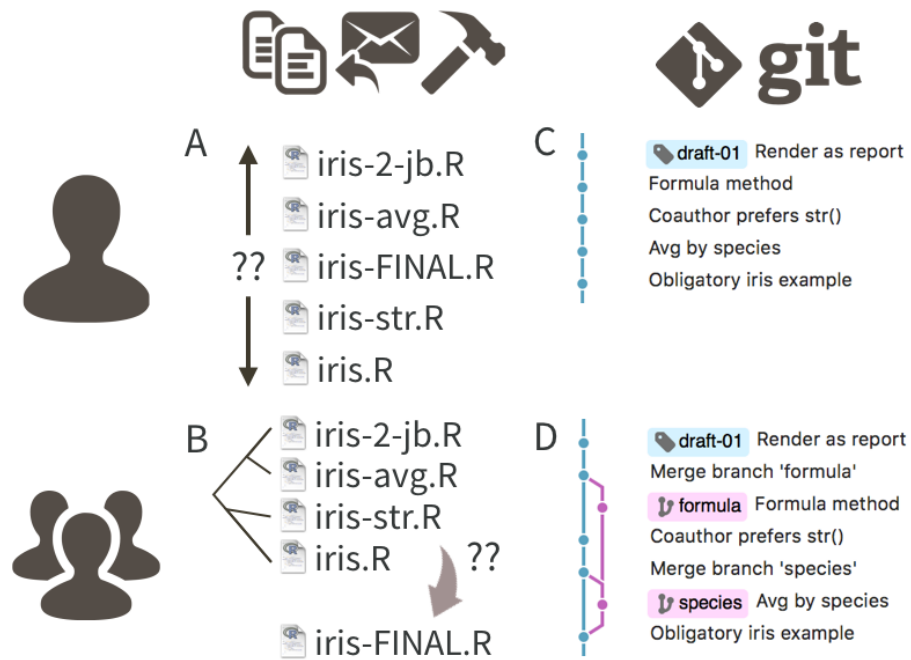


Figure 4.1: This is an example of git version control vs DIY versioning via filesystem



Figure 4.2: <https://www.geekboots.com/story/what-is-the-difference-between-bitbucket-github-and-gitlab>

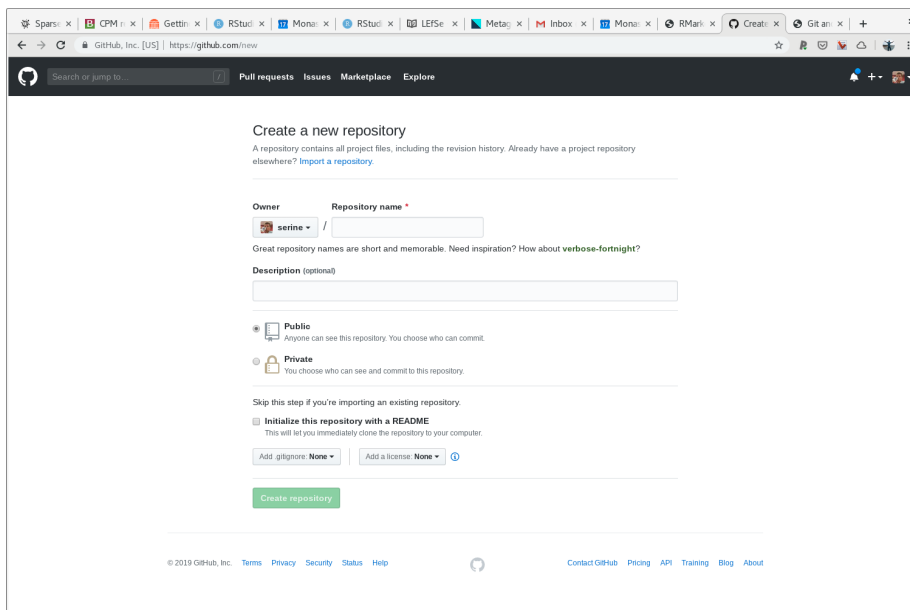
Git and GitHub will help you:

- organise our directory structure
- create “milestones” a.k.a `git commits`
- make apparent which parts of the projects (files) are important
- share your work (e.g GitHub)
- collaborate at the global scale

## 4.1 Github setup

There are a couple of different ways you start a project and initiate git repository - git tracking. We are going to start with GitHub first approach. An alternative approach discussed in appendix section D. I hope that everyone had already created GitHub<sup>2</sup> account. We are going to create a new github repository, which is also automatically a git repository. Remember that GitHub is just a place where we are storing out git repositories and you can only store git repositories on GitHub.

Let's go to GitHub<sup>3</sup> and create new repository and let's name it “learning\_rmarkdown”.



The screenshot shows the GitHub web interface for creating a new repository. The browser's address bar shows the URL <https://github.com/new>. The page title is "Create a new repository". Below the title, there is a sub-header "A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#)". The form includes a "Repository name" field with the text "serine" entered. Below this is a "Description (optional)" text area. There are two radio buttons for visibility: "Public" (selected) and "Private". Below these are checkboxes for "Initialize this repository with a README" and "Add a license". At the bottom of the form is a green "Create repository" button. The footer of the page shows copyright information for 2019 GitHub, Inc. and links to Terms, Privacy, Security, Status, and Help.

<sup>2</sup><https://github.com/>

<sup>3</sup><https://github.com/>

Description of the repository is optional, but a good idea to write a brief sentence there to message to yourself and the public what this project is about. Let's write the following brief sentence; "I'm learning Rmarkdown". Also note that I ticked "Initialize this repository with a README". This completely optional.

The screenshot shows the GitHub 'Create a new repository' page. The browser's address bar shows 'https://github.com/new'. The page has a dark header with navigation links: 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. The main content area is titled 'Create a new repository' and includes a subtext: 'A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)'

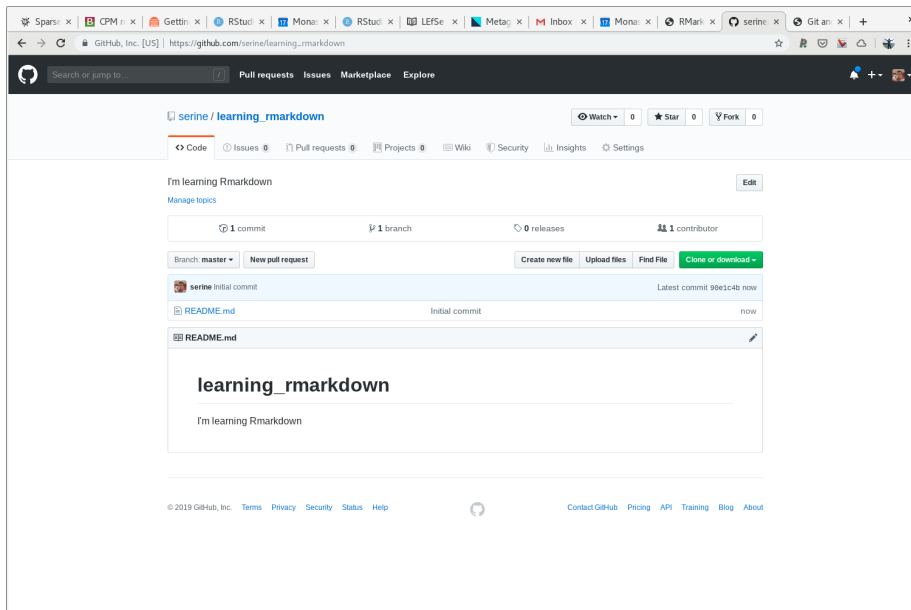
The form fields are as follows:

- Owner:** serine (with a dropdown arrow)
- Repository name:** learning\_markdown (with a green checkmark)
- Description (optional):** I'm learning Rmarkdown
- Visibility:** Public (selected), Private (unselected)
- Initialize this repository with a README:** Checked (indicated by a green checkmark)
- Add a license:** None (selected)

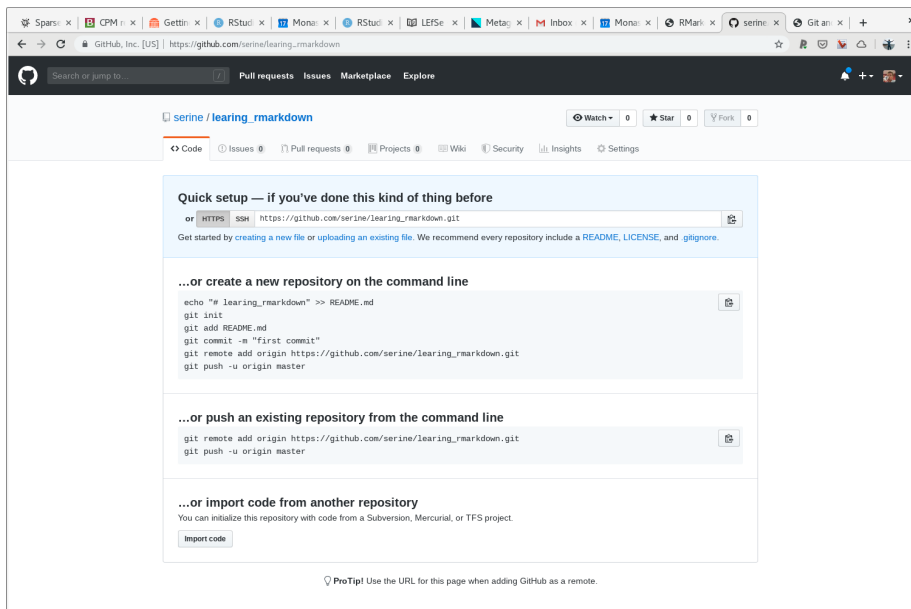
At the bottom of the form is a green 'Create repository' button. The footer of the page includes copyright information '© 2019 GitHub, Inc.' and links to 'Terms', 'Privacy', 'Security', 'Status', 'Help', 'Contact GitHub', 'Pricing', 'API', 'Training', 'Blog', and 'About'.

This is the screen you are going to get when you've initialised it with a README file





And this is a screen you are going to get if you haven't initialised it with a README file



A brief note about README files. It is regarded as a “silent” way of communication, where you can tell all necessary information another person need to

know about your project. For a software tool you would put information about how to build that particular tool and dependencies. In our case we will add information how to build final html report. We will do this a bit later in the workshop.

Either way is completely fine! Once we have our GitHub repository we need to find a link (URL) and copy a.k.a **clone** (git clone) our repository to our working computer, in our case rstudio.cloud<sup>4</sup>. We want to establish connection between rstudio.cloud<sup>5</sup> and GitHub such that we can with little effort we can copy our work, that is file from rstudio.cloud<sup>6</sup> to GitHub.

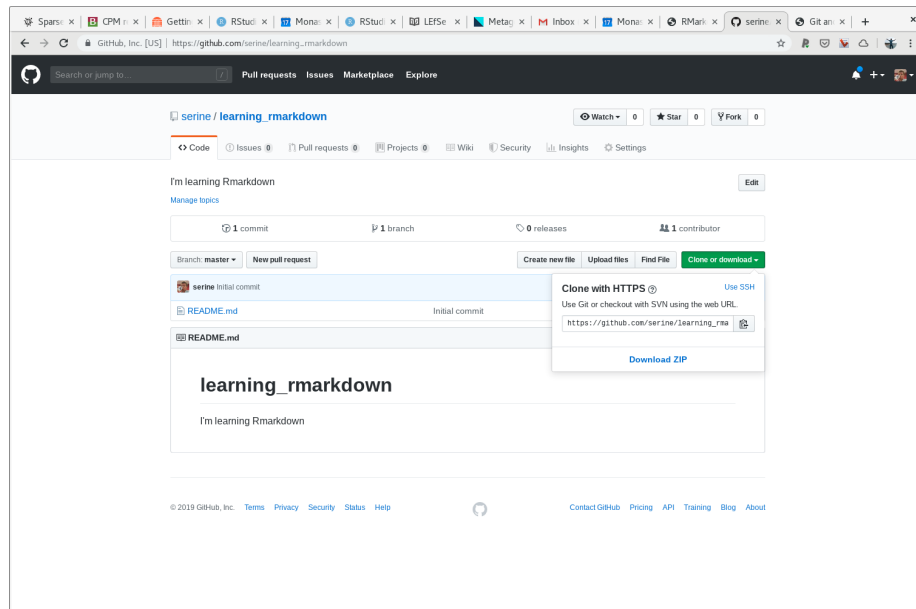
This is how typical URL looks like

`https://github.com/serine/learing_rmarkdown.git`

There are at least three components in that url

- `https://github.com/` the place i.e the name of the website
- `serine/` username
- `learing_rmarkdown.git` repository name (project name)

You can get this url, but either looking at the address bar of your browser or there is a little drop down menu on the right hand site.



<sup>4</sup>`https://rstudio.cloud`

<sup>5</sup>`https://rstudio.cloud`

<sup>6</sup>`https://rstudio.cloud`

## 4.2 RStudio setup

Add description about starting a project from a github at RStudio.

## 4.3 GitHub things

- PR (pull request)
- gitissues place to talk about issues related to a project
- stars acknowledgement
- watch interested in updates on a projects

collaborators and update dates/commits as a proxy of how active the project is. also do check which files typically being changed. Also mention the fact that it is very explicit when the project was started (initiated) how much work has gone into it (commits history) and roughly time frame and intervals of work

in simple workflow and collaborations git merge will work just fine. git will happily merge two different branches i.e all files in one location with all files in the other location if no two file conflict

## 4.4 More git

Git is a command line tool however you don't have to learn command line just yet. There are a few git clients available<sup>7</sup> - graphical user interface (GUI) tool / applications that we can use instead of learning command line. We are going to use RStudio which has good git support and therefore Rstudio will be our git client. One rather important note about git clients, most (all) clients will "simply" form a git commands as you would type it out and execute on command line. This means a couple of things:

1. one can use mixture of clients and command line without any issues. For example if one needs more complicated git command one could run it on the command-line.
2. if you need to do a more complicated git kung fu you might only find solution for command line and then it'll be up to you to figure out how to work it into your client

An interesting note about command line git usage noted in Happy Git with R book<sup>8</sup>; One might think that git via cli is "better", however it is more important to get the work done and have it version controlled rather than fight with the

---

<sup>7</sup><https://happygitwithr.com/git-client.html>

<sup>8</sup><https://happygitwithr.com/git-client.html>

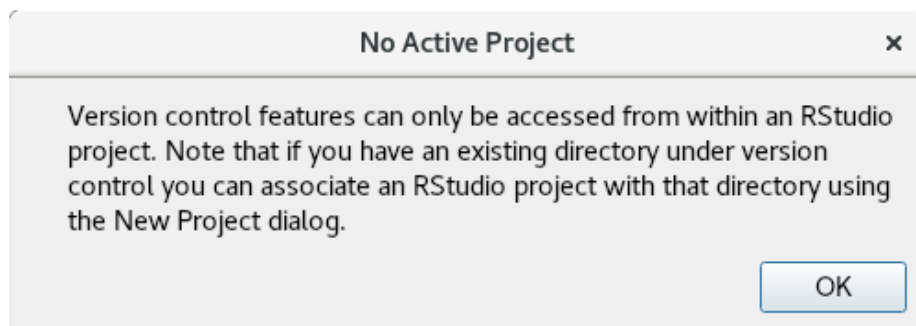
cli. Do take the simplest and quickest path to get your work version controlled. No one will care which client you are using in the end.

## 4.5 git no nos

- no spaces in file names (this goes beyond git)
- no git repositories inside an existing git repository

## 4.6 Starting with git

In Rstudio you can only start working with git when you have an existing Rproject directory



### 4.6.1 Configuring git

You will most certainly forget this step, because you only need to do it once per computer (or new installation of git). Git will remind if you haven't done these steps. These are our very first step in being organised and ready for future collaboration. We need to let git know our name and email address, which will get stored in configuration file.

Unfortunately RStudio doesn't have support for setting up config. It was probably not worth implementing given that you only really do it once. We will have to use terminal (command line) just this once.

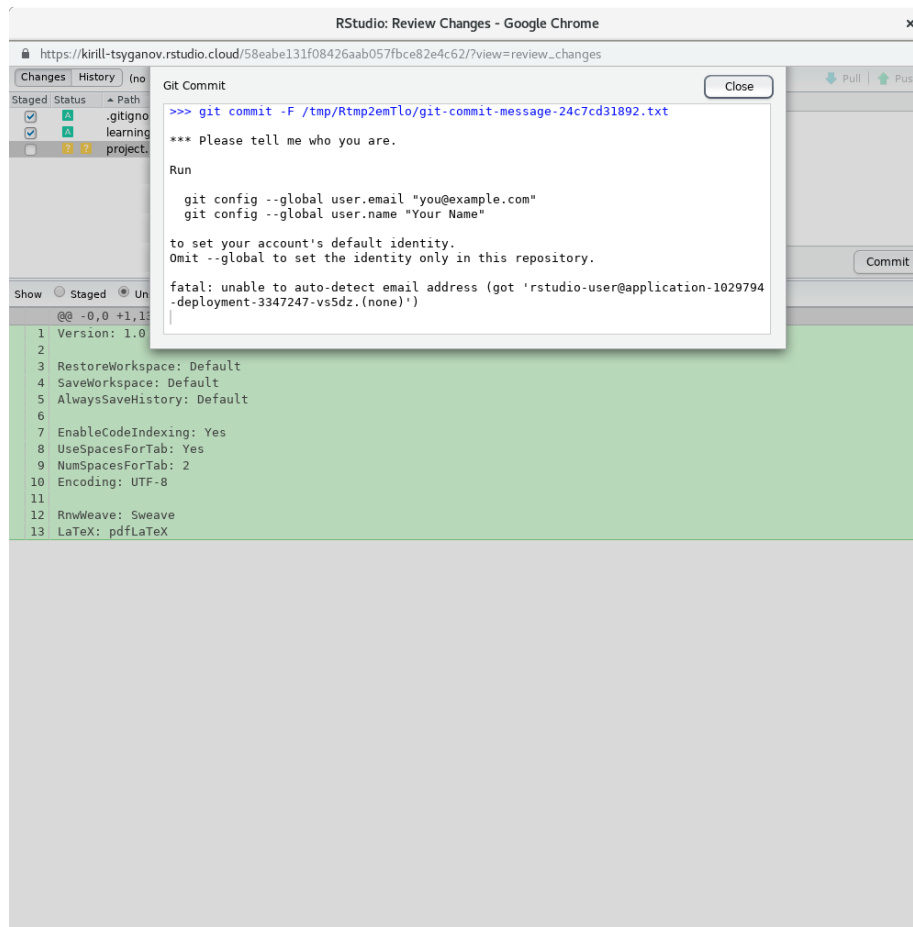
Let's open up a terminal and run a couple of `git` commands

Tools -> Terminal -> New Terminal

```
git config --global user.name kirill
git config --global user.email "kirill.tsyganov@monash.edu"
```

One can then double check that all was set correctly bu running this command.

```
git config --global --list
```



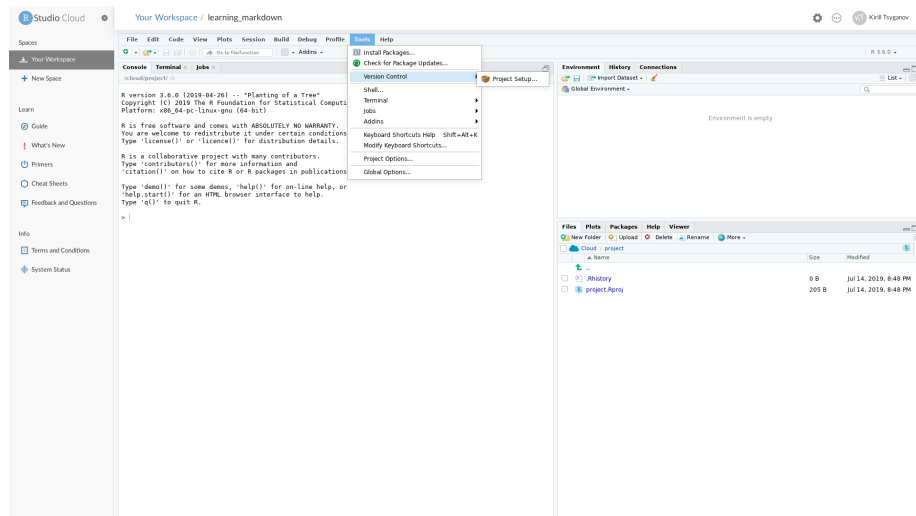
## 4.6.2 Initiating git repository

In git jargon repository is simply your working folder (folders sometimes also called directories). In our case

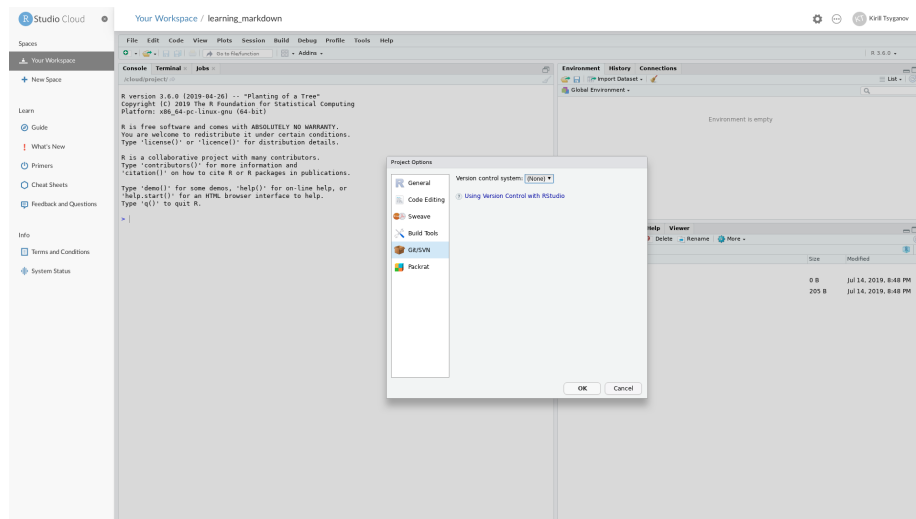
Rproject folder == Rproject directory == git repository == git repo

Let's initiate git repository

Tools -> Version Control -> Project Setup -> Git/SVN



And select from the drop down options “Version control system” Git



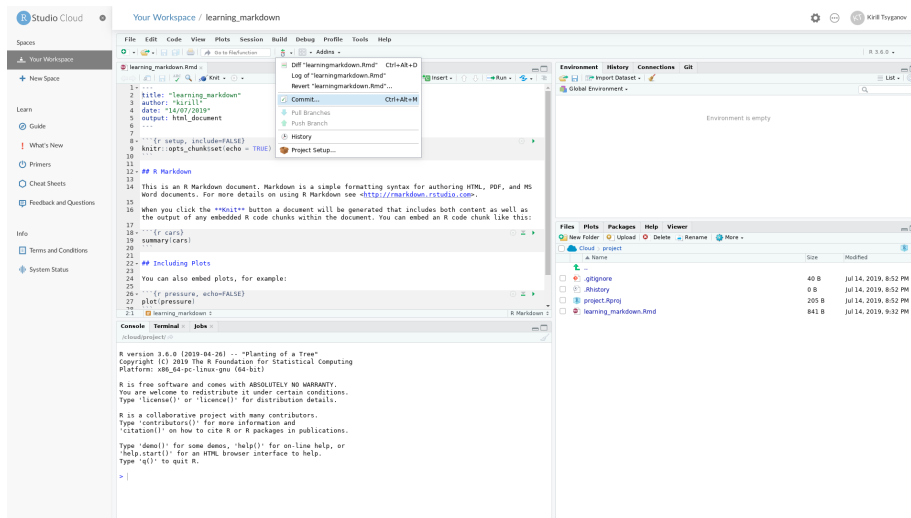
The command line equivalent is navigating to your project directory and running `git init`

### 4.6.3 First commit

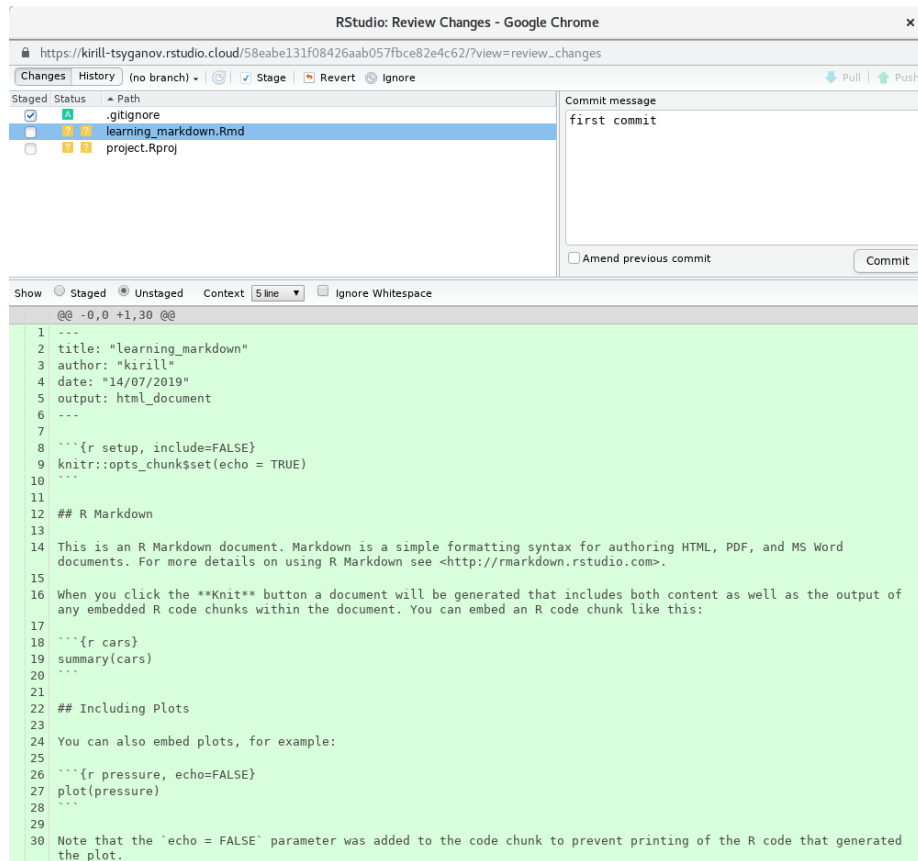
Let's make our first commit, use drop down menu as indicated on the image below to select `commit` option

## 4.6. STARTING WITH GIT

39



You should see a new window popped up

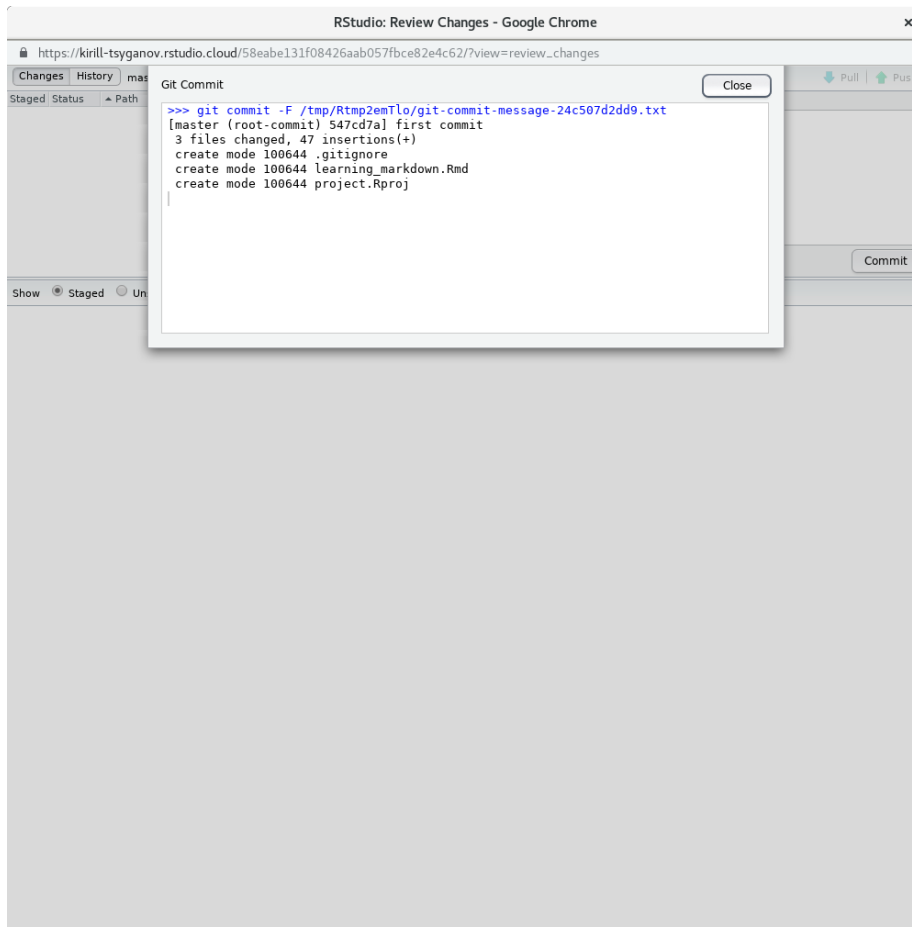


Then we are going to add three files

- `.gitignore`
- `project.Rproj`
- `learning_markdown.Rmd`

Write a commit message and press commit. And this is how happy git commit looks like





The commit message is rather important. Remember that commit message is:

- a message to a future you
- a message to your supervisor
- a message to all other external people

Those commit messages are means of communications e.g

- “fixed figure 1 legend”
- “added new paragraph to chapter 1”
- “I bloody hate this project delete everything, starting from scratch”

Good thing is, as long as you “tracking” your deletes you can always go back to them and check what you have deleted and revert some of those changes back when needed. However in this workshop we won’t be covering much of that.

Also note that commit message don't have to long, and can be as short as one work - "update2", but at the same time well written commit message will help you and other.

<http://r-pkgs.had.co.nz/git.html#commit-best-practices>

## 4.7 Which files to commit?

This section will be extended in the future release, but I highly recommend reading this article, specifically section 10: Which files to commit from here<sup>9</sup>

---

<sup>9</sup><https://peerj.com/preprints/3159/>

## Chapter 5

# The R chunk 2

### 5.1 Working with code chunks

Let's continue our exploration of chunk options and now try a different example using `mtcars` dataset and learn a few more chunk options.

```
```{r}
summary(mtcars)
```
```

```
summary(mtcars)
```

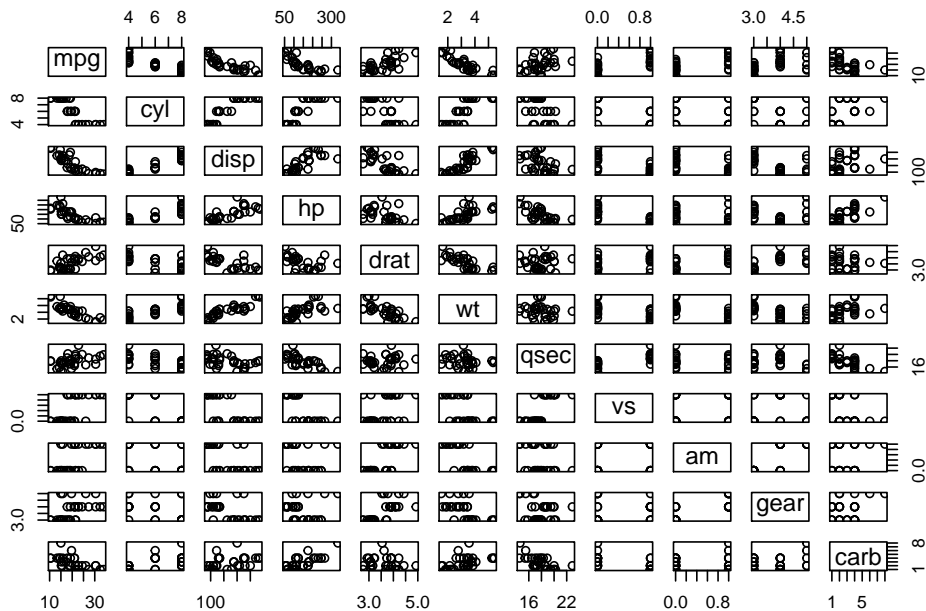
```
mpg cyl disp hp
Min. :10.40 Min. :4.000 Min. : 71.1 Min. : 52.0
1st Qu.:15.43 1st Qu.:4.000 1st Qu.:120.8 1st Qu.: 96.5
Median :19.20 Median :6.000 Median :196.3 Median :123.0
Mean :20.09 Mean :6.188 Mean :230.7 Mean :146.7
3rd Qu.:22.80 3rd Qu.:8.000 3rd Qu.:326.0 3rd Qu.:180.0
Max. :33.90 Max. :8.000 Max. :472.0 Max. :335.0
drat wt qsec vs
Min. :2.760 Min. :1.513 Min. :14.50 Min. :0.0000
1st Qu.:3.080 1st Qu.:2.581 1st Qu.:16.89 1st Qu.:0.0000
Median :3.695 Median :3.325 Median :17.71 Median :0.0000
Mean :3.597 Mean :3.217 Mean :17.85 Mean :0.4375
3rd Qu.:3.920 3rd Qu.:3.610 3rd Qu.:18.90 3rd Qu.:1.0000
Max. :4.930 Max. :5.424 Max. :22.90 Max. :1.0000
am gear carb
Min. :0.0000 Min. :3.000 Min. :1.000
1st Qu.:0.0000 1st Qu.:3.000 1st Qu.:2.000
```

```
Median :0.0000 Median :4.000 Median :2.000
Mean :0.4062 Mean :3.688 Mean :2.812
3rd Qu.:1.0000 3rd Qu.:4.000 3rd Qu.:4.000
Max. :1.0000 Max. :5.000 Max. :8.000
```

**Remember** You can go between Rmarkdown and *console*, to check your code, at any time. You should see your code block is highlighted differently and you should see a green arrow at the right hand site of that block. Press the green arrow to get an output in the *console*. You can also use **ctrl+enter** to do the same with the keyboard short cut.

```
```{r}
plot(mtcars)
```
```

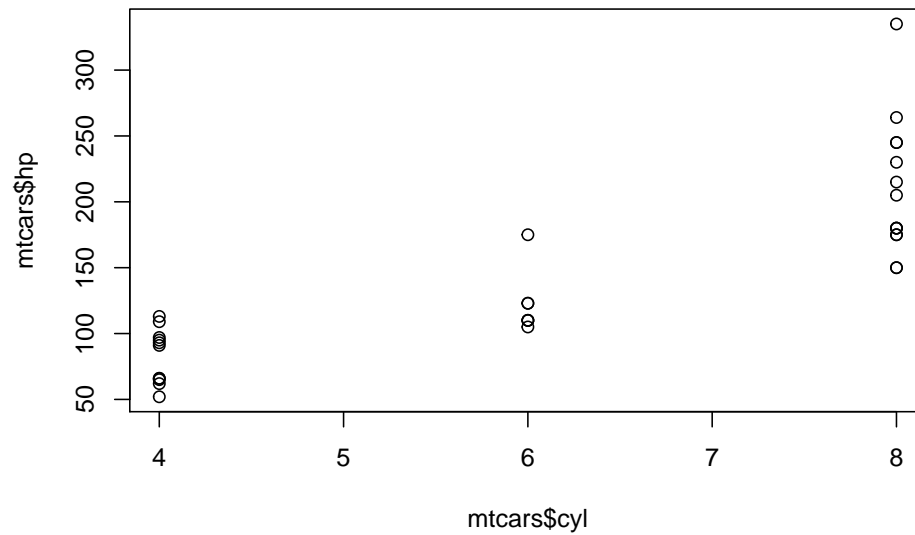
```
plot(mtcars)
```



This is great, but a bit too much information, lets just focus on number of cylinders and hourse power.

```
```{r}
plot(mtcars$cyl, mtcars$hp)
```
```

```
plot(mtcars$cyl, mtcars$hp)
```

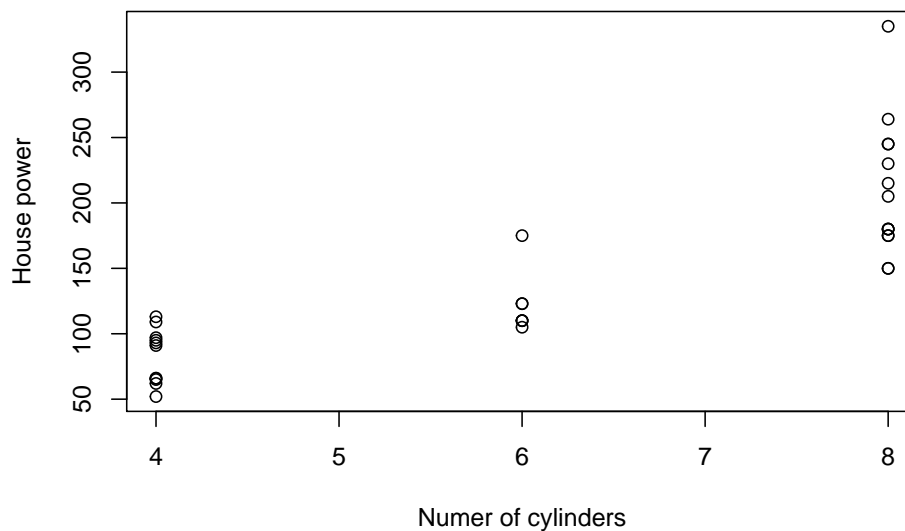


We can add a bit more information to our plot, to make more self descriptive.

```
```{r}
plot(mtcars$cyl,
      mtcars$hp,
      main='House power vs number of cylinders',
      xlab = 'Numer of cylinders',
      ylab='House power')
```
```

```
plot(mtcars$cyl,
 mtcars$hp,
 main='House power vs number of cylinders',
 xlab = 'Numer of cylinders',
 ylab='House power')
```

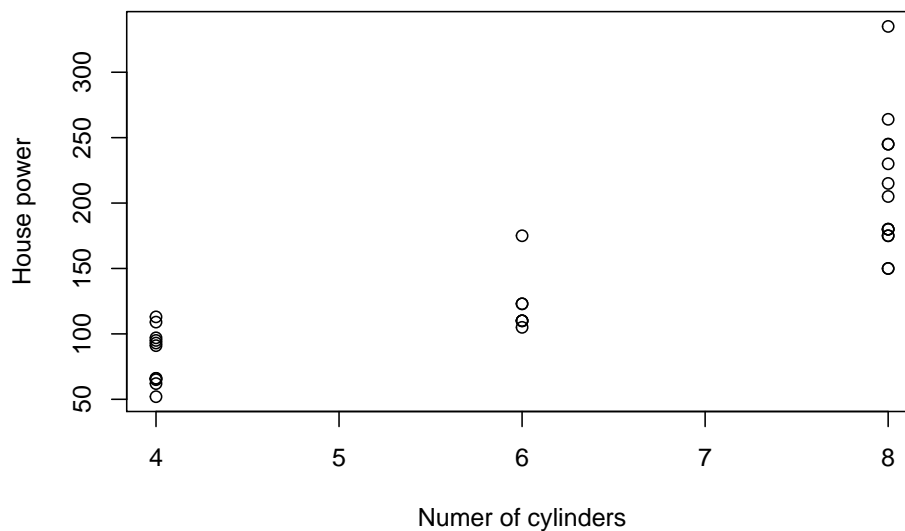
### House power vs number of cylinders



Here is a good example where we can hide our code from the viewer, since it isn't most interesting bit about this data. Let's turn `echo=FALSE` options for all our plots below.

Properly labelled plots are very informative, let's do that as well, starting with a title `main="Travelling speed vs Breaking distance"` and then labelling axis, `x xlab="Travelling speed (mhp)"` and `y ylab="Stopping distance (ft)"`

### House power vs number of cylinders



We are no longer seeing the code, rather just the figure. You can try `eval = FALSE` by yourself to see what happens.

## 5.2 Figures related chunk options

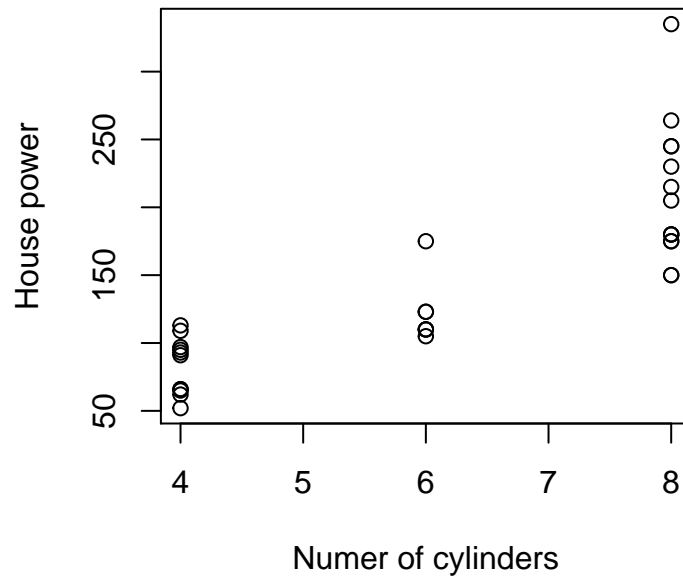
- `fig.align` - left, right, center or default (left)
- `fig.height` - height specified in inches
- `fig.width` - width specified in inches
- `fig.cap` - string of text in quotes

Let me show you a how to resize the plot with `fig.height` and `fig.width` and then we are going to do a challenge.

```
```{r fig.height = 4, fig.width = 4}
plot(mtcars$cyl,
      mtcars$hp,
      main='House power vs number of cylinders',
      xlab = 'Numer of cylinders',
      ylab='House power')
```
```

```
plot(mtcars$cyl,
 mtcars$hp,
 main='House power vs number of cylinders',
 xlab = 'Numer of cylinders',
 ylab='House power')
```

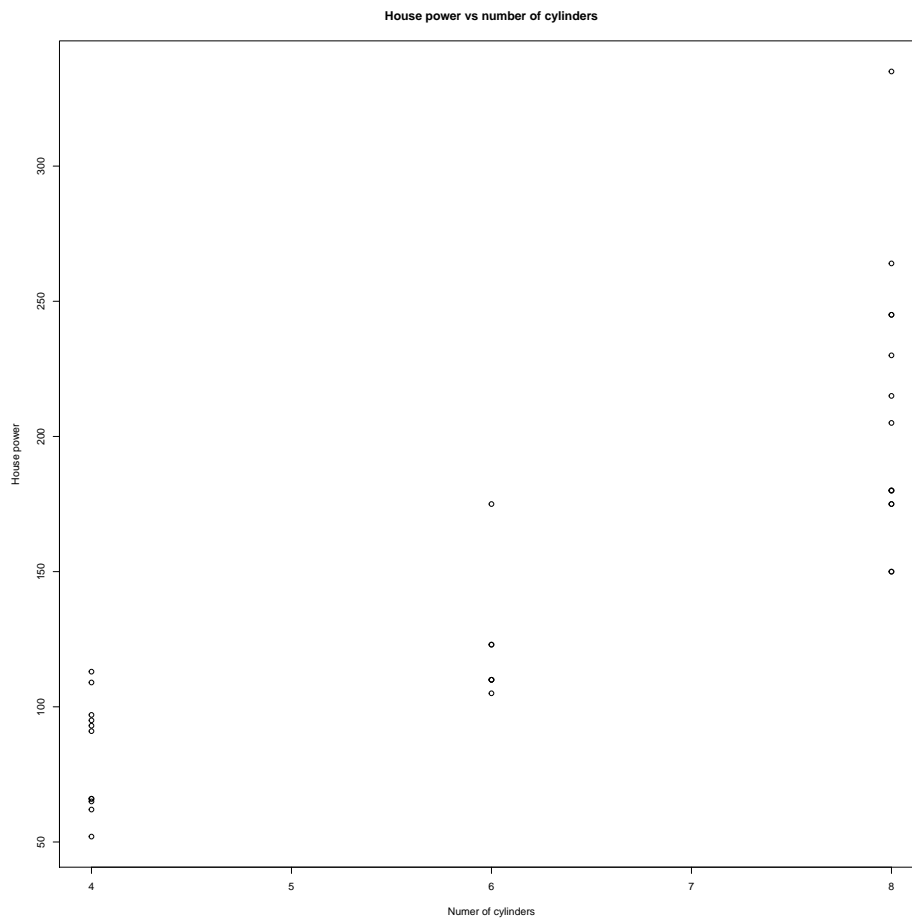
## House power vs number of cylinders



Let's try to make it very big, by trying to set height and width to 15 inches.

```
plot(mtcars$cyl,
 mtcars$hp,
 main='House power vs number of cylinders',
 xlab = 'Numer of cylinders',
 ylab='House power')
```





Note that we are starting to hit “boundaries” of the documents. (want to talk about the fact that plot sits inside a `<div>` box)

## 5.3 Challenge: code chunks

5 minutes

1. Can you align figure to the right?? `fig.align = 'right'`
2. Align figure to the center and add figure legend `fig.align = 'center', fig.cap = 'Figure 1: blah'`
3. Can you add some emphasis to figure legend, e.g make important parts bold or underlined? Remember that figure legend is just a string of text and any text can be marked

## 5.4 More useful chunk options

I'm going to share a few more useful code chunks, some are cosmetic, some you may never use, but hey can be handy in making your document visually different.

- `prompt=FALSE` i.e mimic *console*
- `comment=` remove hash symbol at the front of the output
- `child=` path to another Rmd file
- `warning=FALSE`
- `messages=FALSE`

For this example I'm going to use simple `for` loop. We are going to use this variable `sentence <- c("Let", "the", "computer", "do", "the", "work")`

```
```{r}
sentence <- c('Let', 'the', 'computer', 'do', 'the', 'work')

for(word in sentence){
  print(word)
}
```
```

```
sentence <- c("Let", "the", "computer", "do", "the", "work")

for(word in sentence){
 print(word)
}
```

```
[1] "Let"
[1] "the"
[1] "computer"
[1] "do"
[1] "the"
[1] "work"
```

Let's add `prompt=TRUE`

```
> sentence <- c("Let", "the", "computer", "do", "the", "work")
>
> for(word in sentence){
+ print(word)
+ }
```

```
[1] "Let"
[1] "the"
[1] "computer"
[1] "do"
[1] "the"
[1] "work"
```

Now let's add an external Rmd content into this file using `child` option

```
```{r child = 'child_chunk_example.Rmd'}
sentence <- c('Let', 'the', 'computer', 'do', 'the', 'work')

for(word in sentence){
  print(word)
}
```
```

---

START of lonely file

#### 5.4.1 Stand alone Rmarkdown file

```
a <- c(1, 1)
b <- c(1, 2)
cor(a, b)
```

```
Warning in cor(a, b): the standard deviation is zero
```

```
[1] NA
```

I'm a very lonely Rmarkdown file, can somebody include me in?

Cheer,

Rmd

END of lonely file

---

## 5.5 Challenge: more code chunks

5 minutes

1. Add `library(tidyverse)` to get lots of messages and try to suppress them with chunk options `message = FALSE`
2. In the example about `child_chunk_example.R` gives warning messages, can you suppress them from the output `message = FALSE`

## 5.6 References

- Section 2.6<sup>1</sup>
- Pimp my RMD<sup>2</sup>

---

<sup>1</sup><https://bookdown.org/yihui/rmarkdown/r-code.html>

<sup>2</sup><https://holtzy.github.io/Pimp-my-rmd/>

## Chapter 6

# More Rmarkdown

We are going to increase the difficulty a little now. We are going to start working towards our final document<sup>1</sup>.

Typically you will have some data set that you are trying to analyse. There are likely to be some other prior steps before you get your tabular data. Those prior steps should also be documented. In this workshop we are going to start with a tabular data set. I found this rather interesting data set at [data.gov.au](https://data.gov.au)<sup>2</sup>, Domestic Airlines - On Time Performance<sup>3</sup> and I decided to investigate it a bit closer.

### 6.1 Setup

First thing first is we need to download it. Note that `read_csv` from `readr`<sup>4</sup> package can “read” directly from url, but I wasn’t sure if everytime I compile Rmarkdown to html it would re-download the file or use cached version. I decide to store a local copy on my computer for speed and simplicity. You should always check a licence on the data set, especially if you are going to publish some of your analysis. This data is *Creative Commons Attribution 3.0 Australia* licence, there is no problem in downloading and using the data.

Let’s open new Rmarkdown file and delete everything from it except the yaml header.

---

<sup>1</sup>example.html

<sup>2</sup><https://data.gov.au>

<sup>3</sup><https://data.gov.au/data/dataset/domestic-airline-on-time-performance>

<sup>4</sup><https://readr.tidyverse.org/>

### 6.1.1 Setting global chunk options

As you have learned already you can manipulate each R chunk with options, but you can also set global settings for each chunk. Let's set `echo = TRUE` and `message = FALSE` globally. This means every R chunk will be echoed i.e shown in the finale document and no messages will appear anywhere in the document.

```
options(encoding="utf-8")

knitr::opts_chunk$set(echo = TRUE,
 message = FALSE)
```

### 6.1.2 Loading libraries

We are going to do our analysis with the help of `tidyverse`<sup>5</sup> library, that in itself includes many other libraries.

```
library(tidyverse)
```

### 6.1.3 Downloading the data

We are doing conditional download here, don't re-download if we already have the file.

```
fn_data <- "domestic_airline_performance.csv"
fn_notes <- "domestic_airline_performance_notes.txt"

if(!file.exists(fn_data)) {
 url_data <- "https://data.gov.au/data/dataset/29128ebd-dbaa-4ff5-8b86-d9f30de56452/r"
 url_notes <- "https://data.gov.au/data/dataset/29128ebd-dbaa-4ff5-8b86-d9f30de56452/r"

 download.file(url_data, fn_data)
 download.file(url_notes, fn_notes)
}

df <- read_csv(fn_data, quote = "")
df
```

```
A tibble: 79,537 x 14
Route Departing_Port Arriving_Port Airline Month Sectors_Schedul~
<chr> <chr> <chr> <chr> <dbl> <dbl>
```

<sup>5</sup><https://www.tidyverse.org/>

```
1 Adel~ Adelaide Brisbane All Ai~ 37987 155
2 Adel~ Adelaide Canberra All Ai~ 37987 75
3 Adel~ Adelaide Gold Coast All Ai~ 37987 40
4 Adel~ Adelaide Melbourne All Ai~ 37987 550
5 Adel~ Adelaide Perth All Ai~ 37987 191
6 Adel~ Adelaide Sydney All Ai~ 37987 486
7 Albu~ Albury Sydney All Ai~ 37987 168
8 Alic~ Alice Springs Sydney All Ai~ 37987 63
9 All ~ All Ports All Ports All Ai~ 37987 31913
10 Bris~ Brisbane Adelaide All Ai~ 37987 155
... with 79,527 more rows, and 8 more variables: Sectors_Flown <dbl>,
Cancellations <dbl>, Departures_On_Time <dbl>, Arrivals_On_Time <dbl>,
Departures_Delayed <dbl>, Arrivals_Delayed <dbl>, Year <dbl>,
Month_Num <dbl>
```

## 6.2 Challenge: 1

2 minutes

### 1. Exploring and familiarising yourself with the data set

The Bureau of Infrastructure, Transport and Regional Economics (BITRE) monitors the punctuality and reliability of major domestic airlines operating between Australian airports. The purpose of this is to allow for evaluation of overall industry and individual airline performance, so that consumers of air travel can make informed decisions.

Information presented in this report is for Australian domestic routes for which the passenger load averaged 8 000 or more passengers per month over the previous six months and where two or more airlines operated in competition on those routes. There were 66 routes that met this definition in April 2016. Over time, routes which meet these criteria change as airline networks and traffic levels vary.

On time arrival - A flight arrival is counted as “on time” if it arrived at the gate before 15 minutes after the scheduled arrival time shown in the carriers’ schedule.

Neither diverted nor cancelled flights count as on time. On time departure - A flight departure is counted as “on time” if it departs the gate before 15 minutes after the scheduled departure time shown in the carriers’ schedule. Cancellation - A flight removed from service within 7 days of scheduled departure is regarded as a cancellation.

Participating airlines report their overall monthly network performance where this is possible. Total industry figures encompass all services operated by reporting airlines only. These airlines collectively carried over 95 per cent of total domestic passengers (regular public transport only) in 2015.

The following domestic airlines currently report this information monthly to the BITRE: Jetstar, Qantas, QantasLink, Regional Express, Tigerair Australia, Virgin Australia and Virgin Australia Regional Airlines. Data has been gathered since November 2003, although some airlines commenced reporting at a later date, including Jetstar which first reported in May 2004, MacAir in July 2005, Tigerair Australia in April 2008 and Virgin Australia Regional Airlines in May 2013. MacAir ceased operations in February 2009 and data was not received for December 2008 onwards. Virgin Blue was rebranded as Virgin Australia in May 2011 and Tiger Airways was rebranded as Tigerair Australia in July 2013 and time series data in Microsoft Excel spread sheet format on the BITRE website have been revised to reflect these changes. Services operated by Skywest on behalf of Virgin Australia using ATR/F100 aircraft commenced in November 2011 and were shown separately as Virgin Australia

### 6.3 Exploring the data

Now that we've got the data lets explore it. It always helps if we can find more information about the data set, particular what information each column might have.

Let's understand a bit better our data set by firstly figuring out how many observations and different variables we have. Instead of executing and showing the code we instead going to keep the chunk "silent" but still executing the R code and we are going to reuse our `d` variable later in the document.

```
```{r echo = F}
d <- df %>% dim
```
```

Later in the text we'll access our `d` variable like you would in R

```
`r d[1]`
```

Lets add the following sentence to our Rmarkdown document and then `knit` to see the results.

```
total number of observation `r d[1]` and total number of variables `r d[2]`
```

Now we are going to look at total number of airlines, include the following code into your Rmarkdown document.

```
```{r echo = F}
df %>%
```



```
select(Airline) %>%
distinct() %>%
arrange(Airline)
```

```

Let's summarise our data to see how many observation each airline has.

```
```{r echo = F}
df %>%
group_by(Airline) %>%
summarise(n = n()) %>%
arrange(-n)
```

```

I hope you have noticed “All Airlines” name in the **Airline** column. I'm fairly certain that this isn't any specif airlines. In general we would need to consult people who we got the data from, but in our case we are simply going to filter those observations out.

```
```{r echo = F}
df2 <- df %>% filter(Airline != 'All Airlines')
```

```

## 6.4 Challenge: 1

5 minutes

1. Can you summarise routes in similar way as we did with airlines? use `group_by(Route)`
2. Can spot an odd route in you summary? If you can filter is our from `df2`. `filter(Route != "All Ports-All Ports")`

## 6.5 Visualising the data

In this section we are going to learn a few more chunk options, all to do with figure manipulation.

- `fig.align`
- `fig.cap`
- `fig.height`

- `fig.width`

Firstly lets make sure we have our data properly filtered, just in case you missed the challenge above

```
```{r}
df2 <- df %>% filter(Airline != 'All Airlines', Route != 'All Ports-All Ports')
```
```

Here we are summarising so that we have an idea of how many times a particular location had be use per airline per year and we are only going to look at two airlines, Jetstar and Qantas.

```
```{r}
df2 %>%
  group_by(Airline, Year, Departing_Port) %>%
  summarise(n = n()) %>%
  ungroup %>%
  filter(Airline == 'Jetstar' | Airline == 'Qantas') %>%
  ggplot(aes(Departing_Port, n, color = Airline)) +
  geom_boxplot() +
  theme(axis.text.x=element_text(angle=45, hjust=1))
```
```

In any given year what is the distribution of cancellation

Note the **warning** message that comes up in the text, let's assume we understand it and let's just turn it off by setting `warning = F`

```
```{r, fig.width = 14, fig.height = 9}
df2 %>%
  filter(Airline == 'Jetstar' | Airline == 'Qantas') %>%
  select(Airline, Departing_Port, Cancellations, Year) %>%
  ggplot(aes(Departing_Port, Cancellations, color = factor(Year))) +
  geom_boxplot() +
  facet_wrap(~Airline) +
  theme(axis.text.x=element_text(angle= 45, hjust=1))
```
```

## Chapter 7

# YAML header

At the very top of your `.Rmd` file you can, optionally, include YAML block. In this block you can fine tune your output document, add some metadata and change document's font and theme. You can also pass additional files such as stylesheet file `.css` and bibliography file `.bib` for text citation. I'm only going to show you a few possible options and will let you explore the rest on your own.

Navigate to the top of your `.Rmd` document and find YAML section there. Just like with the options we passed in to manipulate R code block, YAML block also has **key** = **value** pairs, but instead they are separated by colon ( : ). Now let's add table of content to our document, this will make it easier to navigate your page as well as give nice over view of the content our **key** is **toc** with value **true** or **yes** which one you prefer better.

```

title: 'Hello world'
author: 'Kirill'
date: '13 July 2016'
date: `r format(Sys.time(), '%d %B, %Y')`
output:
 html_document:
 toc: true

```

**Note** that you need to bring `html_document` onto new line and indent it with two spaces. `html_document` is a value of `output` key. `output` can have other values e.g `pdf_document`, `word_document`. However `html_document` also becomes a key for `toc` value and `toc` becomes a key for its own value.

Now that we have sort it initial YAML layout we can continue adding more options to style our HTML document. The other two useful options that I like to pass in are `toc_depth` and `number_sections`

```

title: 'Hello world'
author: 'Kirill'
date: '13 July 2016'
date: `r format(Sys.time(), '%d %B, %Y')`
output:
 html_document:
 toc: true
 toc_depth: 3
 number_sections: yes

```

Most of those options are self explanatory. They best way to learn what each does, is to pass them in. Note that you can comment lines out inside YAML section with `#` symbol.

Good place to start reading about different option can be

```
?rmarkdown::html_document
```

The last two options that can change your document apperance are `theme` and `highlight`. There are nubmer of different themes and highlight options. I suggest you find the one you like in your own time.

Remeber to say that each option will have some default value, sometime default can be None/NULL/NA, but it is still a default.

- `collapsed` (defaults to TRUE) controls whether the TOC appears with only the top-level (e.g., H2) headers. If collapsed initially, the TOC is automatically expanded inline when necessary.
- `smooth_scroll` (defaults to TRUE) controls whether page scrolls are animated when TOC items are navigated to via mouse clicks.

```

title: "Habits"
output:
 html_document:
 toc: true
 toc_float:
 collapsed: false
 smooth_scroll: false

```

## 7.1 Tables

To make your tables user friendly set `df_print: paged`

## 7.2 R slides - ioslides

As I mentioned in previous section, `output` has many options, one of which is `ioslides_presentation`. You can simple add

```

output: ioslides_presentation

```

at the top of your document and your `.Rmd` files will be compiled to slide presentation instead.

Another options is select **presentation** options when you were opening R markdown file. Either way you'll notice YAML header reflects your selected output type. Let's open new R markdown document and let's select presentation instead and let's select HTML (ioslides) option there. You can still save your files as `.Rmd`, and then press the the **Knit HTML** button.

The syntax for the document is more or less the same, except `##` is now used to mark new slide.

## 7.3 Extras

This is mainly to talk about Rnotebook<sup>1</sup> and give you some extra tips about it. Hopefully this will grow into section of it own in the near future.

- to turn inline output (default behaviour) on R markdown documents on/off through settings, *Chunk output inline* / *Chunk output in console*
- Output doesn't go to Viewer/Plots pane, it stays inside the notebook
- Working directory is the location of Rmd file. (I think changing directory with in the chunk isn't good idea)
- In general Rnotebook<sup>2</sup> meant to have better error handling, sends one line at a time for execution, compare to all lines for Rmarkdown document

---

<sup>1</sup>[https://rmarkdown.rstudio.com/r\\_notebooks.html](https://rmarkdown.rstudio.com/r_notebooks.html)

<sup>2</sup>[https://rmarkdown.rstudio.com/r\\_notebooks.html](https://rmarkdown.rstudio.com/r_notebooks.html)



## Chapter 8

# Bookdown

~\\_()\\_/-





## Chapter 9

# Bibliographies

Note that .bib file is yet another plain text file that has some type of structure. Once again we need structure so that computer can parse them out and extract meaningful information

<https://www.lifewire.com/bibtex-file-2619874>

<http://www.bibtex.org/>

<https://tug.org/interviews/patashnik.html>



# Chapter 10

## Work in progress

### 10.1 Tabbed sections

```
Quarterly Results {.tabset}

By Product

(tab content)

By Region

(tab content)

Quarterly Results {.tabset .tabset-fade .tabset-pills}
```

### 10.2 How documents looks

- theme
- highlight
  - default
  - tango
  - pygments
  - kate
  - monochrome
  - espresso
  - zenburn

- haddock
- textmate
- null

### 10.3 Figure options via yaml

This sounds interesting

ok, I've tested out and `fig_height` and `width` via `yaml` do the same thing as when passed through chunk options. I guess `yaml` allows global definition, although one can set chunk options globally too..

also need to cover `out.width = "70%"`

pretty good resource about image resizing [https://sebastiansauer.github.io/figure\\_sizing\\_knitr/](https://sebastiansauer.github.io/figure_sizing_knitr/)

### 10.4 tables Rmarkdown

can't really describe at this stage where this is come from. it appears that it has links with `pagedown` and `paged.js` library

- `paged`

`max.print` The number of rows to print. `rows.print` The number of rows to display. `cols.print` The number of columns to display. `cols.min.print` The minimum number of columns to display. `pages.print` The number of pages to display under page navigation. `paged.print` When set to `FALSE` turns off paged tables. `rownames.print` When set to `FALSE` turns off row names.