

Reproducible Research in R

2019-07-19

Contents

0.1	Learning outcomes	6
0.2	Workshop description	6
0.3	Prerequisite	6
0.4	Keywords	7
0.5	Schedule	7
0.6	Authors and copyright	7
I	Part I	9
1	Introduction to Rmarkdown	13
1.1	Rmarkdown core parts	13
1.1.1	YAML header	14
1.1.2	The R chunks	14
1.1.3	Everything else is plain old markdown	14
1.2	Challenge: Introduction 1	14
1.3	Rmarkdown ecosystem	15
1.3.1	Pandoc	15
1.3.2	YAML	15
1.3.3	LaTeX	15
1.3.4	Knitr	16
1.3.5	Rmarkdown	16
1.3.6	Bookdown	16
1.3.7	Others	16

1.4	Setup	16
1.5	Challenge: Introduction 2	17
1.6	Useful tips	17
2	Vanilla Markdown	19
2.1	Vanilla tags	19
2.2	Practice vanilla markdown	20
2.3	Challenge: Markdown 101	21
3	Rmarkdown	23
3.1	Embedding R code	23
3.1.1	echo and eval	26
3.1.2	include	27
3.1.3	results	27
3.2	Challenge: code chunks	29
II	Part II	31
4	Git and GitHub introduction	33
4.1	Github setup	35
4.2	RStudio setup	39
4.3	GitHub things	39
4.4	More git	39
4.5	git no nos	40
4.6	Starting with git	40
4.6.1	Configuring git	40
4.6.2	Intiating git repository	41
4.6.3	First commit	42
4.7	Which files to commit?	46

<i>CONTENTS</i>	5
5 The R chunk 2	47
5.1 Working with code chunks	47
5.2 Figures related chunk options	51
5.3 Challenge: code chunks	53
5.4 More useful chunk options	54
5.4.1 Stand alone Rmarkdown file	55
5.5 Challenge: more code chunks	56
5.6 References	56
6 More Rmarkdown	57
6.1 Setup	57
6.1.1 Setting global chunk options	58
6.1.2 Loading libraries	58
6.1.3 Downloading the data	58
6.2 Challenge: 1	59
6.3 Exploring the data	60
6.4 Challenge: 1	61
6.5 Visualising the data	61
7 YAML introduction	63
7.1 YAML header	63
7.1.1 Example 1	64
7.1.2 Example 2	64
7.1.3 Example 3	64
7.2 Challenge: YAML 1	64
7.3 General yaml header	65
7.4 Challenge: YAML 2	65
7.5 Rmarkdown yaml header	66
7.6 Rmarkdown rendering	67
7.7 Challenge: YAML 3	67
7.8 More Rmarkdown tags	67
7.9 Presentation slides	68

III Part III	71
8 Bookdown	73
8.1 work in progress	73
9 Bibliographies	75
10 Work in progress	77
10.1 YAML	77
10.2 LaTeX	77
10.3 Tabbed sections	78
10.4 Figure options via yaml	78
10.5 tables Rmarkdown	78
Appendix	78
A Appendix	81
A.1 Long list of chunk options	82

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.

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

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

³<https://www.monash.edu/research/infrastructure/bioinformatics>

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

⁵<https://carpentries.org/>

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 tool⁶ 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 GitHub⁷ 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 full⁸, if you have to prioritise, read at least these section (1,2,6,10).

⁶<https://git-scm.com/book/en/v1/Getting-Started-About-Version-Control>

⁷<https://github.com>

⁸<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 Harrison⁹
- Adele Barugahare¹⁰
- Kirill Tsyganov¹¹

This work is licensed under a CC BY-4: Creative Commons Attribution 4.0 International License¹². The attribution is “Monash Bioinformatics Platform” if copying or modifying these notes.



⁹<mailto:paul.harrison@monash.edu>

¹⁰<mailto:Adele.Barugahare@monash.edu>

¹¹<mailto:kirill.tsyganov@monash.edu>

¹²<http://creativecommons.org/licenses/by/4.0/>

Part I

Part I

In this part of the book we will begin by talking about Rmarkdown language, where it has originated and the ecosystem that surrounds it. We are then going to look closely at the core parts of the Rmarkdown document. After that we will start with understanding basics of vanilla markdown as a building block of Rmarkdown. We will finish off this section of the book with hands on experience writing Rmarkdown documents with embedded R code chunks.

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 pandoc⁷ to convert markdown to other document types. knitr⁸ 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 knitr⁹ 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 pandoc¹¹ 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 rmarkdown¹³ 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.

- xaringan¹⁴
- blogdown¹⁵
- thesisdown¹⁶

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```
```





Part II

Part II



## 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 visiable, 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>

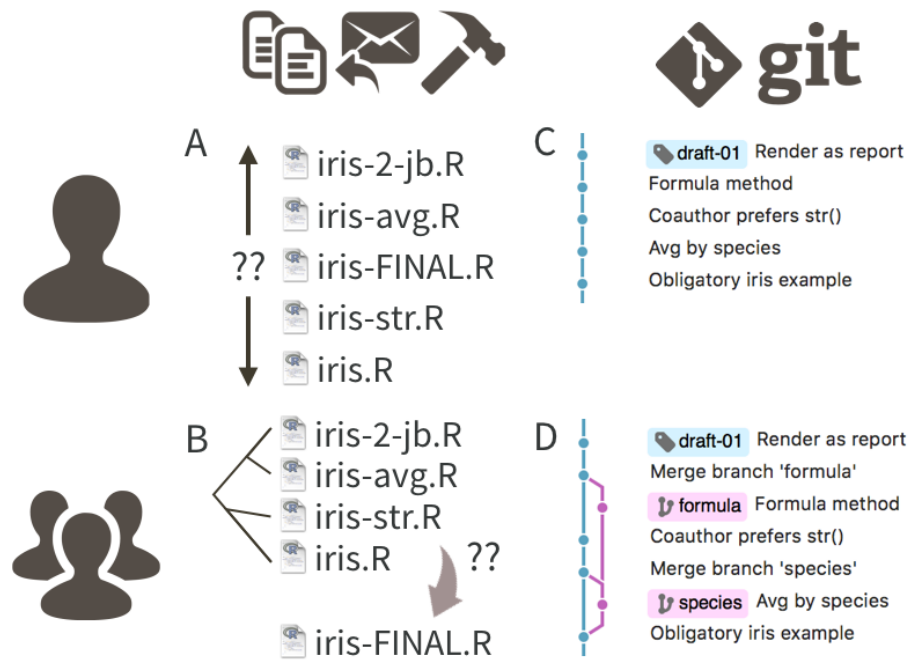


Figure 4.1: This is an example of git version control vs DIY versioning via filesystem



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

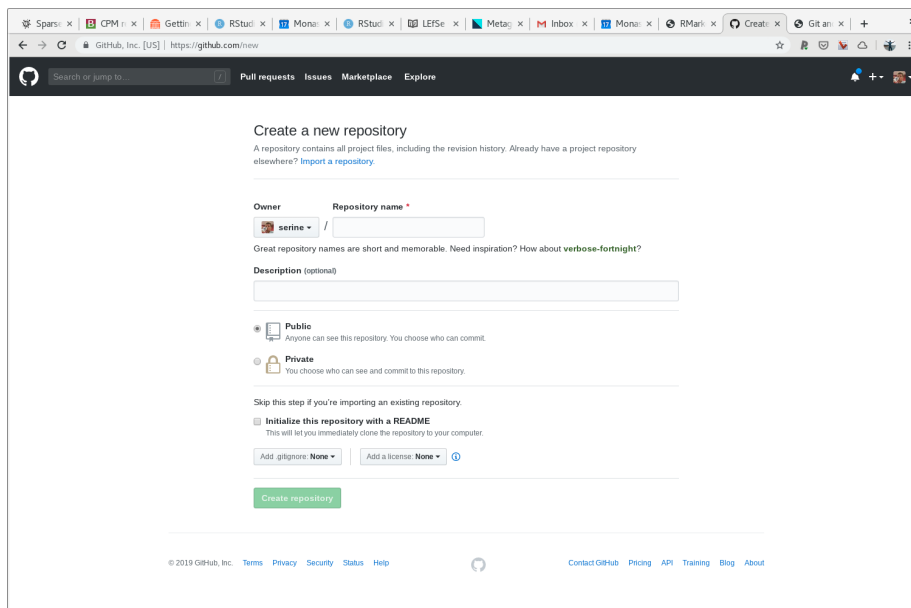
Git and GitHub will help you:

- 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 the GitHub logo and links for "Terms", "Privacy", "Security", "Status", "Help", "Contact GitHub", "Pricing", "API", "Training", "Blog", and "About".

<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.

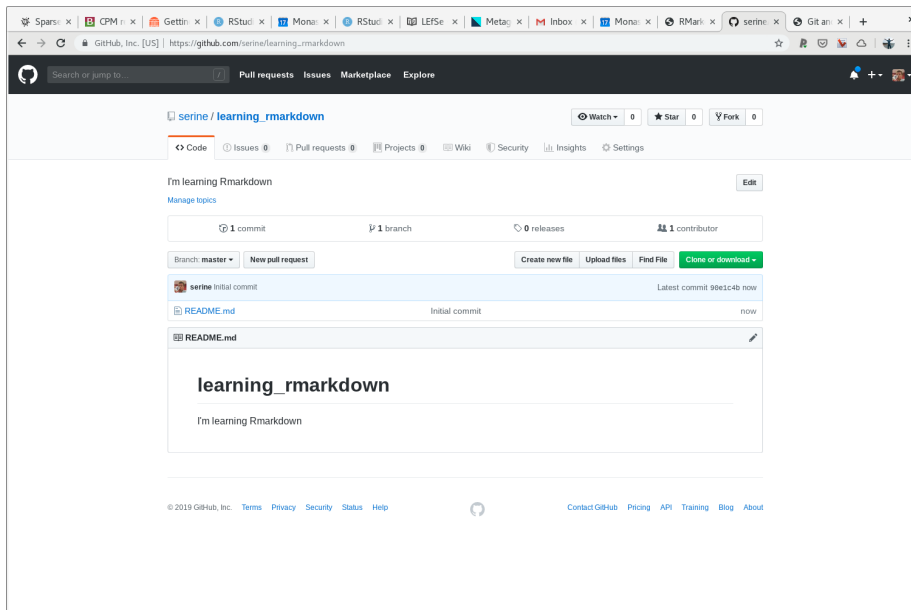
The screenshot shows the GitHub 'Create a new repository' page. The browser's address bar shows 'https://github.com/new'. The page has a dark header with navigation links: 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. The main content area is titled 'Create a new repository' and includes a 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:

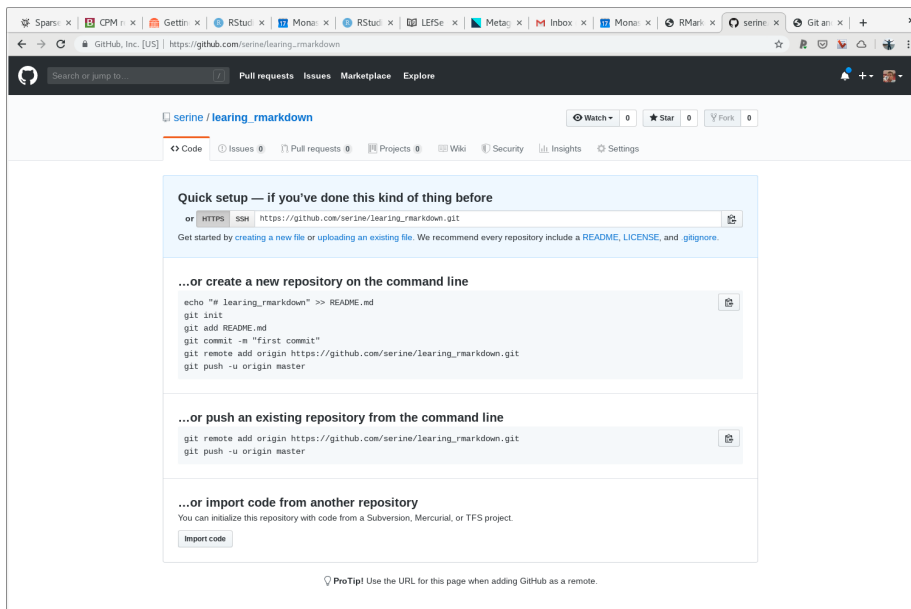
- Owner:** 'serine' (with a dropdown arrow).
- Repository name:** 'learning\_markdown' (with a green checkmark).
- Description (optional):** 'I'm learning Rmarkdown'.
- Visibility:** 'Public' (selected) and 'Private' (unselected).
- Initialize this repository with a README:** Checked (indicated by a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

A green 'Create repository' button is located at the bottom of the form. The footer of the page includes copyright information '© 2019 GitHub, Inc.' and various links: '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.

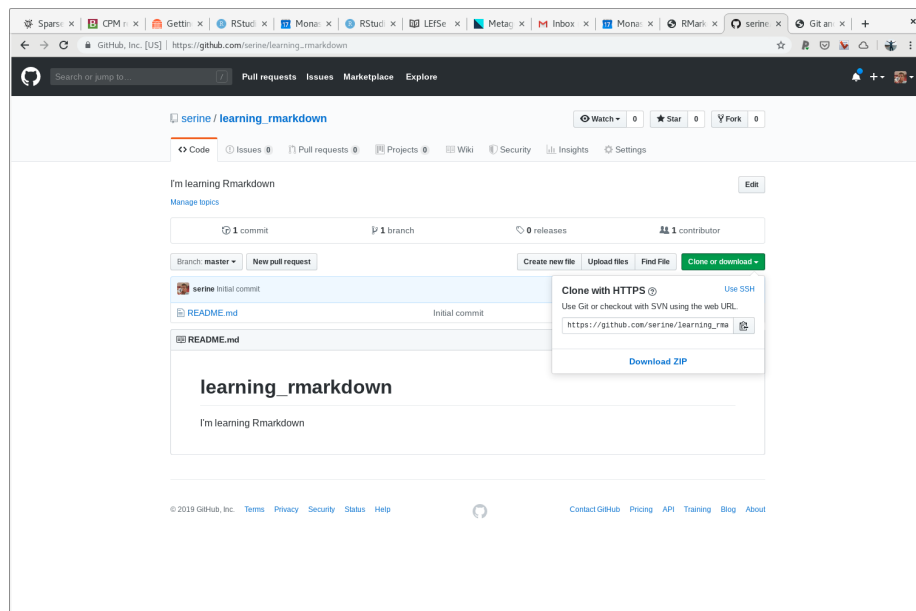
This is how typical URL looks like

`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>

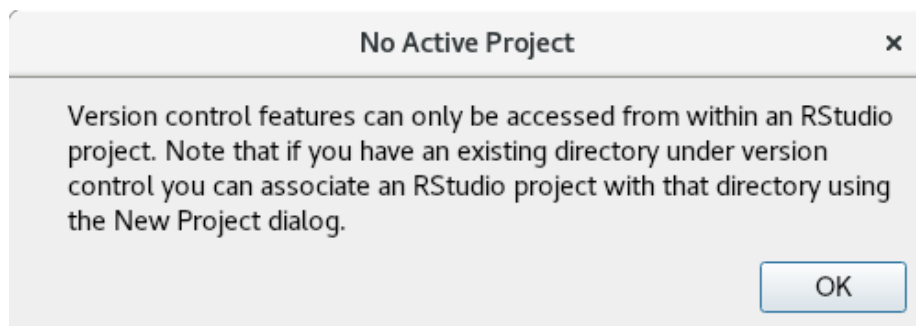
cli. Do take the simplest and quickest path to get your work version controlled. No one will care which client you are using in the end.

## 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.

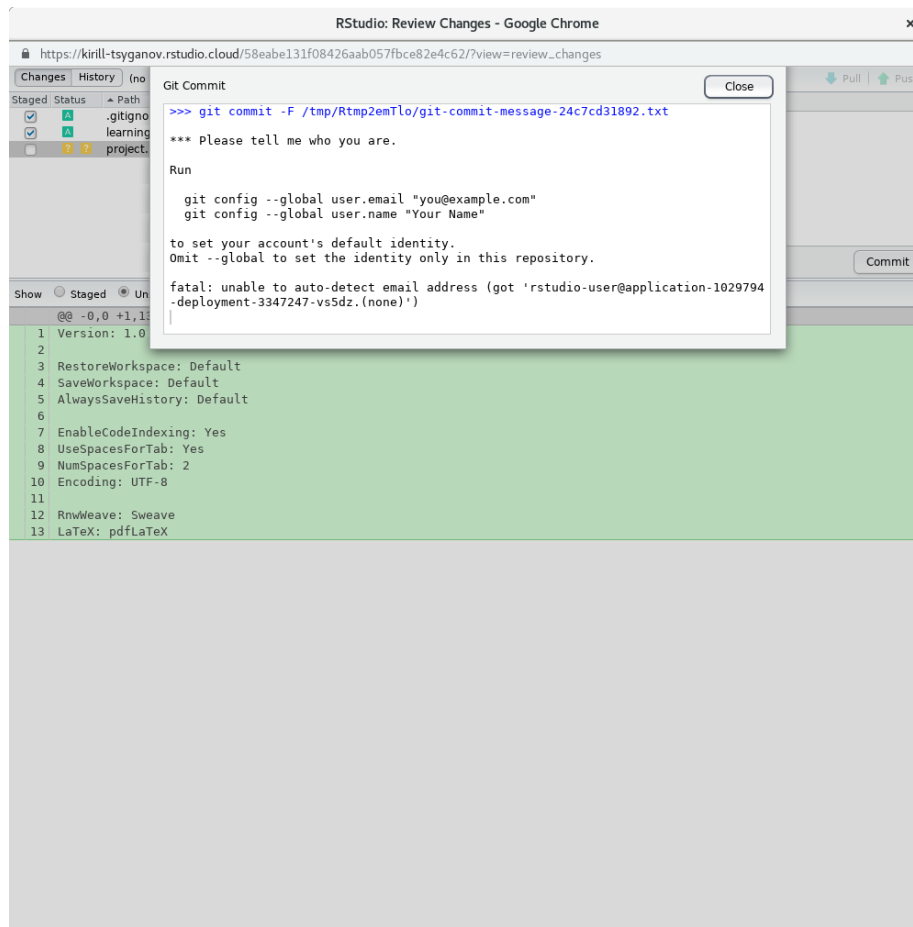
Let's open up a terminal and run a couple of `git` commands

Tools -> Terminal -> New Terminal

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

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

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



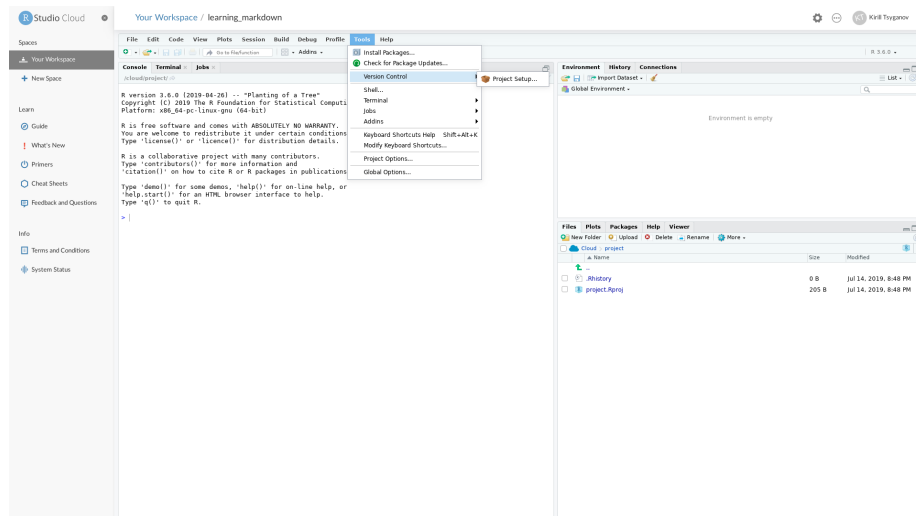
### 4.6.2 Initiating git repository

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

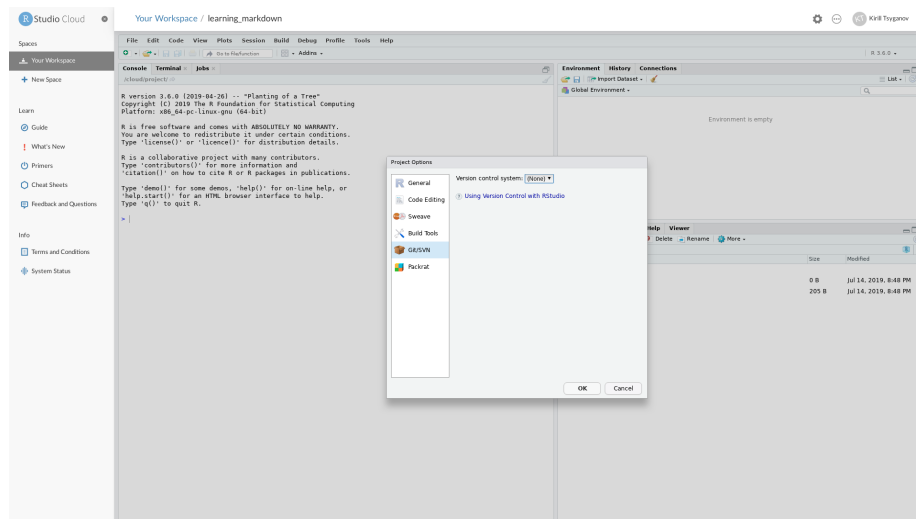
Rproject folder == Rproject directory == git repository == git repo

Let's initiate git repository

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



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



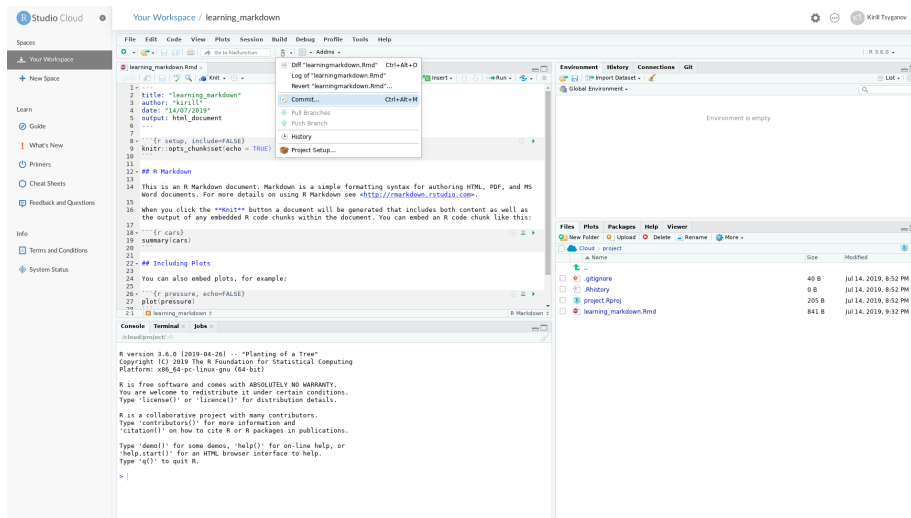
The command line equivalent is navigating to your project directory and running `git init`

### 4.6.3 First commit

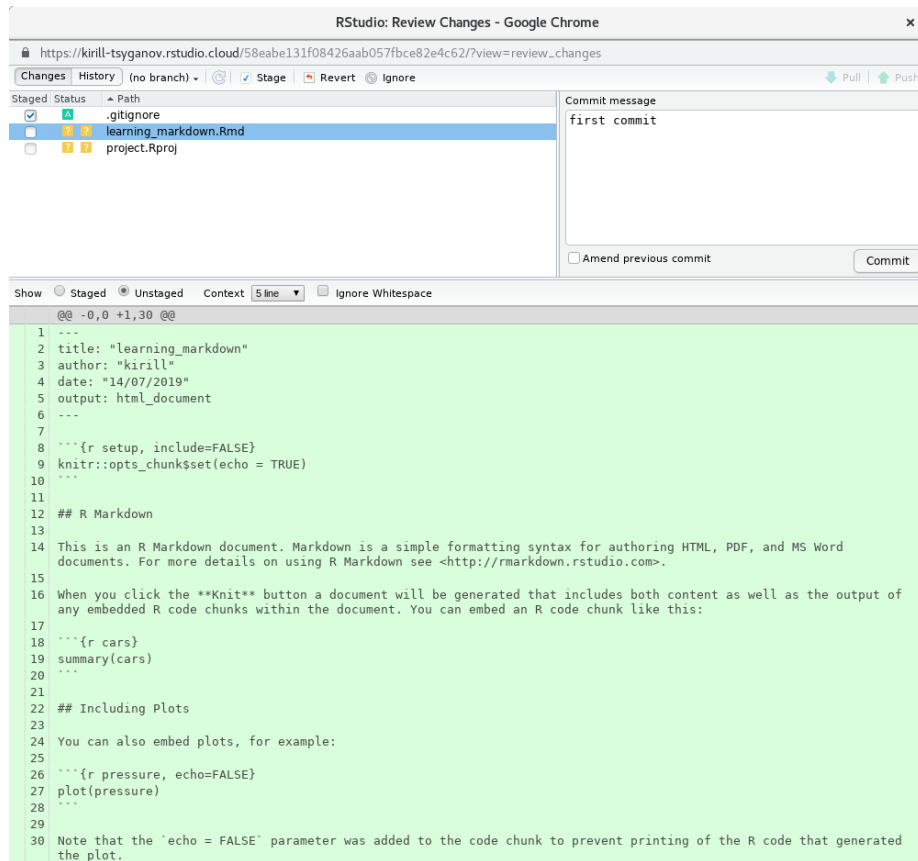
Let's make our first commit, use drop down menu as indicated on the image below to select `commit` option

## 4.6. STARTING WITH GIT

45



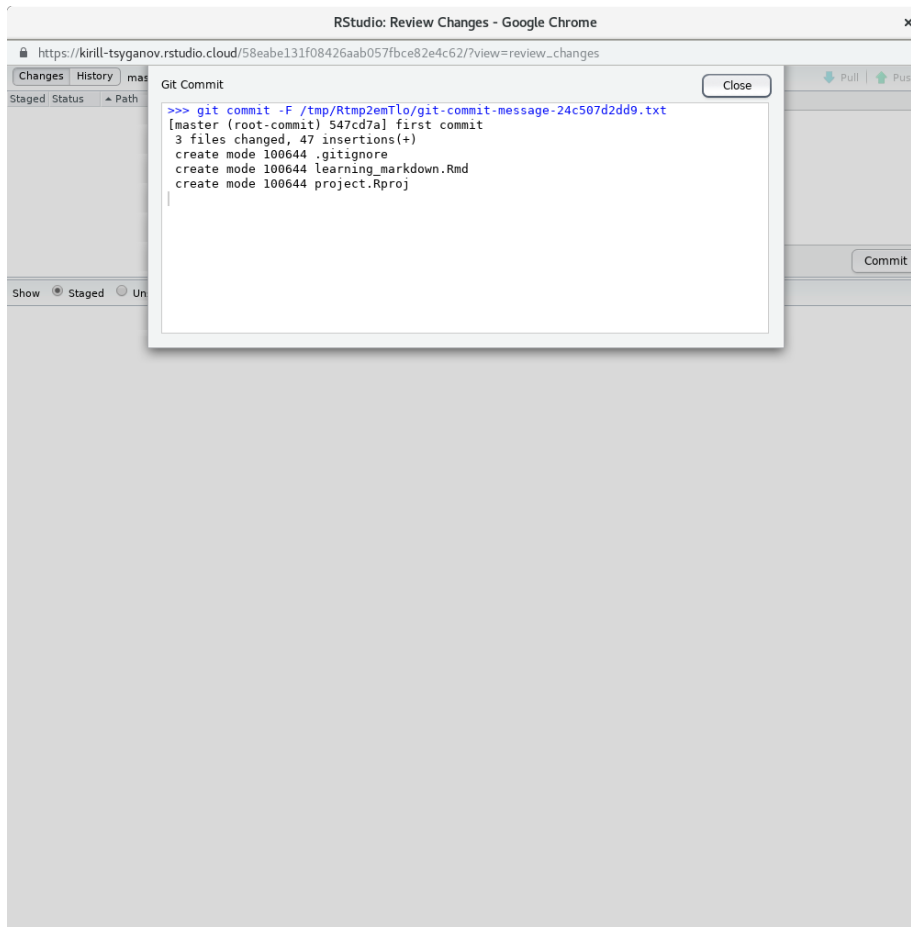
You should see a new window popped up



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)
```

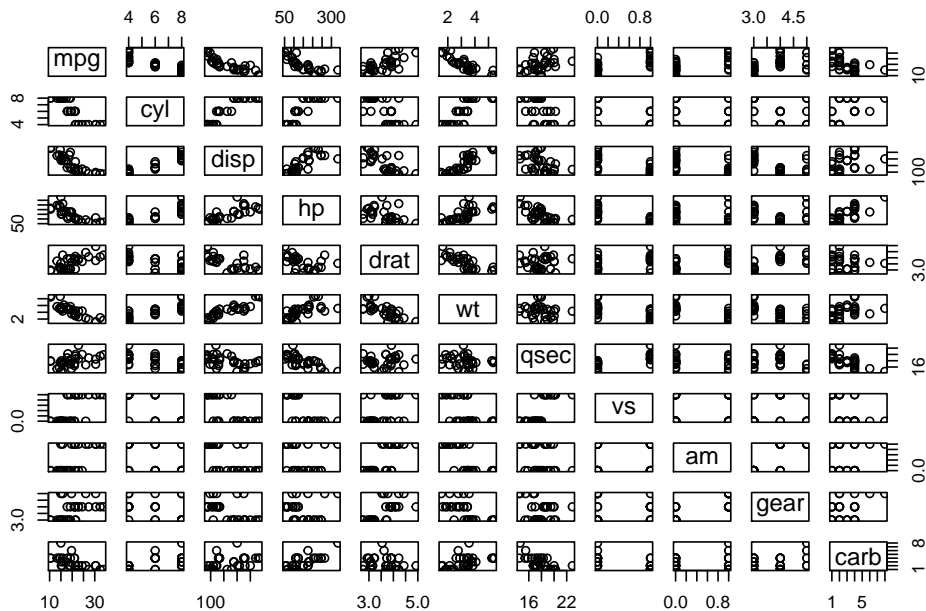
```
mpg cyl disp hp
Min. :10.40 Min. :4.000 Min. : 71.1 Min. : 52.0
1st Qu.:15.43 1st Qu.:4.000 1st Qu.:120.8 1st Qu.: 96.5
Median :19.20 Median :6.000 Median :196.3 Median :123.0
Mean :20.09 Mean :6.188 Mean :230.7 Mean :146.7
3rd Qu.:22.80 3rd Qu.:8.000 3rd Qu.:326.0 3rd Qu.:180.0
Max. :33.90 Max. :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)
```
```

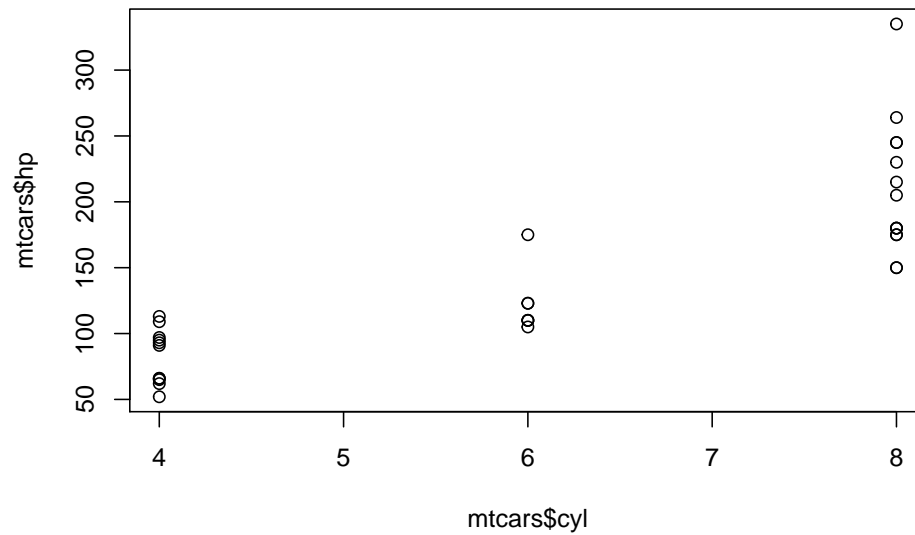
```
plot(mtcars)
```



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)
```

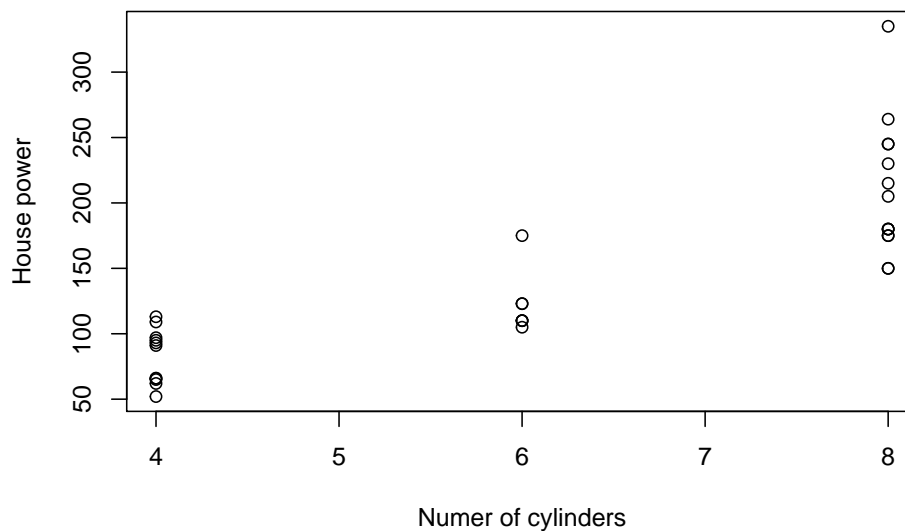


We can add a bit more information to our plot, to make more self descriptive.

```
```{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')
```

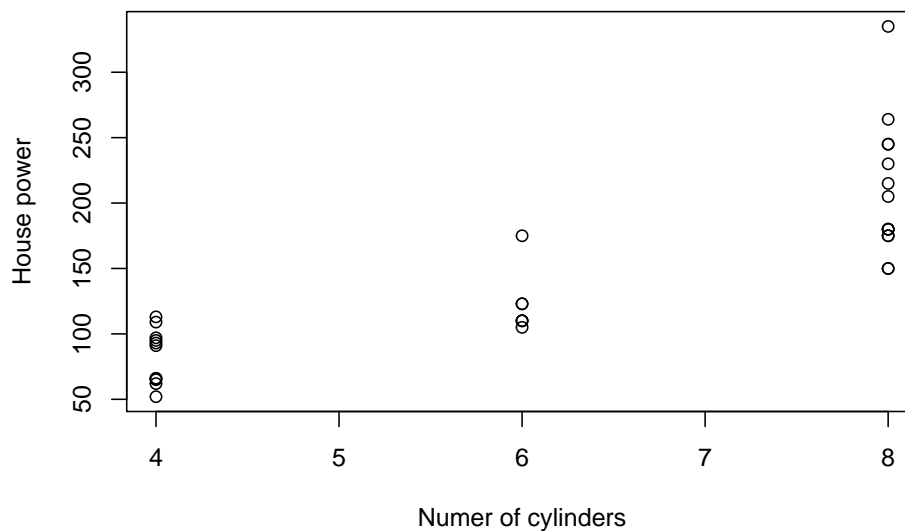
### House power vs number of cylinders



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

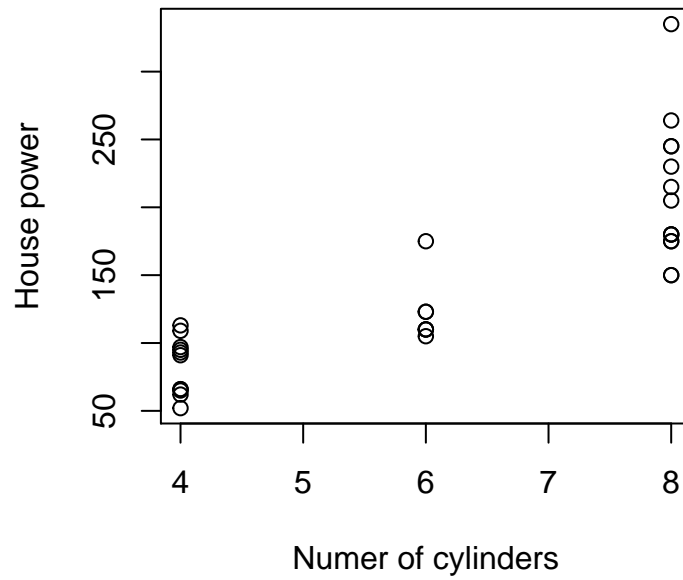
- `fig.align` - left, right, center or default (left)
- `fig.height` - height specified in inches
- `fig.width` - width specified in inches
- `fig.cap` - string of text in quotes

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')
```
```

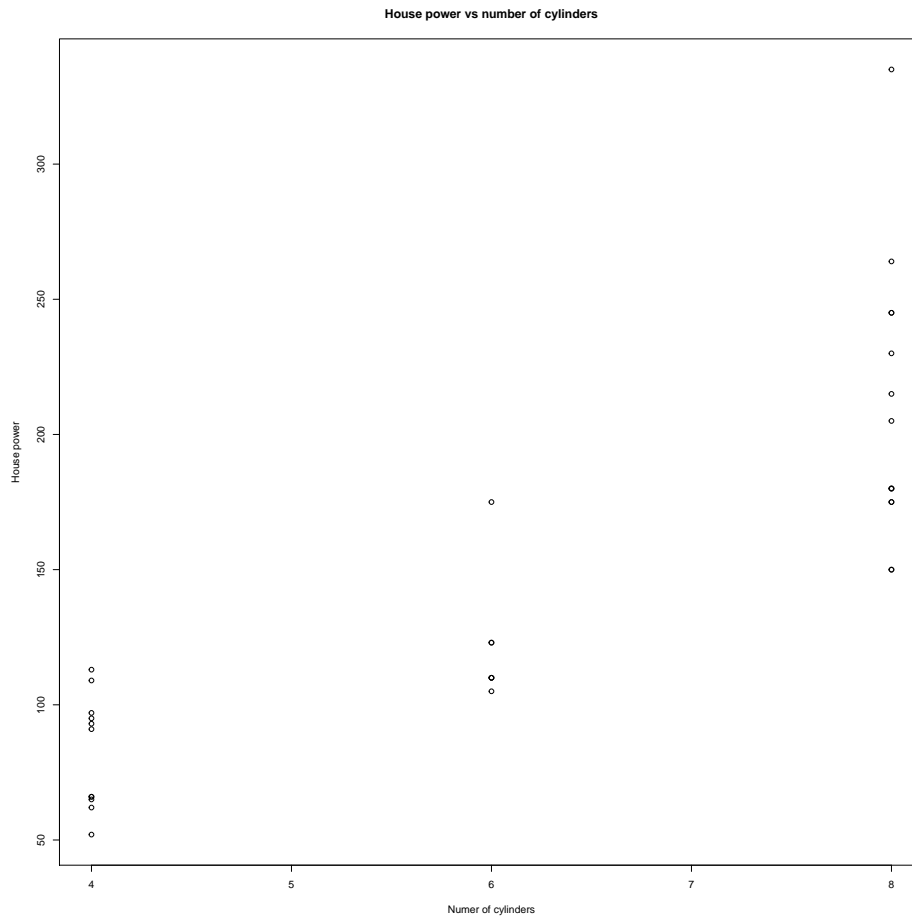
```
plot(mtcars$cyl,
 mtcars$hp,
 main='House power vs number of cylinders',
 xlab = 'Numer of cylinders',
 ylab='House power')
```

### 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 introduction

As it was mentioned earlier in the book yaml is a stand along language, often it is used as a configuration file, which is true in the case of Rmarkdown. We are going to use it in conjunction with Rmarkdown documents by embedding yaml blob into Rmarkdown files, just like we did with R chunks. YAML blob, or in the context of Rmarkdown YAML header, must be at the very top of your Rmarkdown file. The yaml header enable high degree of customisation for our final documents. We can also use yaml header to “attach” or reference other external files such as bibliographies and styling in the form of css<sup>1</sup>.

For the purpose of configuring Rmarkdown documents you need to know three variable types:

- **scalar** stand along value e.g 3 or “car”
- **list** a collection of items e.g [“learing”, “to”, “use” “yaml”]
- **map** a different collection type that can hold simple types. e.g rmd\_files: [“00-index.Rmd”, “01-introduction.Rmd”]

### 7.1 YAML header

YAML is relatively similar to another file format JSON<sup>2</sup>, if you are familiar with one you should have little problem wrapping your head around the other. Fundamentally both file types trying to provide structured relationship between items via key = value pairing. Keys sometimes interchangeably used with tags. Remember that yaml language, unlike json, is very sensitive to spaces and indentations. Below are some examples of valid and invalid yaml syntax.

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Cascading\\_Style\\_Sheets](https://en.wikipedia.org/wiki/Cascading_Style_Sheets)

<sup>2</sup><https://en.wikipedia.org/wiki/JSON>

### 7.1.1 Example 1

This is a valid yaml header. In this exapmle we have a key `title` with scalar value `Learning Rmarkdown`

```

title: "Learning Rmarkdown"

```

### 7.1.2 Example 2

This is also valid YAML header. Here we have `rmd_files` key that has as it's value a list of items. In yaml there is two ways one can specify a list.

```

rmd_files:
 - 00-index.Rmd
 - 01-introduction.Rmd

```

And

This is an alternative way to specify a list.

```

rmd_files: ["00-index.Rmd", "01-introduction.Rmd"]

```

### 7.1.3 Example 3

This on the other hand isn't valid YAML header

```

rmd_files:
 00-index.Rmd
 01-introduction.Rmd

```

## 7.2 Challenge: YAML 1

5 minutes

1. Is this a valid YAML header?

```

title: "Hello world!"
author: Me!

```

Yes, everything looks good to me

2. What about this YAML header?

```

title: "Hello world!"
author: Me!
bookdown::gitbook:
 config:
 toc:
 collapse: subsection

```

Yes, this is highly neted, but still a completely valid yaml header

3. How do you get a list of all possible keys and values (discussion question) ?  
Read the docs (will need to expand this answer)

## 7.3 General yaml header

This handful of tags are general between different document types. Most other key, values are aimed at the specific document type. In the next few sections we will look at yamls keys specific for Rmarkdown, bookdown and pdf final files configuration.

```

title: 'Hello world'
author: 'Kirill'
date: '13 July 2016'

```

## 7.4 Challenge: YAML 2

5 minutes

3. Is this a valid yaml?

```

date: 19 July, 2019

```

It is certainly valid by Rmarkdown standards, but it may not be in other systems

## 7.5 Rmarkdown yaml header

Here we are starting to see another new key `output` with a value `html_document`

```

title: 'Hello world'
author: 'Kirill'
date: '13 July 2016'
output: html_document

```

These are some of the possible values that `output` can take

- `html_notebook` Interactive R Notebooks
- `html_document` HTML document w/ Bootstrap CSS
- `pdf_document` PDF document (via LaTeX template)
- `word_document` Microsoft Word document (docx)
- `odt_document` OpenDocument Text document
- `rtf_document` Rich Text Format document
- `md_document` Markdown document (various flavors)

Turns out that each one of those values is also a key that can take other values.

```

title: 'Hello world'
author: 'Kirill'
date: '13 July 2016'
output:
 html_document:
 toc: true
 toc_depth: 3
 number_sections: yes

```

Note that indentation is very important here as it reflects the relationship between key value pairs.

## 7.6 Rmarkdown rendering

As we have discussed in the introduction Rmarkdown is an ambiguous word. It could mean an rmarkdown R package<sup>3</sup> that converts or renders markdown into various document types. We can also be referring to an Rmarkdown document that we are writing. And we also could use it in a more generic sense referring to the ecosystem.

Here I'm going to specifically talk about Rmarkdown R package and the `render` function i.e `rmarkdown::render()`. `render` can take an option for output format. You can either specify the output format by directly giving it a function i.e

```
rmarkdown::render(output_format = "html_document")
```

Or pass that same information via yaml header. As you might have noted above, `html_document` itself has a bunch of options. There would be a way to pass them manually in R console, but it is much nice and more reproducible to pass them via yaml header.

The two way to figure out which options are available for `html_document` is to either google (an obvious one) or take a look at the help page for that function i.e

```
?rmarkdown::html_document
```

## 7.7 Challenge: YAML 3

5 minutes

1. Can you find in the help page which values `df_print` from `html_document` can take? run `?rmarkdown::html_document` in Rstudio console to open help page and search for `df_print`
2. Set `df_print` to a value that will give you paginated HTML table. `paged`

## 7.8 More Rmarkdown tags

Let's add more options to our yaml file and `knit` it and see what has changed.

---

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

```

title: 'Hello world'
author: 'Kirill'
date: '13 July 2016'
output:
 html_document:
 df_print: paged
 toc: true
 toc_float: true
 toc_collapsed: true
 toc_depth: 3
 number_sections: false
 code_folding: "hide"
 theme: sandstone
 highlight: espresso

```

As I have eluted earlier if you simply change `output: html_document` to `output: pdf_document` you are going to get pdf document instead. Also remember that all of those options that we gave to `html_document` are specific to that document type only, although some tags might have the same name in other document types e.g `toc` and `toc_depth` keys also exist in `pdf_document`. We are not going to spend time on building a pdf document. It is a more finicky to build because it relying on LaTeX engine and LaTeX can be very finicky. This is more relevant for more complex documents. This is why we recommend to firstly build html document and only once you have finished you analysis and start building your pdf documents.

The more interesting, in my view, document type is `ioslides_presentation`

## 7.9 Presentation slides

As I've mentioned in previous section, `output` has many options, one of which is `ioslides_presentation`. Let's comment `html_document` with all it's options out for now, add `output: ioslides_presentation` and re-compile out document

```

title: 'Hello world'
author: 'Kirill'
date: '13 July 2016'
output: ioslides_presentation
#output:
html_document:

```

```
df_print: paged
toc: true
toc_float: true
toc_collapsed: true
toc_depth: 3
number_sections: false
code_folding: "hide"
theme: sandstone
highlight: espresso

```

As you might have guessed `##` symbol in the case of `ioslides_presentation` means the beginning of the slide. While amount of work is minimal to convert between `html_document` and `ioslides_presentation` you will obviously need to reduce amount of text.





Part III

Part III



## Chapter 8

# Bookdown

~\\_()\\_/-

All `.Rmd` files located in the same directory will be compiled into the book in the (alphabetical?) order.

- files that start with an underscore are skipped
- if there is an `index.Rmd` it will always be treated as a first file
- those settings can be overwritten via `_bookdown.yml`
- `_bookdown.yml` must co-exist with `.Rmd` files in the “book” directory

Bookdown has extended markdown even further for math <https://bookdown.org/yihui/bookdown/markdown-extensions-by-bookdown.html#tab:theorem-envs>

### 8.1 work in progress

Besides these `html_document()` options,

`gitbook()` has three other arguments:

- `split_by` argument specifies how you want to split the HTML output into multiple pages
  - `rmd`: use the base filenames of the input `Rmd` files to create the HTML filenames, e.g., generate `chapter3.html` for `chapter3.Rmd`.
  - `none`: do not split the HTML file (the book will be a single HTML file).

- `chapter`: split the file by the first-level headers.
  - `section`: split the file by the second-level headers.
  - `chapter+number` and `section+number`: similar to `chapter` and `section`, but the files will be numbered.
- `split_bib`
- `config`
- `collapse`
  - `section`
  - `subsection`
  - `scroll_highlight`: yes before: null

## 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>

A normal paragraph.

```
plot(cars) # a scatterplot
```

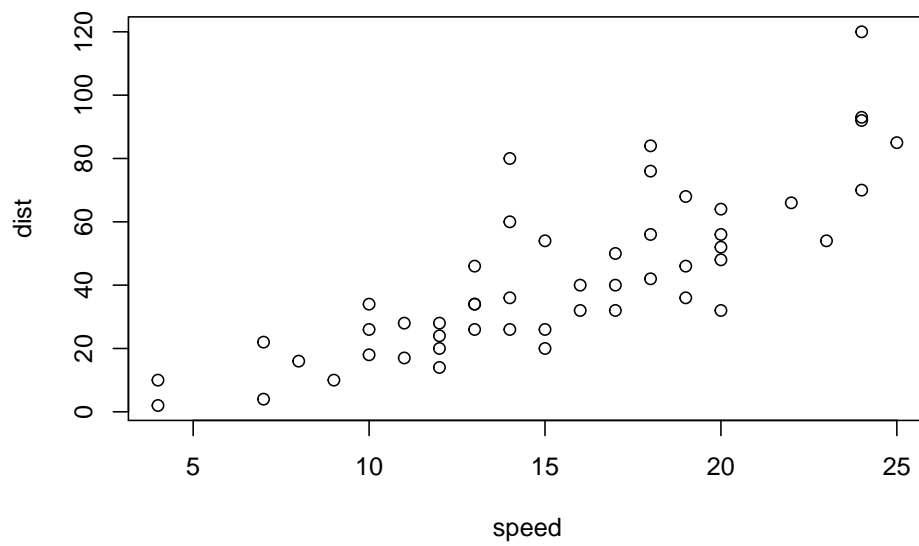


Figure 9.1: A scatterplot of the data `cars` using **base** R graphics.

## Chapter 10

# Work in progress

In general think about fast turn around time and ease of editing.  
this will remove numbering from that header

# Preface {-}

### 10.1 YAML

WE are using yaml language define certain parameters that meant to do to different tools

some are designated for pandoc other for bookdown package other general rmarkdown/knitr settings

### 10.2 LaTeX

$$f(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

$$\begin{array}{ccc} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{array}$$

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \end{bmatrix}$$

### 10.3 Tabbed sections

```
Quarterly Results {.tabset}

By Product

(tab content)

By Region

(tab content)

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

### 10.4 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/)

### 10.5 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.

```
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)
```





## Appendix A

# Appendix

### A.1 Long list of chunk options

| name           | value     | type            | description                                                                        |
|----------------|-----------|-----------------|------------------------------------------------------------------------------------|
| child          | NULL      | code_evaluation | A character vector of filenames. Knitr will knit the files in the order specified. |
| code           | NULL      | code_evaluation | Set to R code. Knitr will replace the code in the chunk with the code specified.   |
| engine         | 'R'       | code_evaluation | Knitr will evaluate the chunk in the named language.                               |
| echo           | TRUE      | results         | If FALSE, knitr will not display the code in the chunk.                            |
| 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 results.                    |
| purl           | TRUE      | code_evaluation | If FALSE, knitr will not include the chunk when running the document.              |
| collapse       | FALSE     | results         | If TRUE, knitr will collapse all the source and output into a single line.         |
| results        | 'markup'  | results         | If 'hide', knitr will not display the code's results in the document.              |
| error          | TRUE      | results         | If FALSE, knitr will not display any error messages.                               |
| message        | TRUE      | results         | If FALSE, knitr will not display any messages generated by the chunk.              |
| warning        | TRUE      | results         | If FALSE, knitr will not display any warning messages.                             |
| comment        | '##'      | code_decoration | A character string. Knitr will append the string to the end of each line of code.  |
| highlight      | TRUE      | code_decoration | If TRUE, knitr will highlight the source code in the document.                     |
| prompt         | FALSE     | code_decoration | If TRUE, knitr will add > to the start of each line of code.                       |
| strip.white    | TRUE      | code_decoration | If TRUE, knitr will remove white spaces that appear at the start of each line.     |
| tidy           | FALSE     | code_decoration | If TRUE, knitr will tidy code chunks for display with the pretty() function.       |
| opts.label     | NULL      | chunks          | The label of options set in knitr::opts_template() to use with the chunk.          |
| R.options      | NULL      | chunks          | Local R options to use with the chunk. Options are passed to the R process.        |
| ref.label      | NULL      | chunks          | A character vector of labels of the chunks from which to pull references.          |
| autodep        | FALSE     | cache           | If TRUE, knitr will attempt to figure out dependencies between chunks.             |
| cache          | FALSE     | cache           | If TRUE, knitr will cache the results to reuse in future runs.                     |
| cache.comments | NULL      | cache           | If FALSE, knitr will not rerun the chunk if only a comment has changed.            |
| cache.lazy     | TRUE      | cache           | If TRUE, knitr will use lazyload() to load objects in the cache.                   |
| cache.path     | 'cache/'  | cache           | cache.vars NULL A character vector of object names to cache.                       |
| dependson      | NULL      | cache           | A character vector of chunk labels to depend on.                                   |
| dev            | 'png'     | plots           | The R function name that will be used as a graphics device.                        |
| dev.args       | NULL      | plots           | Arguments to be passed to the device, e.g. dev.args=c("width=100", "height=100").  |
| dpi            | 72        | plots           | A number for knitr to use as the dots per inch (dpi).                              |
| external       | TRUE      | plots           | If TRUE, knitr will externalize tikz graphics to save space.                       |
| fig.align      | 'default' | plots           | How to align graphics in the final document. One of left, center, right.           |
| fig.cap        | NULL      | plots           | A character string to be used as a figure caption in the document.                 |
| fig.env        | 'figure'  | plots           | The Latex environment for figures.                                                 |
| fig.ext        | NULL      | plots           | The file extension for figure output, e.g. fig.ext='png'.                          |
| fig.height     | 7         | plots           | The height to use in R for plots created by the chunk.                             |