Reproducible Research in R

Version 1

Contents

Reproducible Research in R

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

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 tool[1] 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 GitHub[2] can provide centralised location for your research. All together Rmarkdown, git and github can enable reproducible research that is visiable and accessible by greater public including supervisors and management.

0.3 Prerequisite

This is an introductory level workshop, however some prior exposure to R is assumed.

0.4 Keywords

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

0.5 Notes

These are course notes for the "Reproducible Research in R" course given by the Monash Bioinformatics Platform[3] for the Monash Data Fluency[4] initiative. Our teaching style is based on the style of The Carpentries[5].

- PDF version for printing[6]

Source code

- GitHub page[7]

[1] https://git-scm.com/book/en/v1/Getting-Started-About-Version-Control
[2] https://github.com
[3] https://www.monash.edu/researchinfrastructure/bioinformatics
[4] https://monashdatafluency.github.io/
[5] https://carpentries.org/
[6] https://monashdatafluency.github.io/r-rep-res/Reproducible-Research-in-R.pdf
[7] https://github.com/MonashDataFluency/r-rep-res

Authors and copyright

This course is developed for the Monash Bioinformatics Platform by Paul Harrison[8], Adele Barugahare[9] and Kirill Tsyganov[10]

0.6 References

- knitr vs rmarkdown vs bookdown[11]

[8] mailto:paul.harrison@monash.edu
[9] mailto:Adele.Barugahare@monash.edu
[10] mailto:kirill.tsyganov@monash.edu
[11] https://stackoverflow.com/questions/40563479/relationship-between-r-markdown-knitr-pandoc-and-bookdown
[12] http://creativecommons.org/licenses/by/4.0/

Chapter 1

Introduction to Rmarkdown

Work towards remote repository becomes your "cannonical" location for the project and gives you nice workflow with your supervisor

Rmarkdown has become much more then just embedding R code into a document. For me it has essentially become the status quo of doing research. These days Rmarkdown document goes without saying with any analysis I do. The Rmarkdown incepts the right practices of doing analysis; it makes you write things down, to explain to yourself and others what you did, it makes your analysis open, anyone can see exactly what you've done and it makes your analysis reproducible since you'll be able to re-run that code to get those same tables and figures again and again. The ultimate goal for any projects, be that honours, PhD or any other analytical work, to be logged and shared via Rmarkdown with peers and supervisor(s). Depending on the aspects of the work, sharing should be done via remote repositories, for example github or gitlab two popular resources.

Rmarkdown is a natural evolution of vanilla markdown, backed by R package `knitr` and an external tool `pandoc`. Markdown is a powerful language for writing plain text documents that are easily parsed and/or converted to other more sophisticated document types, for example **PDF** or **HTML** to name a couple.

There are other plain text languages alike markdown that have the same goal of simplify documents writing and with means of light text tagging or marking up ability to indicate important parts of the text such as titles, paragraphs and code blocks. The underlying idea for then markdown is that it is easy-to-write and easy-to-read. In this workshop we will be mainly focusing on Rmarkdown flavour, although it has really diverged into it's own language essentially now. And although you can use any text editor[1] to write your markdown, we'll be using RStudio[2] for all of our work.

[1] https://en.wikipedia.org/wiki/Text_editor
[2] https://rstudio.com

1.1 Rmarkdown document 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 a final document. Each one of those parts can be customised with further options, which will cover later in the book. One thing to note is that Rmarkdown is multi-component ecosystem, for example YAML[3] is a stand alone language and can be used outside of Rmarkdown context. For example ansible configuration management software[4] uses YAMl as internal definition language. In Rmarkdown case we are using YAML header to pass additional information about type and apperance of the final document, but more on that later.

1.1.1 YAML header section

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

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

1.1.3 Everything else is plain old markdown

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

[3] https://en.wikipedia.org/wiki/YAML
[4] https://en.wikipedia.org/wiki/Ansible_(software)

1.2 Challenge: Rmarkdown 101

5 minutes

1. What file extention should we typically use for saving our **Rm**ark**d**own files? answer link[5]

2. What document types can be produced (compiled) from Rmarkdown?

3. Will I have to learn more "languges" 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 Rmarkdown[6]

1.3 Useful to know

> One cannot know all of Rmarkdown wisdom. One would continuesly learn to make oneself more Rsome. — Kirill Tsyganov (2019)

- don't attempt to compile to `pdf_document` until absolutelly necesserary. `LaTeX` engine that is used for 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, intermidiate compilation to `html_documents` are totaly fine, before worrying about technical issues

[5] https://superuser.com/questions/249436/file-extension-for-markdown-files
[6] link%20to%20github%20that%20the%20line%20of%20code%20above

Chapter 2

Vanilla Markdown

The original (vanilla) version of Markdown invented by John Gruber[1] defines a handful tags, discussed shortly. Rmarkdown[2] isn't the only flavour, CommonMark[3] that tries to unify all the different flavours and GitHub Flavored Markdown (GFM)[4] which mainly enhances content appears on github site.

2.1 Vanilla tags

```
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

I know I said vanilla, but we are launching an Rmarkdown file. It doesn't matter for two reasons, under the hood Rmarkdown will always be converted to vanilla markdown and we are simply using RStudio to write text, can be anything.

From now onward we are going to start using `knitr` is an R package that does all the magic of converting and running your R markdown and R code respectively.

[1] https://en.wikipedia.org/wiki/Markdown
[2] https://rmarkdown.rstudio.com/
[3] http://commonmark.org/
[4] https://guides.github.com/features/mastering-markdown/

There actually not that much to core (vanila) of markdown essentially all of it can be summarised below

```
# 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 markdown[5] is comprised entirely of punctuation characters.

2.3 Challenge: Markdown 101

5 minutes

1. How to mark text so that it appears underlined? answer link[6]

2. Can markdown replace html[7] (discussion question)? It has replaced html and latex in documention 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. On would not want to replace html + css for large website project

[5] https://daringfireball.net/projects/markdown/syntax

[6] https://softwareengineering.stackexchange.com/questions/207727/why-there-is-no-markdown-for-underline

[7] https://en.wikipedia.org/wiki/HTML

Chapter 3

Rmarkdown ecosystem

Rmarkdown ecosystem is relatively complicated and 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. By and large the whole ecosystem rotates around pandoc[1] tool.

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.

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] https://pandoc.org/

3.3 Knitr

As was mentioned before we are using pandoc[2] to convert markdown to other document types. knitr[3] provides function to convert Rmarkdown files into vanialla markdown, which then in turn can be converted by pandoc into html document for example. Some of the things that knitr[4] does includes R code execution and assembling the results into markdown.

3.4 Rmarkdown

An rmarkdown R package[5] will convert `.Rmd` files into other format types. Under the hood it will use pandoc[6] to do so. The main function that we are concerned with is `rmarkdown::render()` which will call `knitr::knit()` when required.

3.5 Bookdown

A bookdown R package[7] enhances rmarkdown[8] by enabling multi-page documents e.g books and easy cross-referencing.

3.6 Others

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

- xaringan[9]
- blogdown[10]
- thesisdown[11]

[2] https://pandoc.org/
[3] https://yihui.name/knitr/
[4] https://yihui.name/knitr/
[5] https://github.com/rstudio/rmarkdown
[6] https://pandoc.org/
[7] https://github.com/rstudio/bookdown
[8] https://github.com/rstudio/rmarkdown
[9] https://github.com/yihui/xaringan
[10] https://github.com/rstudio/blogdown
[11] https://github.com/ismayc/thesisdown

3.7 References

- related stackoverflow question[12] a- Presentation slides on Rmarkdown ecosystem[13]

[12] https://stackoverflow.com/questions/40563479/relationship-between-r-markdown-knitr-pandoc-and-bookdown
[13] https://slides.yihui.name/2017-DSM-rmarkdown-Yihui-Xie.html

Chapter 4

The R chunks

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

```
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

4.1 Why Rmarkdown again?

The reason that we are using Rmarkdown[1] is because there is no alternative, really? Really.

Irrespective of the programming language used we'll have to write code and one way or anther you'll be writing notes and comments that will go along with that chunk of code. It makes sense to keep those comments (text) close to the code, inside the same file ideally. That idea isn't novel I and hope that all of you comment your code well. Common commenting characters:

- `#`

[1] https://rmarkdown.rstudio.com/

- `/*  */`
- `//`

But the comments are never seem in the output and can only be accessed by one looking into the source. But wouldn't it be nice if we can see the code, the output and the comments together?

This type approach is supper common and supper useful when you working with some data and you are trying to convey some message to a different person (people) by means of examples and/or illustration of the analysis flow. Not only one can read your notes about the analysis, that are placed exactly next the piece of code or data by the way, but one can also see the code that one wrote for that analysis.

4.2 Embedding R code

RStudio is pretty good it templates our Rmd file a bit. Lets delete all the text past these lines.

````
---
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` bit shortly, but for now let's skip that part and move on to embedding R code into the document.

Any R code has to be inside "special" block - chunk, this one

````
```{r}

```
````

Little `r` there specifies the "engine", which interpreter to use to "understand" the text inside that chunk. Here we are saying use `R` engine (language) to evaluate the code. The list of languages[2] is rather long actually, but for now let's only

[2] https://bookdown.org/yihui/rmarkdown/language-engines.html

focus on R language, otherwise it becomes even harder to explain Rmarkdown. As I've hinted in the introduction section is so ubiquitous that you can't avoid it anymore.

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

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 options[3]. 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 on the internet. Note that all chunk options have a default value, by not specifying the options you are simply using the default option.

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 three chunk options:

[3] https://bookdown.org/yihui/rmarkdown/r-code.html

- `echo` show what has been typed in i.e show the code
- `eval` evaluate or execute that code
- `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. 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.

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)
```

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.

4.3 Challenge: code chunks

3 minutes

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

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

4.4 References

Here is nice cheatsheet[4] that has comprehensive cover of all the options you can pass in.

[4] http://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf

Chapter 5

Git 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 your vertically handing off 70 feet rock

Git[1] 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. Git will help us:

- organise our directory structure
- create "milestones" a.k.a `git commits`
- make apparent which parts of the projects (files) are important
- git will also help us to share our work
- and last but not least git will help us collaborate

Git is a command line tool however you don't have to learn command line just yet. There are a few git clients available[2] - 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.

[1] https://git-scm.com/doc
[2] https://happygitwithr.com/git-client.html

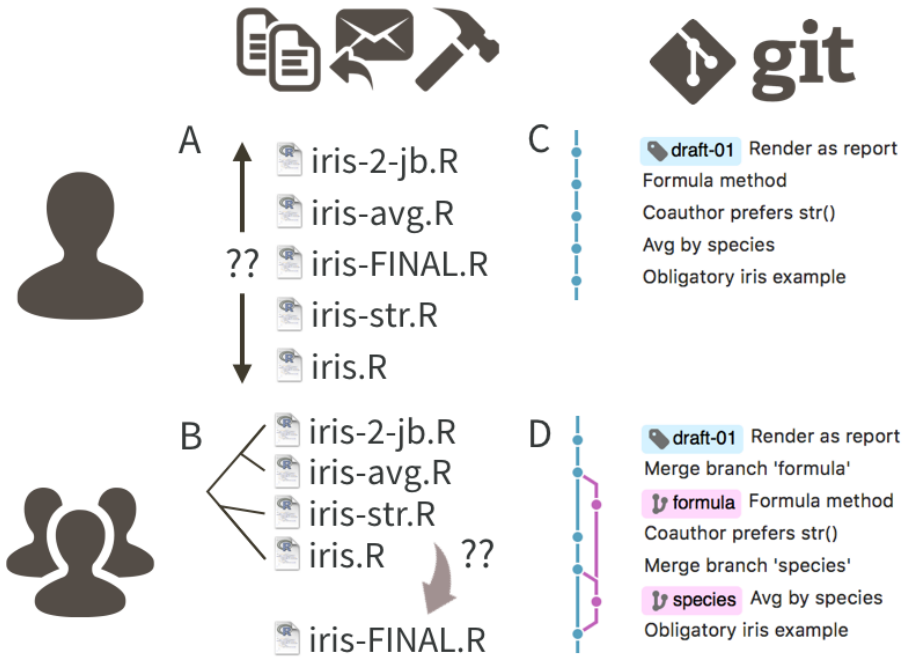


Figure 5.1: This is an example of git version cotrol vs DIY versioning via filesystem

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

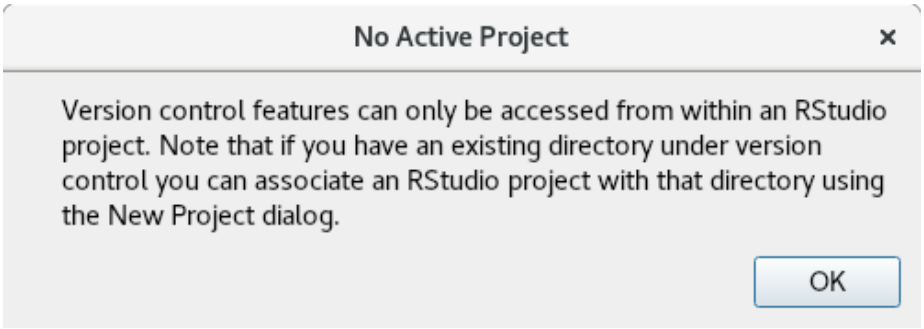
An interesting note about command line git usage noted in Happy Git with R book[3]; One might think that git via cli is "better", however it is more important to get the work done and have it version controlled rather then fight with the cli. Do take the simplest and quickest path to get your work version controlled. No one will care which client you are using in the end.

5.1 git no nos

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

5.2 Starting with git

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



5.2.1 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 organise and ready for future collaboration. We need to let git know our name and email address, which will get stored in configuration file.

[3] https://happygitwithr.com/git-client.html

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.

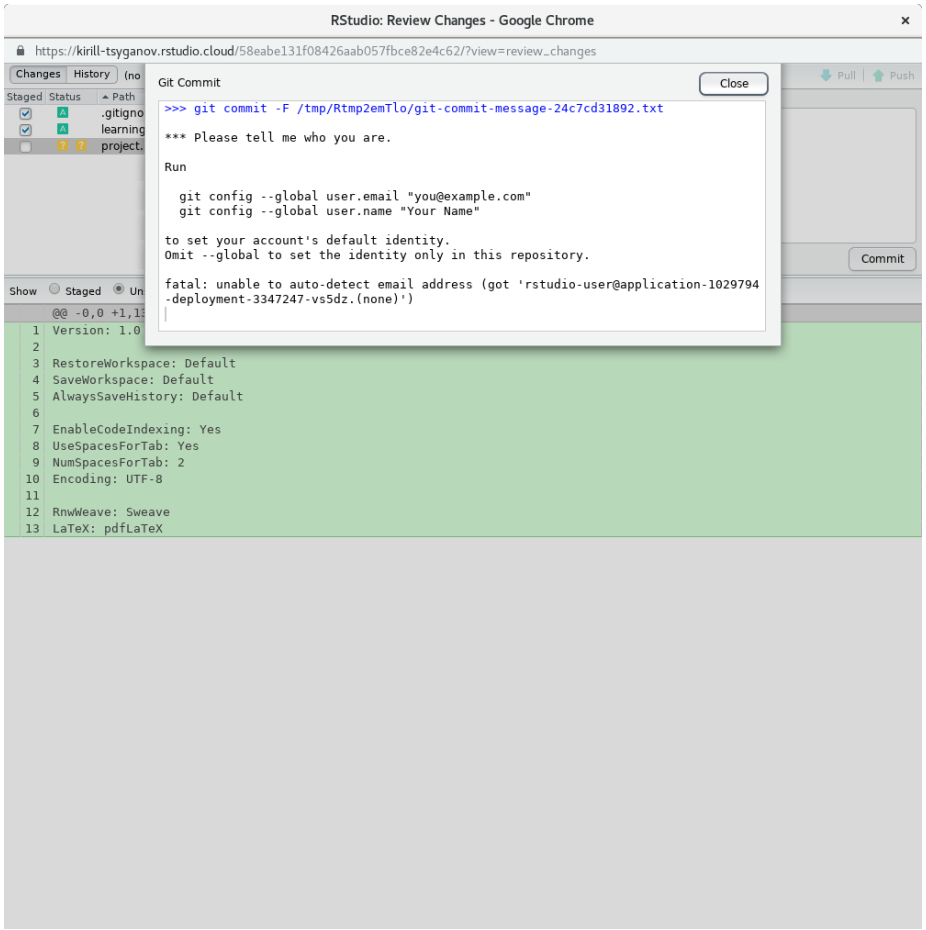
Let's open up a terminal and run a couple of `git` commands

Tools -> Terminal -> New Terminal

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

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

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



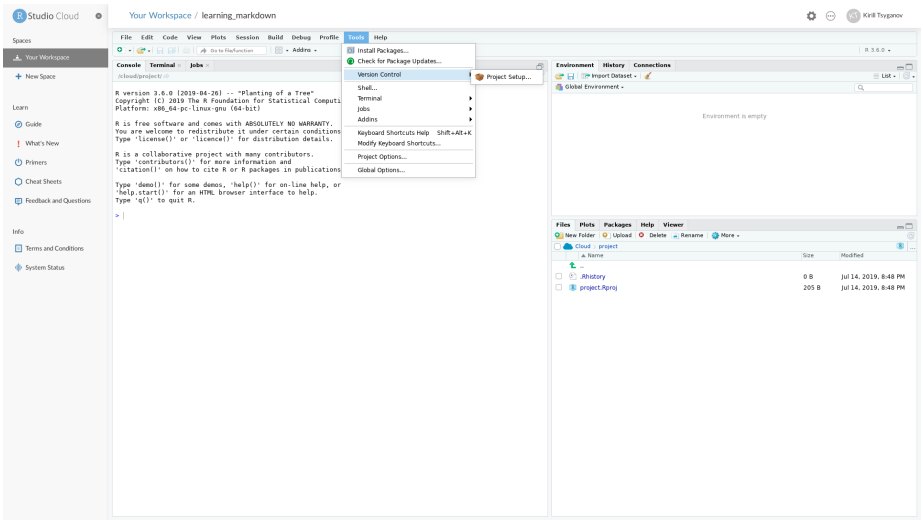
5.2.2 Intiating git repository

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

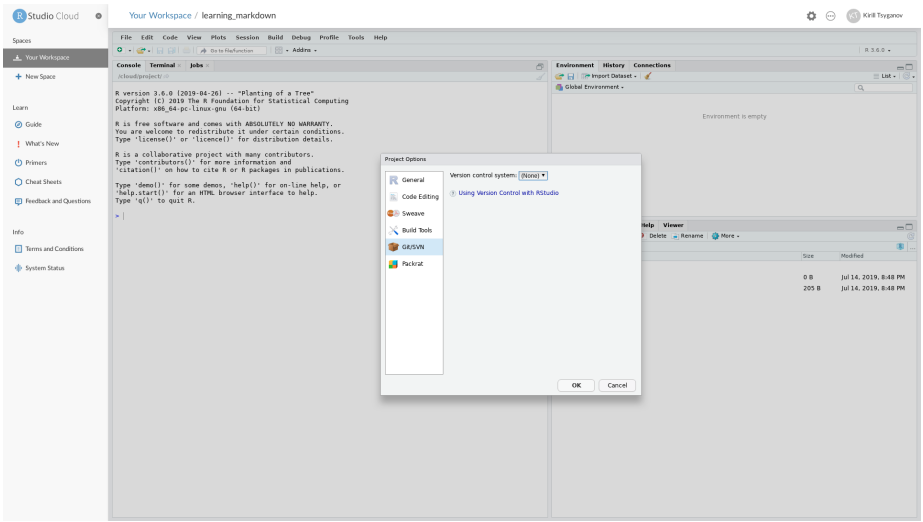
`Rproject folder == Rproject directory == git repository == git repo`

Let's initate git repository

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



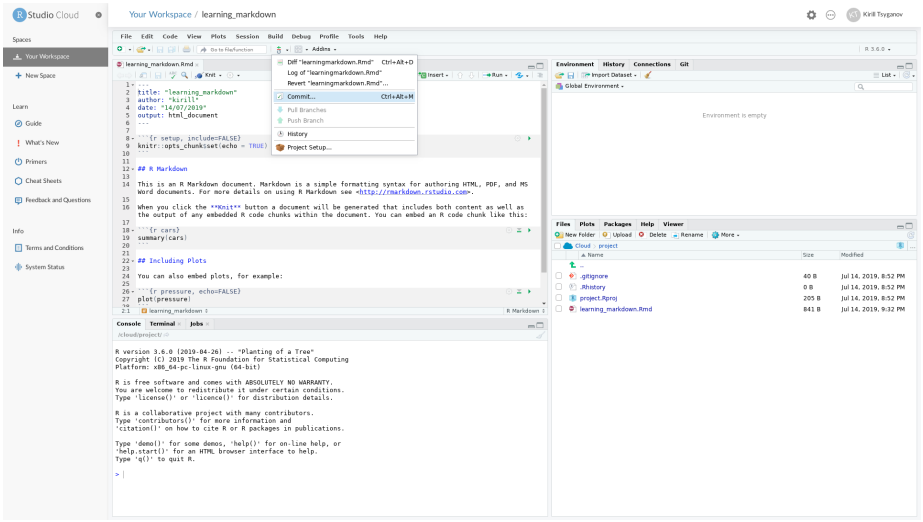
And select from the drop down options "Version control system" Git



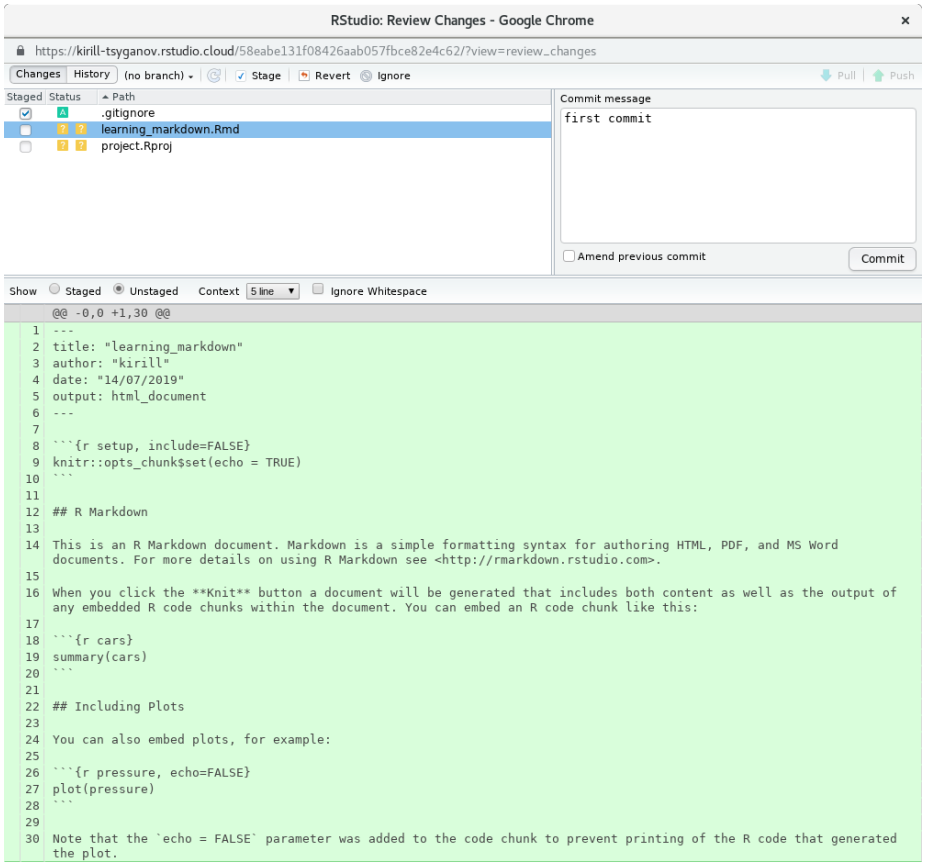
The command line equivalent is navigating to your project directory and running `git init`

5.2.3 First commit

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



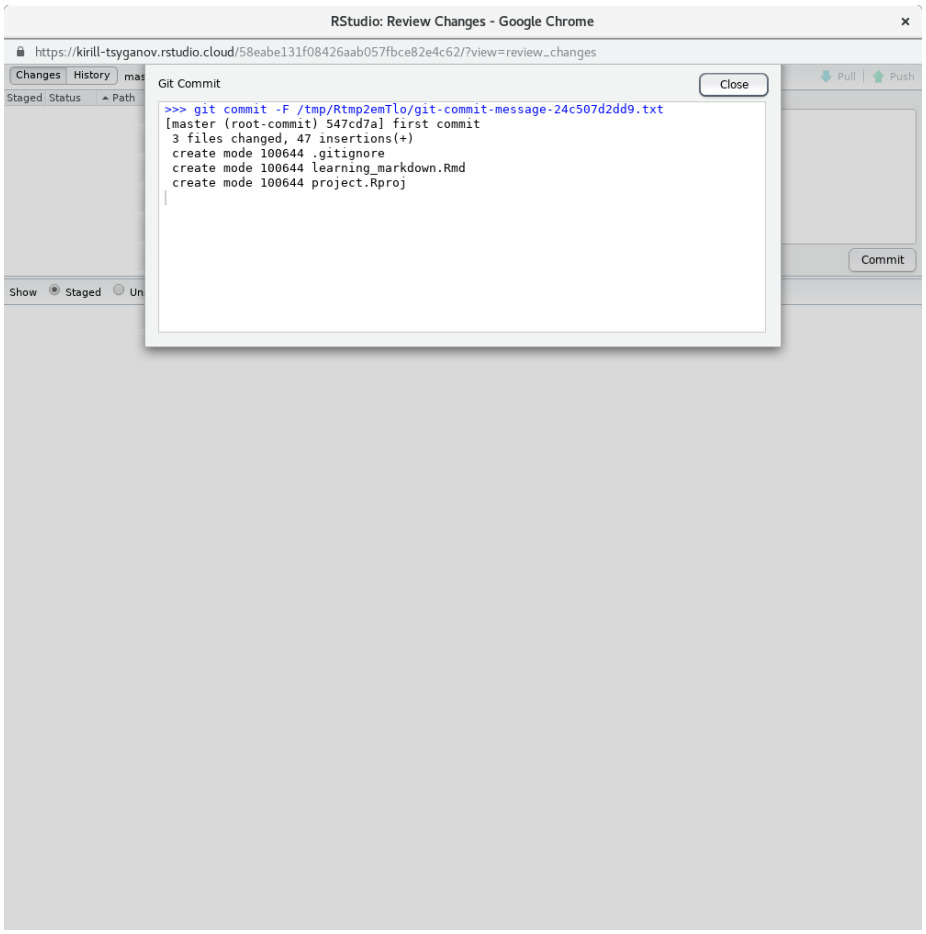
You should see a new window poped up



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

5.3 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[4]

[4]https://peerj.com/preprints/3159/

Chapter 6

The R chunk 2

6.1 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)
```

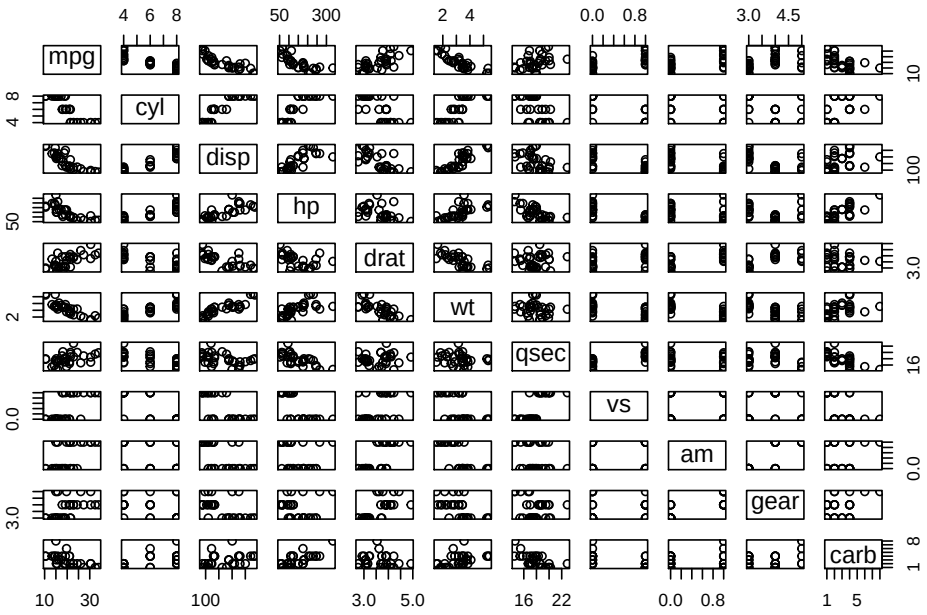
```
##       mpg             cyl             disp             hp       
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0  
##  Mean   :20.09   Mean   :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)
```
````

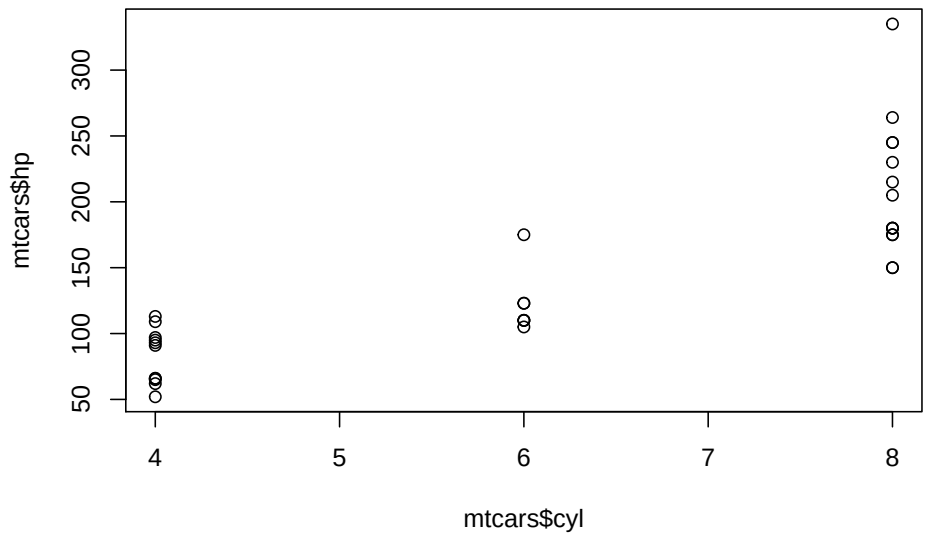
```
plot(mtcars)
```



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)
```
````

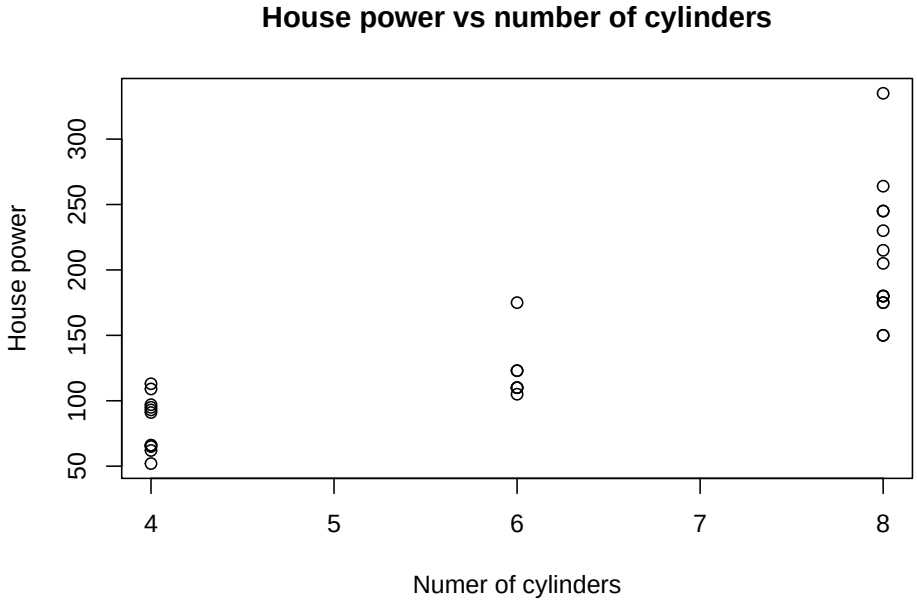
```
plot(mtcars$cyl, mtcars$hp)
```



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

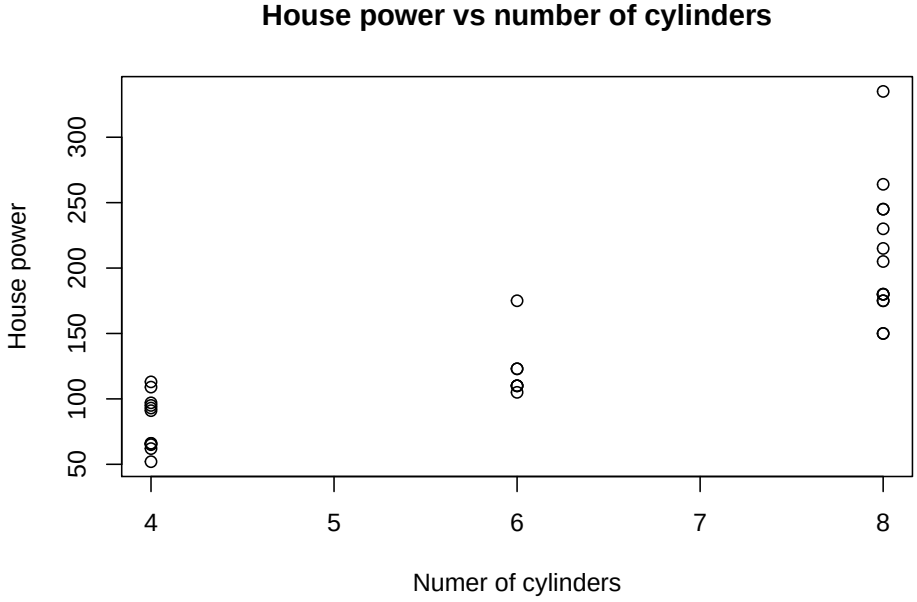
````
```{r}
plot(mtcars$cyl,
           mtcars$hp,
           main='House power vs number of cylinders',
           xlab = 'Numer of cylinders',
           ylab='House power')
```
````

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



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)"`



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

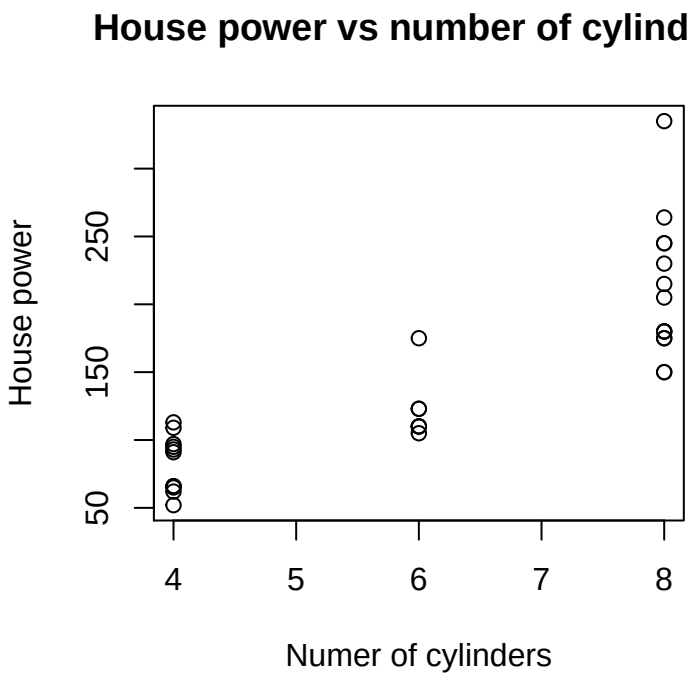
6.2 Figures related chunk options

- `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.

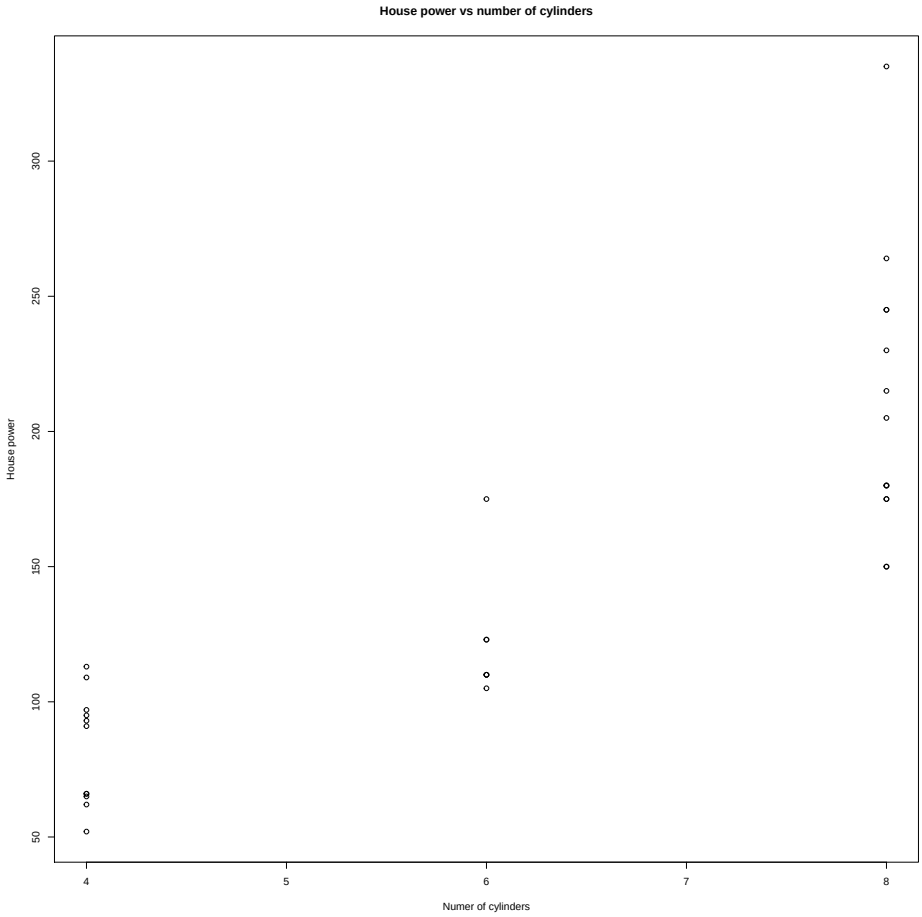
````
```{r fig.height = 4, fig.width = 4}
plot(mtcars$cyl,
           mtcars$hp,
           main='House power vs number of cylinders',
           xlab = 'Numer of cylinders',
           ylab='House power')
```
````

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



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)

6.3 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

6.4 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 *consosle*
- `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

6.4.1 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

6.5 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`

6.6 References

- Section 2.6[1]
- Pimp my RMD[2]

[1] https://bookdown.org/yihui/rmarkdown/r-code.html
[2] https://holtzy.github.io/Pimp-my-rmd/

Chapter 7

Github introduction

7.1

make a diagram where git at the center and arrows out to different places gitlab, github, bitbucket personal server, even the same comoputer but different "bare" folder (this isnt "safe" nor accessible by others, but totaly possible)

briefly talk about github things like PR, gitissues, staring and watching 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

mention at some point README file as a general means of "silent" communication then README.md and index.html as "special" files. README.md is the front page - apperance for your project. One can have multiple README.md files in subdirectories if necessary.

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

Chapter 8

YAML header introduction

8.1 YAML header

At the very top of your `.Rmd` file you can, optionaly, include YAML block. In this block you can fine turn 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"
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` becames 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"
output:
  html_document:
    toc: true
    toc_depth: 4
    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.

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.

8.2 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 complied 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, expcept `##` is now used to mark new slide.

8.3 Extras

This is mainly to talk about Rnotebook[1] 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 (defualt 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[2] meant to have better error handling, sends one line at a time for execution, compare to all lines for Rmarkdown document

[1] https://rmarkdown.rstudio.com/r_notebooks.html
[2] https://rmarkdown.rstudio.com/r_notebooks.html

Chapter 9

Bibliographies

Chapter 10

Work in progress

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
---
```

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 defition, 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/

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.