An introduction to NAPdown

A Team

2021-10-21

Contents

Chapter 1

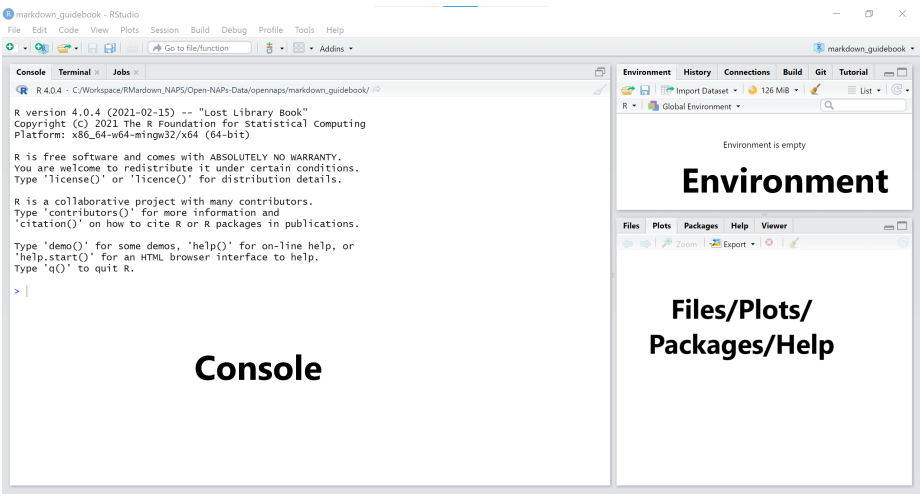
Prerequisites

1. You have installed R (https://cran.microsoft.com/)
2. You have installed RStudio (https://www.rstudio.com/products/rstudio/download/)

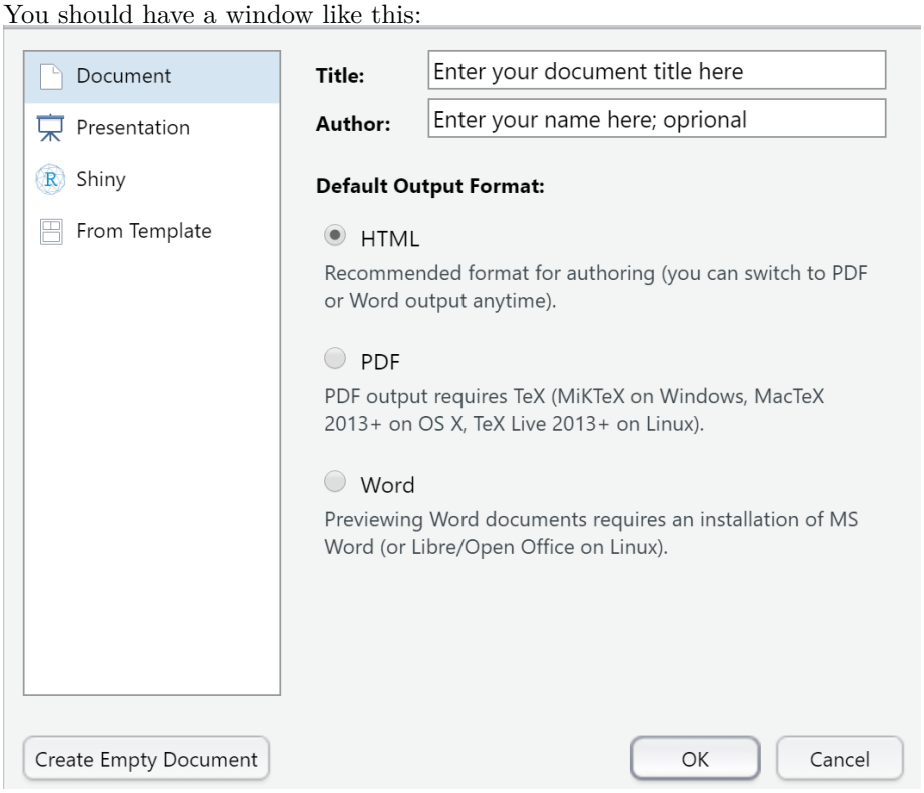
The installation prompts for installing both r and rstudio are straight-forward. However, should you require a step-by-step guide, you may refer to this beginners guide

1.1 Opening a markdown file

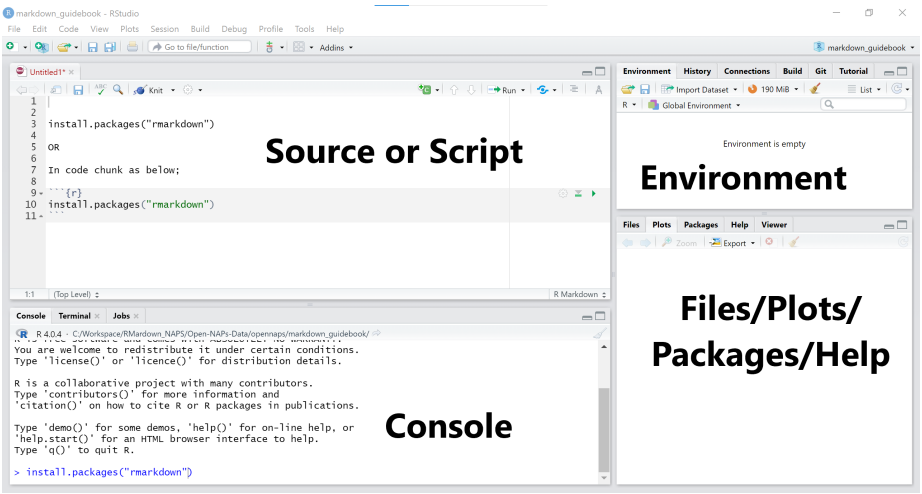
1. Launch your r/studio app. Your rstudio workspace should look more or less similar to this:



If your rstudio opens up with 4 windows instead of 3, skip next step.
2. From the main menu, go to File-»New File-»R Markdown.
You should have a window like this:



3. Fill in the details for your document title and author. Leave the rest as default.
4. Click Ok.
This opens up your 4th rstudio window with an rmarkdown script.
Now your app area should look more or less like this;



The arrangement may however be different. If so, do not worry, it's just semantics.

As you might have seen, you may open other scripts such as R, python, SQL etc in similar manner.

1.1.1 Rstudio windows explained:

- **The Source**
  This is your scripting area.
  You may write, edit and run your code from here.

- **The Console**
  All processing in r happens here. You may this is the r kitchen. When you run your code, this is where it is evaluated.
  You may also use the console to write and execute your code. However, you can not save your code, nor can you edit it (you will have to rewrite it). Thus, it is always advisable to write your code in the script/source area.

- **The Environment**
  This panel holds your data objects and their metadata. As you create or add new objects into your project, they will appear in this window. You may view and remove objects. It also contains the 'History' tab from which you may view all your r processing history.

- **The Files/Plots/Packages/Help Panel**
  This panel provides a shortcut/alternative to different functions such as to

access your project files, to view plots from your code evaluations, install packages and access the Help resource. You may add/remove items here but we will learn that much later. For now we leave it as is.

1.2 Installing packages

Options:

1. From the console;
After the greater than (>) sign, write command `install.packages("package name")` and hit enter on your keyboard.
See example;

```
> install.packages("rmarkdown")
```

Figure 1.1: install packages from console

2. In script window Inside your script write command `install.packages("package name")`
Click on 'Run' (Run button is on the top right of your source panel).
Alternatively, if your write your command in a code chunk, you may hit the 'Play' button on far right of your code chunk. See example;

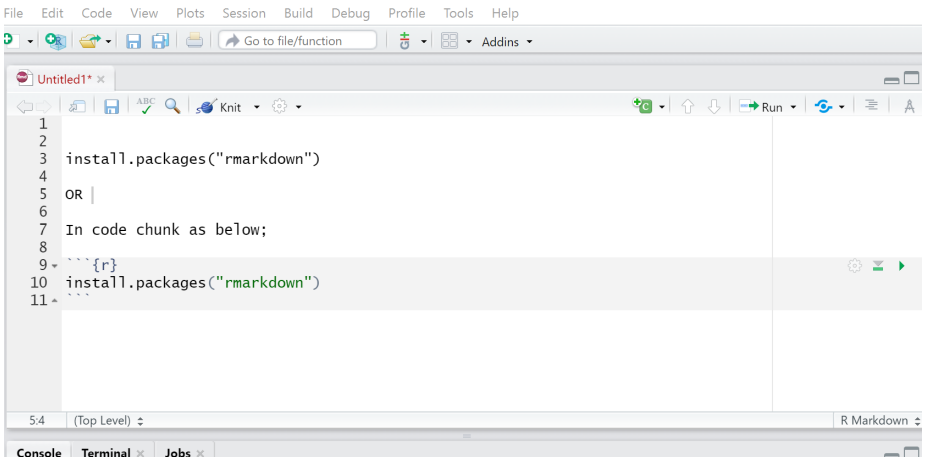


Figure 1.2: install packages from script wnindow

*We see more about code chunks in the next chapter.

3. From the menu bar From the menu bar, go the the tab 'Tools' and select 'Install Packages'.
You should see a window like this one:

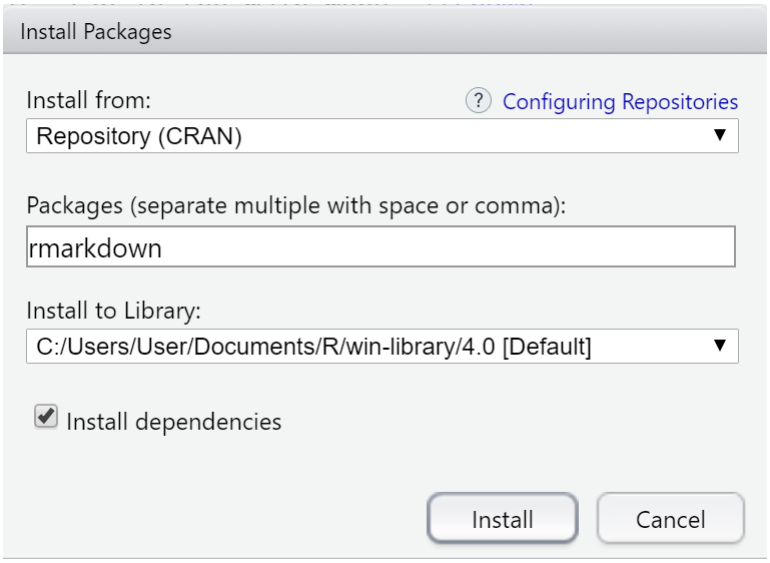


Figure 1.3: install packages from menu bar

In the 'Install from' field, select 'Repository (CRAN)', then enter the name(s) of packages to be installed.
Leave everything else as default. Click Install.

4. From Files/Plots/Packages/Help Panel
Click on the 'Packages' tab. There are 2 tabs under this, Install & Update.
Click on Install.

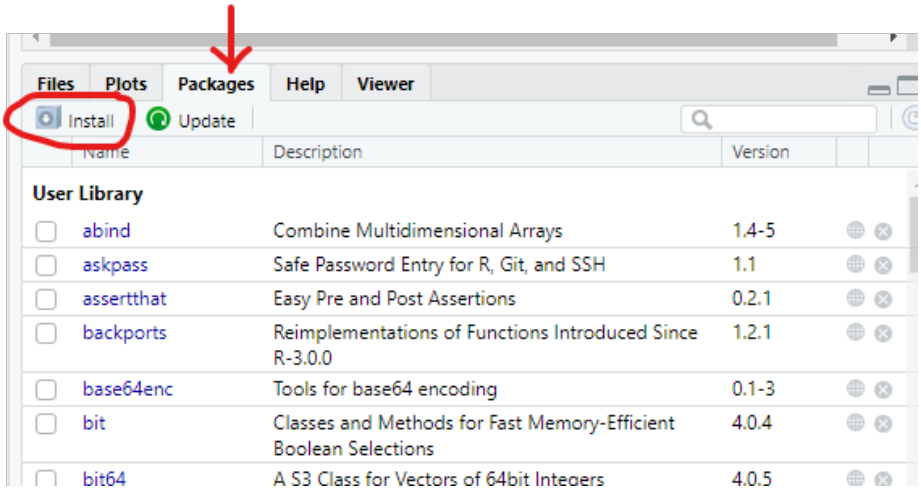


Figure 1.4: install packages

If the 'Install from' is not set to 'Repository (CRAN)', set it so, and enter name(s) of packages to be installed.
Leave everything else as default. Click Install.

Note: *Packages can only be installed once. To mean, if you close your r/rstudio app and come back to it the next day/session, you do not need to install the packages you already installed in your previous session. You will only need to load their libraries (see next section)*

1.2.1 Removing/un-installing packages

To remove a package use command `remove.packages("package name")`

1.3 Loading Libraries

To be able to make use of the packages installed, you need to call their libraries. Most libraries take the name of the package. For instance, for the package `rmakdown`, the respective library is `rmarkdown`.To call the rmarkdown library, use command `library(rmarkdown)`. So in general to load/call libraries use command `library(name of library)`.
You may load libraries in the console or in your script.

Note: *Unlike packages,library functions expire when you close a project or end a session. Therefore, each time you open an r session, you have to load/call relevant libraries*

Chapter 2

Getting started

2.1 About RMarkdown

RMarkdown is a file document that allows you to write, save and execute code, as well as text and figures to help generate reproducible reports that can be shared in several formats. The file extension is .rmd.
A markdown file has three main sections;

1. **YAML header**
   This is where your document metadata go to e.g your document title, author, date, output file type etc. This parameters are set when opening a new .rmd file. And more can be set after. This section MUST always be at the beginning of your document and between a set of three dashes i.e three dashes before section and three dashes after section.

2. **Text**
   Your narration/prose in markdown format. More details about formatting in subsequent sections.

3. **Code chunk(s)**
   They start with "'{r}

   and end with "'

To start using RMarkdown, install the following base packages using any of the methods described in the Installing packages section

> ('base64enc', 'digest', 'evaluate', 'glue', 'highr', 'htmltools', 'jquerylib', 'jsonlite', 'knitr', 'magrittr', 'markdown', 'mime', 'rmarkdown', 'stringi', 'stringr', 'tinytex', 'xfun', 'yaml')

2.2 Editing in markdown

2.2.1 Headers

To create headers for your reports/document e.g. chapters, sub-chapters and so on; use the hash sign '#' in front of the title. Sequentially increase the number of "#' signs to denote subsequent header levels.
Insert blank line before each header (except in the beginning of document).

```
# Header level 1
## Header level 2
### Header level 3
#### Header level 4
```

Figure 2.1: headers

Note: For the purpose of this example, I have added a space before each line so that r doesn't create the headers. Remove the space to create the headers

2.2.2 Bold and italic text

To create emphasis in your markdown texts, use an asterisk before and after text *to italicize* or double asterisk **to make your text bold**.
Alternatively, you may use single underscore *for italics* or double underscore **for bold**.

```
*italic*   **bold**
_italic_   __bold__
```

Figure 2.2: text emphasis

2.2.3 Create Lists

2.2.3.1 Ordered list

Use numbers to order your list items. and a plus sign '+' to create sub-items on your list.

```
1. List item 1
2. List item 2
  + Sub-item 1
  + Sub-item 2
+ Sub-item 3
3. List item 3
```

Figure 2.3: ordered list

2.2.3.2 Unordered list

Use an asterisk '*' before list item to create an unordered list.
Use a plus '+' sign for sub-items.
Use tab command or two spaces on your keyboard to indent the list items

```
* List item
* List item
   + Sub-item
     + Sub-item
       + Sub-sub item
* List item
```

Figure 2.4: unodered list

2.2.4 Manual line breaks

Use two or more spaces at the end of a line
to insert a line break

2.2.5 Insert links

You may insert a link using the plain http address such as https://rmarkdown.rstudio.com/ or insert it as a linked phrase using square brackets and parenthesis such as in screenshot

```
[our link phrase goes here](https://rmarkdown.rstudio.com/).
```

to produce this:
our link phrase goes here.

2.2.6 Insert figures/images

To insert images to our document, we use the same syntax as links, but start with an exclamation mark'!' before syntax. For an image from a url use; `![text to accompany your image e.g a caption](your https link)`
or for an image file in your local directory use, `![your image text](path to local image file)`.
For instance, I downloaded the UN Climate logo and saved it as a .jpg in my working directory as exact name `unfccc_logo.jpg`
The following syntax will insert the logo into my document:

```
![UN Climate logo](unfccc_logo.jpg)
```

Figure 2.5: UN Climate logo

When inserting images from local file, it is strongly recommended to have the image in your working directory.

2.2.7 Insert block quotes

To insert a block quote within your text, use the greater than sign '>' in the beginning of quote. The quote must begin on a new line, and remember to insert blank line before and after. For instance the screenshot below;

```
> This exercise may seem complicated at first, but trust me, it is not.
> You will agree with me sooner than later :)
```

Figure 2.6: block quote

produces this:

> This exercise may seem complicated at first, but trust me, it is not.
> You will agree with me sooner than later :)

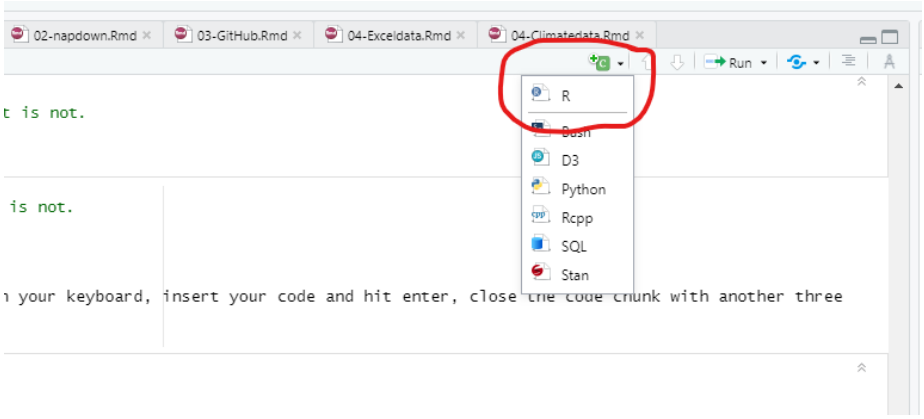
2.2.8 Insert code chunks

To write your code use open code chunk with three backticks and the curly brackets {insert your code language}, hit enter on your keyboard, insert your code and hit enter, close the code chunk with another three backticks.

```
```{r}
# your code here
```
```

Figure 2.7: insert code chunk

Alternatively, you may do it from top right corner of your script/source window as shown below. Click on the green +c icon and select R from the drop down list.



Code chunks for other languages may be added from this list as well.

2.2.9 Create tables

2.2.9.1 Option 1

Create table headers with dashed lines below the header title. Separate headers with tab or space between the headers and corresponding dashed lines.
Type in row values below the dashed lines. The row value length may exceed the dashed line length but MUST not extend into the next header's dashed line.
Column alignment is based on the position of the header/column title relative to the dashed line below it.
To insert a caption or alt text to your table use, full colon ':' followed by your caption text at the end of the table.
Alternatively, use 'Table: your caption or text'. The following syntax (screenshot) produces the ensuing table.

```
  Header 1    Header 2    Header 3        Header 4
  ---------  ---------   ----------      --------
  12343       895         0.5867891011    1
  Name        Rank        score           remark
  Type        3           TRUE            12.1
  Left align  Right align Center          Default

: you may insert table caption here
Table: Alternative caption option
```

Figure 2.8: manual table

Table 2.1: you may insert table caption here
Table: Alternative caption option

Header 1	Header 2	Header 3	Header 4
12343	895	0.5867891011	1
Name	Rank	score	remark
Type	3	TRUE	12.1
Left align	Right align	Center	Default

2.2.9.2 Option 2

You may also create simple tables using a `knitr` function called `kable`.
The code below tells r that we want to create a data set with 3 columns, X, Y & Z, assigning them the values enclosed in the letter c. The letter c used together

with brackets indicates a list of elements. Thus, in the example below, we tell r that our column X, will contain a list of 4 elements i.e 20, 30, 10 & 50.
Then we tell r to create the data set by combining all the columns X, Y & Z into a data frame. Finally, we call the function `kable` and enter the data we created. This function converts our data frame into a table format.
Optionally, you may add a caption to the table and specify cell alignment.

```
```{r kable, fig.cap= "My first simple table with kable"}
X<-c(20,30,10, 50)    # create variable X
Y<-c(1.4, 4.3,5.9,2.7)    # create variable Y
Z<-c("yes","no","true","false")   # create variable Z
mydata<-data.frame(X,Y,Z) # combine variables into table format
knitr::kable(mydata, caption = "My first simple table with kable", align = 'c') # plot table with kable
```
```

Figure 2.9: kable table

Table 2.2: My first simple table with kable

X	Y	Z
20	1.4	yes
30	4.3	no
10	5.9	true
50	2.7	false

Note that our code chunk is labelled 'kable'. You can label code chunks as below

```
```{r chunkname goes here}

```
```

Figure 2.10: labled chunk

This will be useful for cross-referencing as we'll see in the Chapter References section.

2.2.10 Page breaks

Use three or more asterisks or dashes to insert a page break.

```
----   or

***
```

Figure 2.11: page break

Remember to add a blank line before the asterisks or dashes

2.2.11 Process a markdown document to desired output

To create the desired output file document from the markdown format, use the function `render ("your .rmd file name)`.
Alternatively,and most commonly used, is the `Knit` button from the markdown script environment. The button is a blue ball of yarn around a crotchet, and is labeled 'Knit' When a document is rendered, rmarkdown saves the results/output file into your working directory, giving it the same name as your .rmd file, but with relevant extension (e.g. as html if output type was set to html)

2.2.12 References

2.2.12.1 Citations/Bibliography

To add citations to our document, we need to add our reference document details to a text file saved in the .bib format e.g myrefences.bib. We also need to add this file to our bibliography parameter in the yaml section of the index.rmd file. The references for this exercise are saved in the file 'book.bib'.
To add your reference documents to the .bib file, use the BibTeX citation style i.e

```
> @documentype{documentname,
  title={document title},
  author={author name(s)},
  year={publication year},
  publisher={publisher name},
  url={document url}
}
```

Figure 2.12: citations

Sites such as Google Scholar have ready to use/ formatted bibliography styles such as BibTeX, EndNote etc. You may copy-paste the BibTeX text into your .bib file in r.

You may have as many documents listed in the .bib file but rmarkdown will only include those that have been referenced within your markdown document. To reference documents in your narrative and thus, have them included in the reference section, use the format `[@documentname]`.
The references will appear at the end of the chapter where they are referenced, as well as in the overall 'References' section.

2.2.12.2 Chapter References

To reference chapter, type in the chapter title inside square brackets. For instance, this command references our second chapter named NAPdown.

```
[NAPdown]
```

Figure 2.13: chapter ref

The same applies to section headers. Just type [section header name].

2.2.12.3 Table & Figure References

To reference figures use syntax '@ref(fig:code chunk label)' or'@ref(tab:code chunk label)' for tables. For instance, we created our kable table in the code chunk named kable. To reference this table anywhere in our document, just type in command

```
\@ref(tab:kable)
```

Figure 2.14: Ref table

This will insert a reference link to the kable table.

2.2.13 Pandoc & Knitr

`Pandoc` is a universal document converter designed to convert thousands of markup languages. So when we create our document in markdown and want to output it as a pdf, pandoc does the work.

`Knitr` on the other hand, is an r package that enables the integration of yaml, text and code evaluations into an output document. `Knitr` contains the `Knit` function through which we render our rmarkdown documents to our desired output format. When you render a document in rmarkdown (or call the knit function), the rmarkdown document is converted to a basic markdown language (.md) which is then converted by pandoc to say html, pdf, word, etc as per user specifications. `Knitr` and `pandoc` come in bundled with rmarkdown, and thus, there is no need to install them separately.
However, should you need to install pandoc as standalone, you may do so from the Pandoc homepage. In this regard, it is important to note that in as much as standalone installations may provide much higher versions of the software than what is already bundled in r, they are often not streamlined for use in r, and may thus cause some compatibility issues.

Chapter 3

NAPdown

3.1 Setting up an eNAP

We will use the package `bookdown` to generate NAP document in a book format. Journal articles or reports can be produced in the same way. We will also use the package `tinytex` to build pdf format of the book document.
1. First, launch the rstudio app in your pc. 2. Install the bookdown and tinytex packages in case you have not installed them. Use any of the methods shown in the Installing packages section.
3. Then from the menu bar go to `File->>New Project->>New Directory->>Book Project using bookdown`.
A new window appears

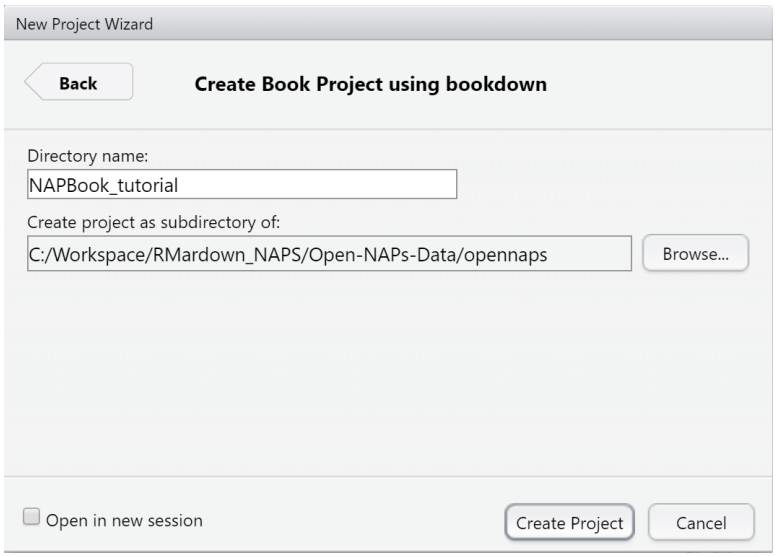
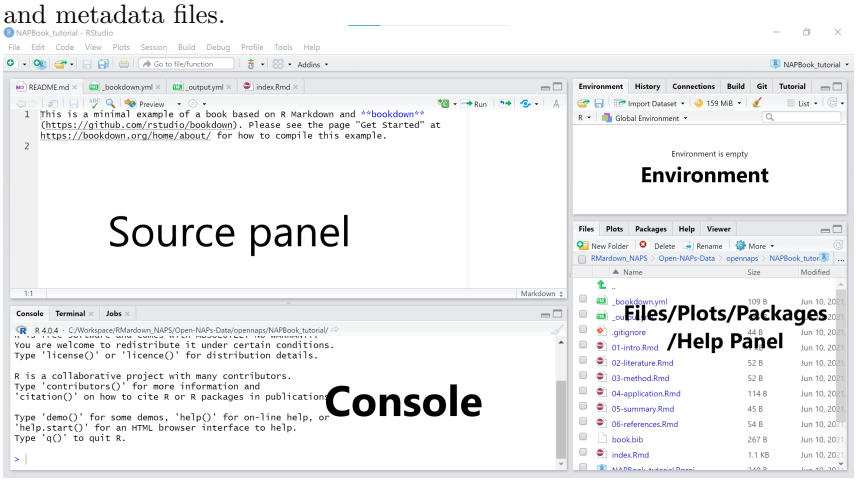


Figure 3.1: Create Bookdown project…

4. Give your bookdown directory a name 'NAPBook_tutorial'. This is the name of a new folder that will be created to store your bookdwon project files.

5. In the 'Create project as subdirectory of' field, use the browse button to navigate to where you want to save your bookdown project folder.

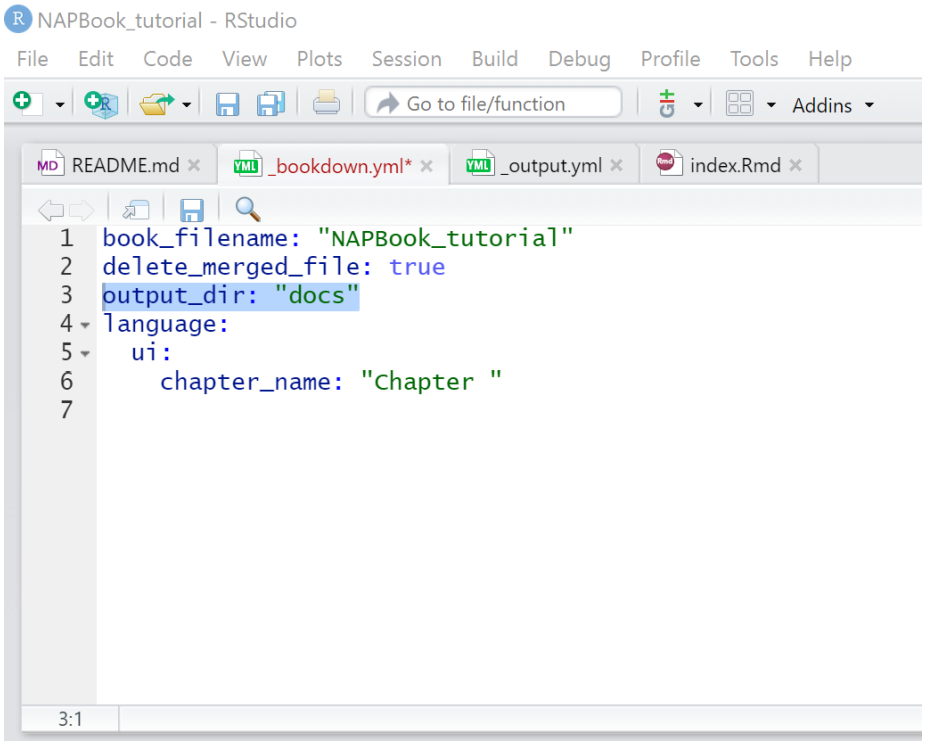
6. After you select the directory location, you are returned to the 'Create Book project…' window. Click 'create project'.
A new project session opens up, with skeleton chapters and other sections and metadata files.



Your project window might appear with panel arrangements different from mine/in screenshot. Your project files can be accessed from the Files tab in bottom right of the project window.
Each chapter is compiled from a single .rmd file, and always starts with a first level header sign '#'.
From the files pane, you can open any chapter, and edit it to your liking.
Lets take a look at the '_bookdown.yml' file.

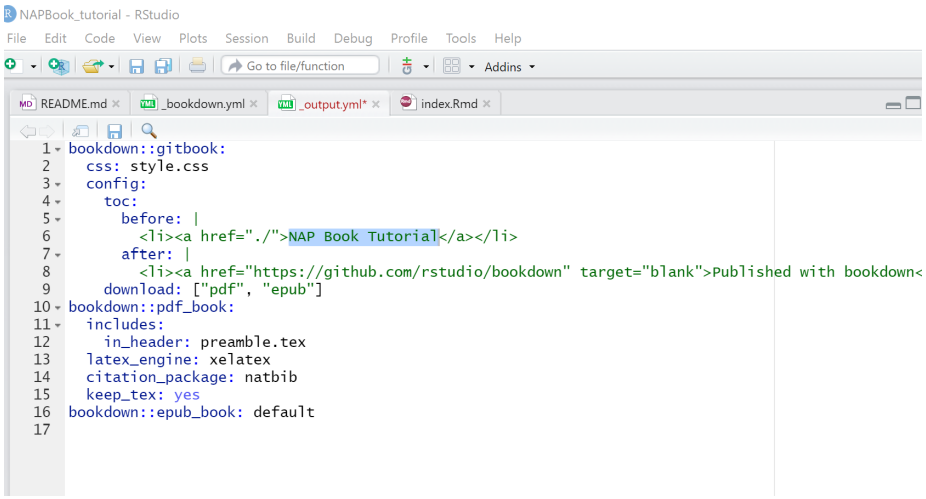
7. From the 'Files' tab, click on the _bookdown.yml' file.
   The file open up in your script/source pane. Notice, the book_filename takes the name of the folder we created when opening the project.

8. Let's add the line `output_dir: "docs"` to this file. See screenshot below;



This command creates an additional folder called 'docs' and specifies that we want to store our outputs here. This folder is necessary for creating webpages from our book project.
9. Save changes. Use the 'Save' icon on the script window or Ctrl + S on your keyboard.
10. Next, click to open the '_output.yml' file.
11. Let's edit the line that contains the title to our book. Change the text 'A Minimal Book ...' to 'NAP Book Tutorial'.

Should look like this;



Ignore everything else for now :)

12. Save.
13. Scroll down your files to find the 'index.rmd' file. Click to open.
14. Edit the title, author & description fields in the yaml header section.
15. Delete all text, except the last code chunk.
16. Add a chapter named 'Preliminaries' before the code chunk.

Chapters are created using the first level header #.

17. Add sections to your chapter using second level header ##.

Your script should look more or less similar to this;

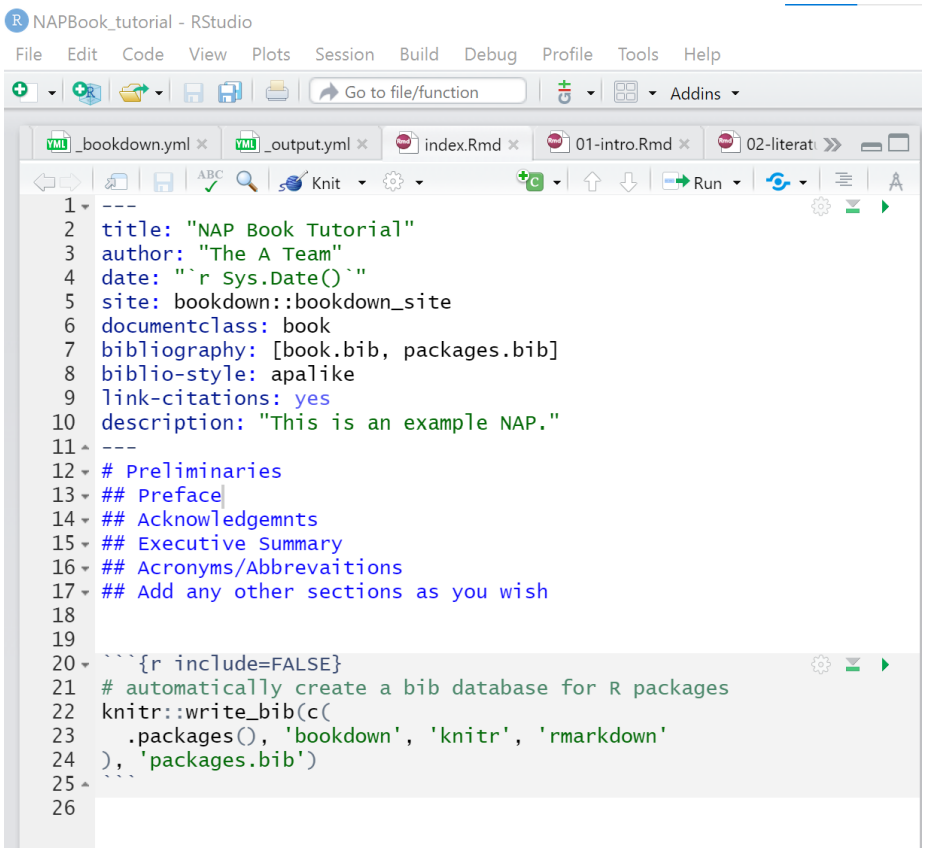


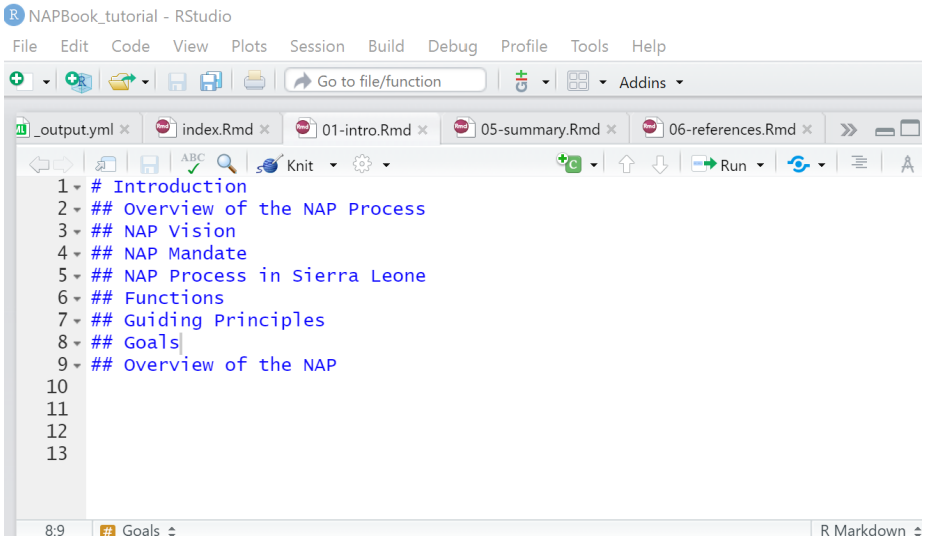
Figure 3.2: edit book title and author

18. Next, open the 'README.md' file and edit the text to this 'This is an example NAP designed to help you get started with creating your NAPs with bookdown.'. Or add a description of your choice.
19. Save your work.

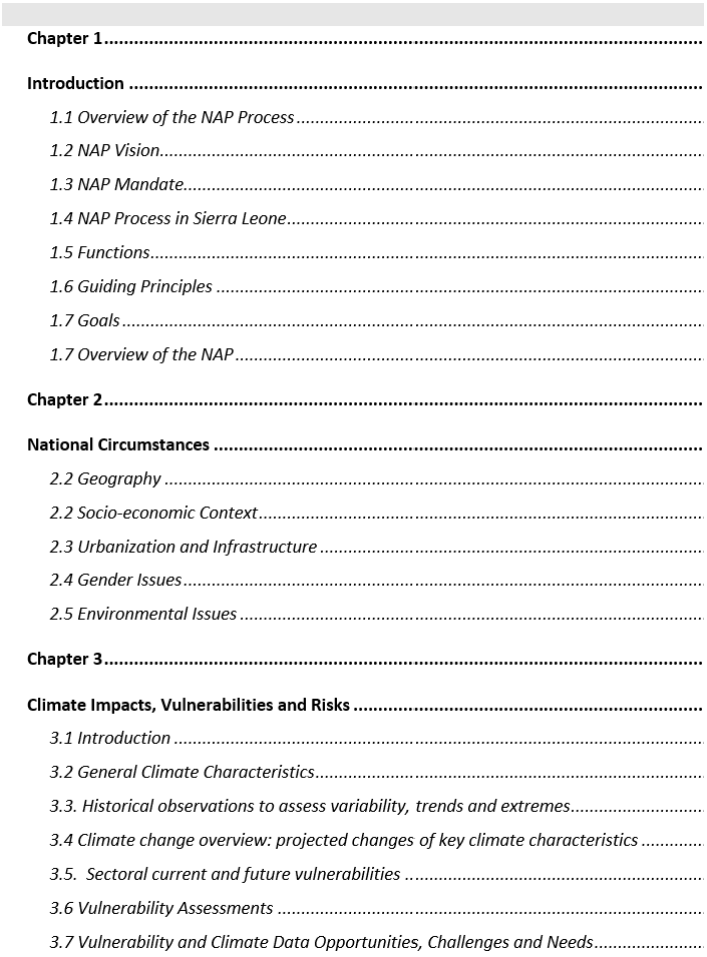
20. Open the '01-Intro.rmd' file.

21. Type in the chapter title 'Introduction'

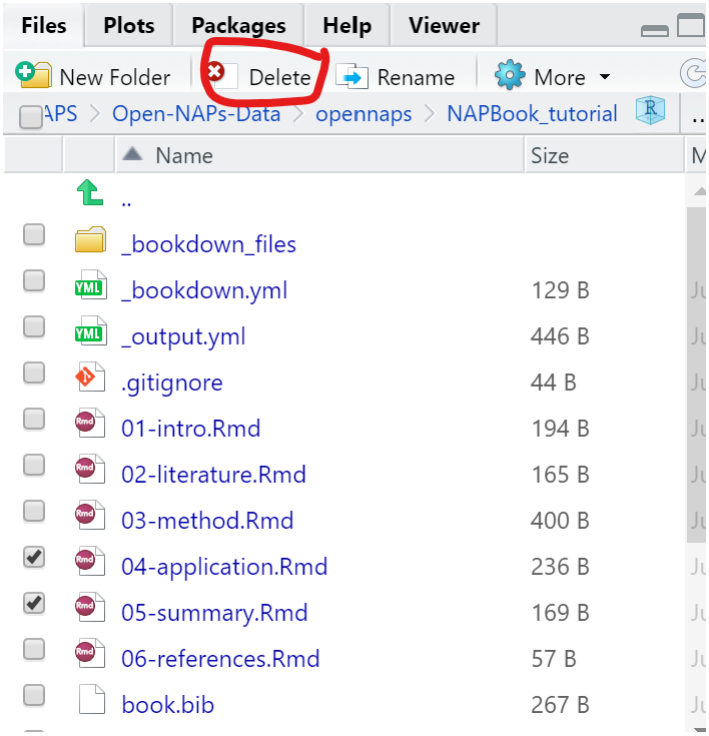
22. Type in sub-chapters with the names 'Overview of the NAP Process, NAP Vision, NAP Mandate, NAP Process in Sierra Leone, Functions, Guiding Principles, Goals, Overview of the NAP'
    Your script should resemble this;



This text has been borrowed from the Sierra Leone NAP. Use the text from the screenshot below to edit the next two chapters (.rmd files). i.e '02-literature.rmd & 03-methods.rmd'.



For the interest of time, let's delete chapters 4 and 5 from our project files. 23. From the files tab (bottom right), click on the check-boxes to the left of the application and summary .rmd files to select them. 24. Click delete



25. Save project.
26. From the 'Environment' window, click on the tab 'Build'.
27. On the new window that opens, click the hammer/Build Book button to start rendering the book.

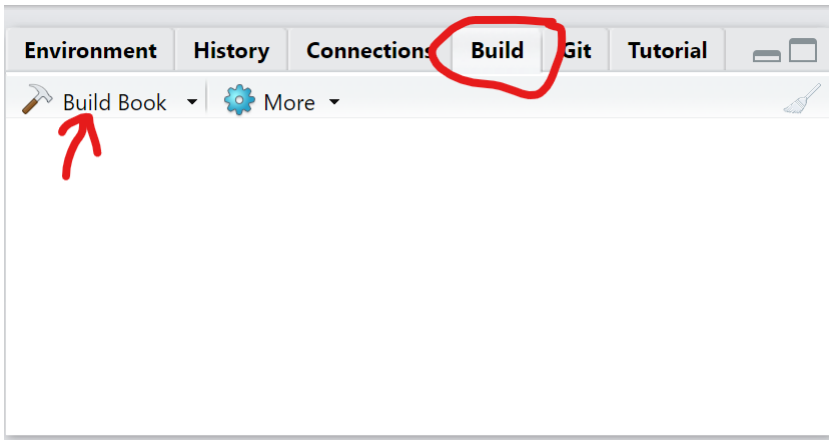
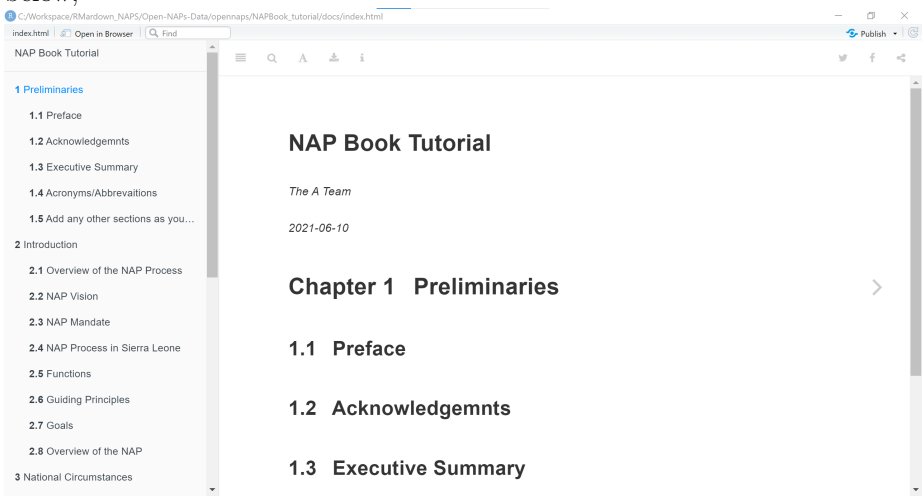


Figure 3.3: Build book

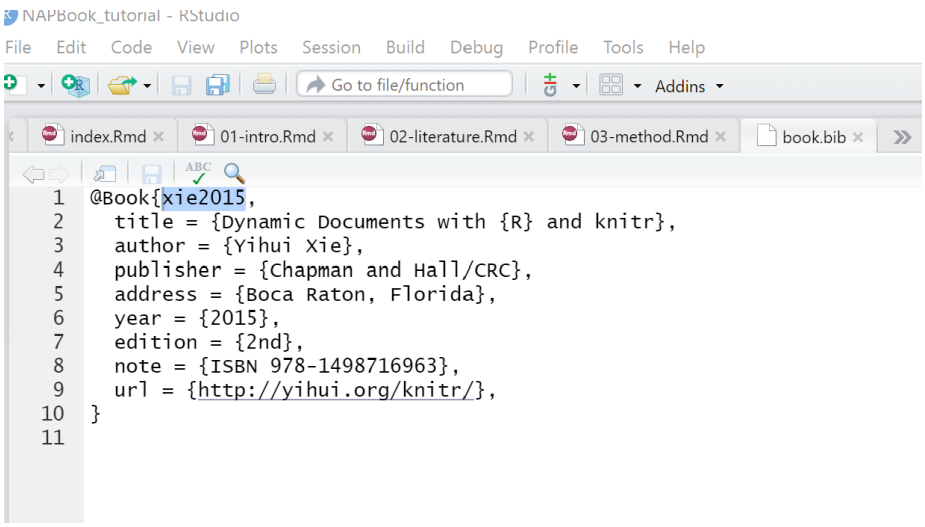
This process may take a few minutes. You can monitor the progress in the Environment window.

When the process is done, your book should open up in an html window as below;



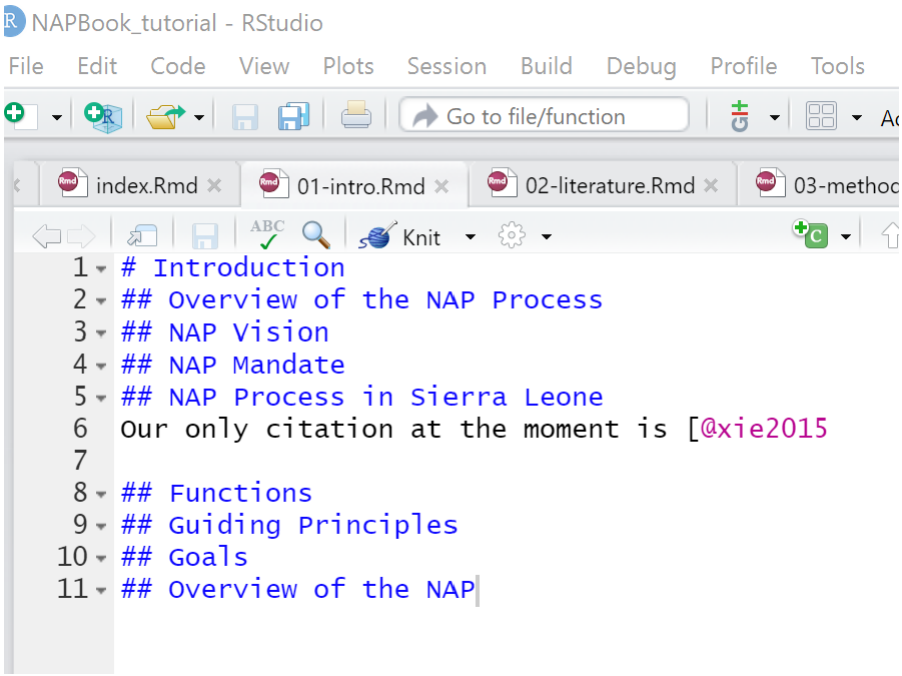
28. Scroll down your chapters to find the 'References' chapter. Oops! No reference! Don't sweat.

29. Back to our project, from the Files tab in bottom right window, open the file named 'book.bib'. This is the text file holding our citations.Currently, only one. You will notice that the document type is a book, and the the book name is 'xie2015'.



30. To cite this book in our NAP book, open square brackets and type the '@' sign followed by the book name. i.e [Xie, 2015].

31. Let's use this citation say in the Introduction chapter. Open the chapter, and insert the text 'Our only citation at the moment is [Xie, 2015]'. See example:



32. Run the 'Build Book' command again.
33. Now you should see that your book has been added to the 'References' section, both in the chapter in which you cited it as well as in the overall References Chapter.

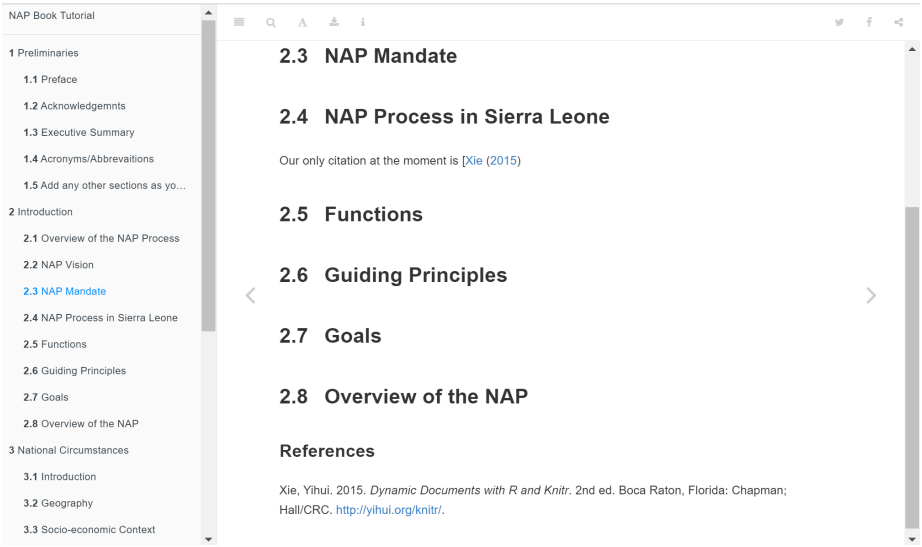


Figure 3.4: citation added to ref

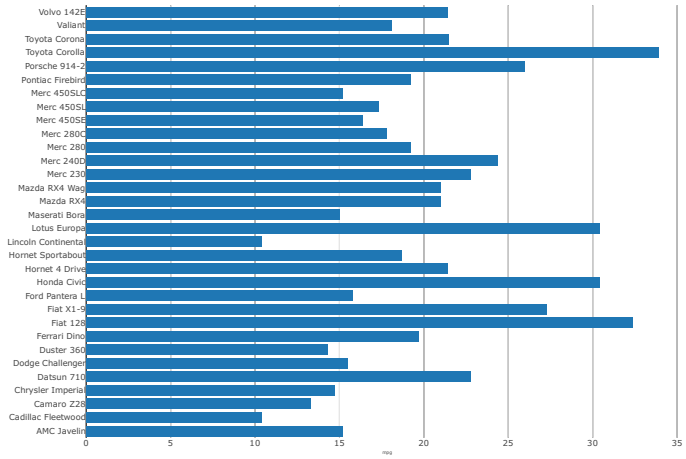
Congratulations!!!

3.2 Integrating code evaluations within your text/ chapters

To integrate a code (+ its evaluation), open a code chunk at your desired section or paragraph and write your code as below. Please refer to the section on Insert code chunks if need be.

For instance in the code below we use one of the in-built R datasets named `mtcars` to plot a bar chart of Car 'Model' against 'mpg' (miles per gallon). When this code is evaluated, the bar chart will be plotted within your document as specified. You may enter text before or after the plot.

```
plotly::plot_ly(mtcars,type = 'bar',
                y=row.names(mtcars), x=~mpg)%>%
  plotly::layout(yaxis=list(tickfont=list(size=26)),
                 xaxis=list(tickfont=list(size=26)))
```

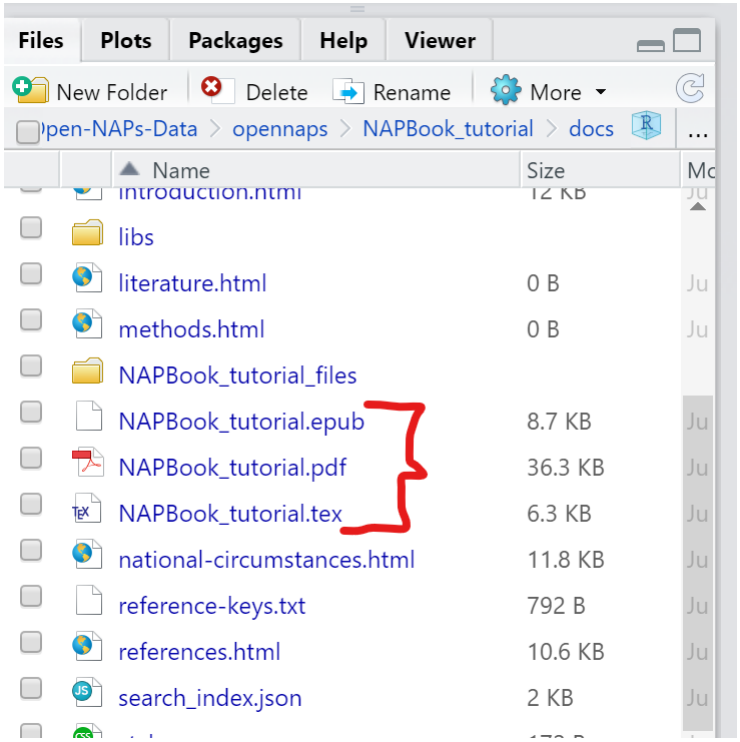


3.2.1 Some things to note

1. To create additional chapters, create a new /empty .rmd file and save it under your bookdown directory. Use the same numbering structure as the default chapters (i.e. 01,02,03, etc).
   We will learn about advanced numbering and re-ordering chapters later.

2. Use the `knit` function to render and preview a single chapter.

3. From our output.yml file, you will notice that we have 3 output options for our file; gitbook, pdf and epub. This is the default. You may select the most preferred output type by deleting the other/unwanted formats or leave this as default and choose a single output format when building the book. Since we left the default 3 options, when we rendered the book, it was compiled in .tex/gitbook, pdf and epub. This can be accessed from the 'docs' folder.



4. When building the book. Here you may choose to build one or all formats. From the Build Book button, click on the drop-down arrow and select your preferred format.

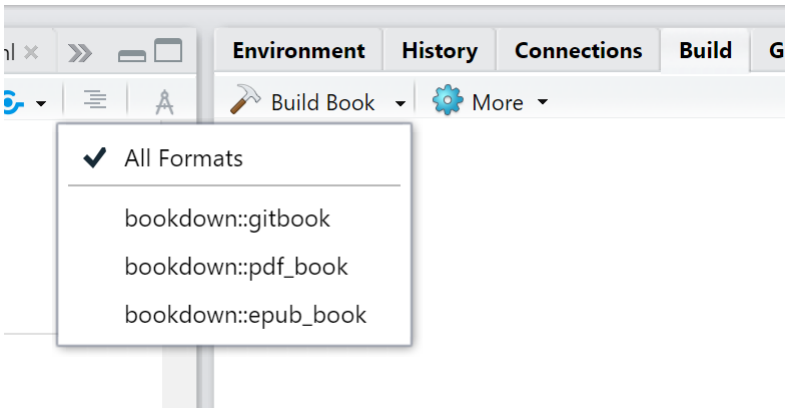


Figure 3.5: Build book format options

3.3 Some Troubleshooting

Creating pdf documents using LaTeX engines and distributions such as TinyTeX is not a straightforward task and may produce errors as the process involves multiple processing activities. However, most errors can be solved using suggestions contained in the error messages produced in the Environment or Console windows.
If, in any case, the LaTeX error generated is not clear, you may use any the commands options below to solve the problem.

Remove the # sign to run code

```
# remotes::install_github('yihui/tinytex') ##install the development version of tinytex

# update.packages(ask = FALSE, checkBuilt = TRUE) ## update your r and
# tinytex::tlmgr_update() ## tinytex packages

# tinytex::reinstall_tinytex()  ## 3 reinstall tinytex

# options(tinytex.verbose = TRUE) ## 4 set this option in an r code chunk.This helps p
```

See more on TinyTex Debugging.

Additionally, quite often, LaTeX formatting is not very compatible with other output formats such as html. But with the use of `html widgets` and other advanced formatting options, this problem can be overcome. To produce a pdf document from a document with both LaTeX and html formats, it may be useful to install the package `webshot` from CRAN.

```
#install.packages("webshot")
# webshot::install_phantomjs()
```

Further reading on html widgets here

Chapter 4

GitHub & GitHub Pages

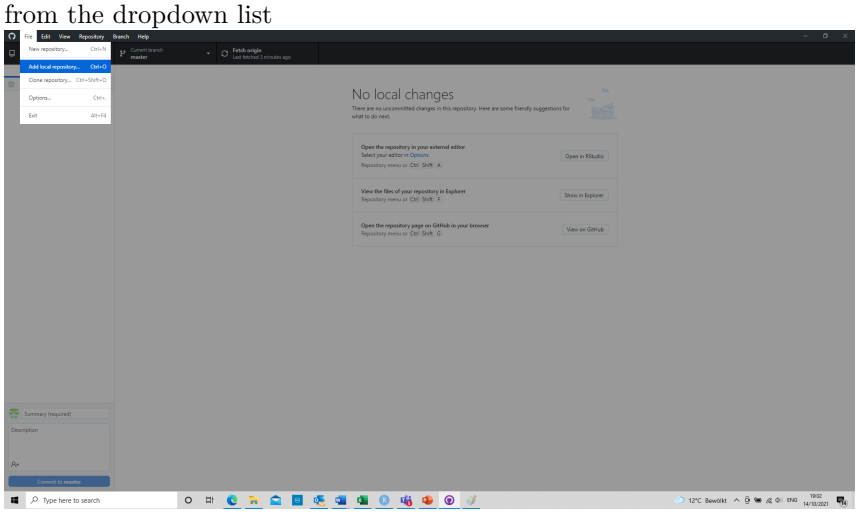
Github is great for project collaboration, backup and version control. To use github as your repository manager, Create an account at (https://github.com/).

4.1 Sharing Repositories on GitHub

4.1.1 Via GitHub Desktop

1. Open GitHub desktop on your pc/laptop

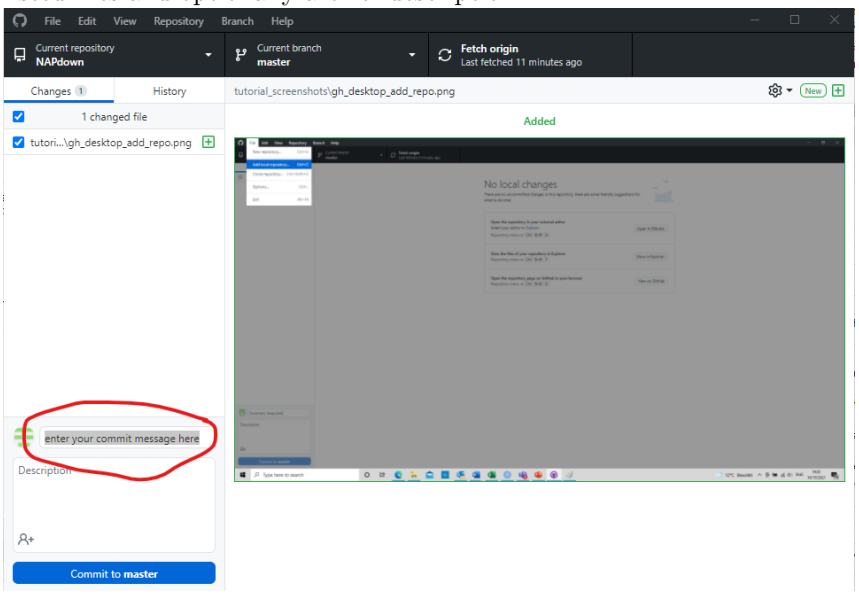
2. On the main menu, click on `File` and select `Add local repository` from the dropdown list



3. Browse to your your repository is sitting in your pc, select it and click 'Add Repository'

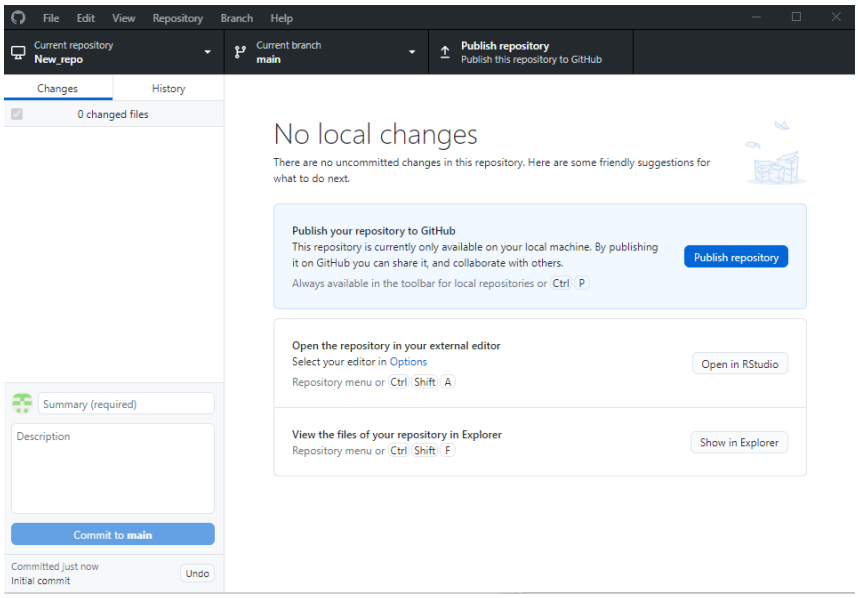
4. A list of all your files and sub-directories appears from which you may select and deselect what you want to publish to GitHub.

5. Enter a commit message on the 'Summary' text-box right below your listed files and optionally a brief description.

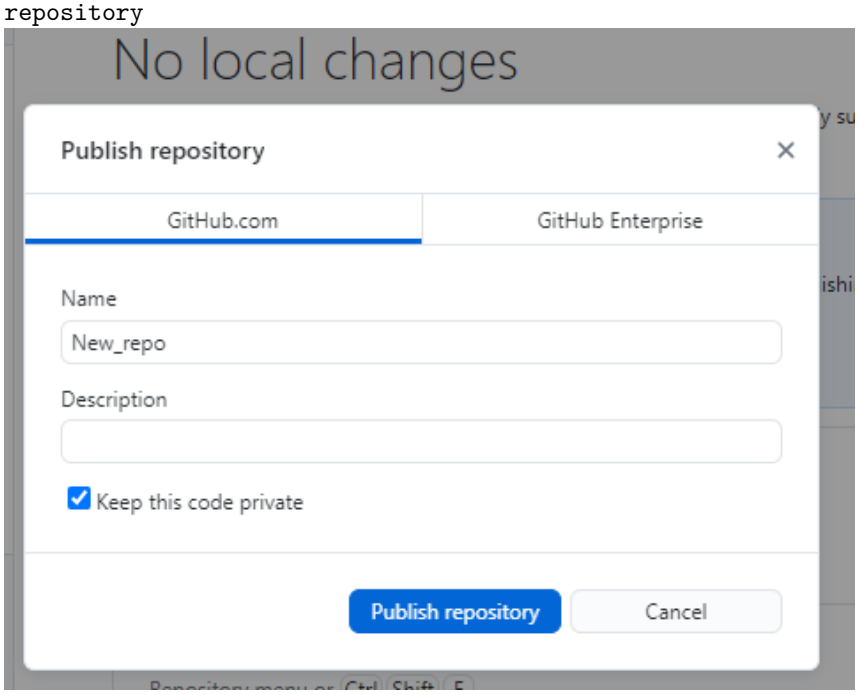


6. Hit 'Commit to master'

7. To the right of the GitHub desktop window, the app will notify you that this repository is only available on your local machine and if you would like to share/publish it to GitHub. Click on publish repository



8. Uncheck the box for 'keep the code private' and Click `Publish repository`

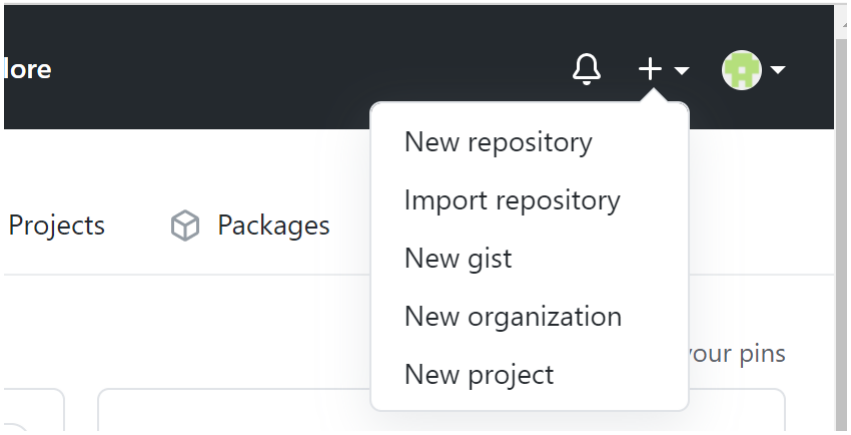


9. From the menu bar on GitHub desktop, click on `Repository` and select `Push` from the drop-down list.

10. Congratulations, you just published your repository to GitHub

4.1.2 Via direct file upload

1. Log in to your GitHub.com account
2. Click on the plus sign + located at the top right corner of your github account page.



3. Click on 'New Repository' from the drop-down menu.
4. On the new window that appears, give the repository a name: NAP-Book_Tutorial
5. Leave the repository as 'Public' 6. Leave everything else as is
7. Scroll down and click on 'Create Repository'
8. On the resulting window, click on 'upload an existing file'

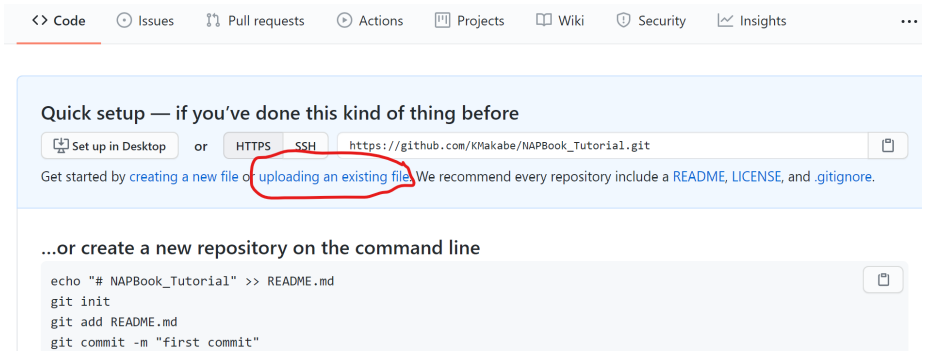


Figure 4.1: import repo to github

This will take you to a new window, from which you can drag-&-drop or browse

to your files.

9. Click on 'choose your files'.

10. Navigate with the file explorer to where we saved our 'NAPBook_tutorial' folder in your pc.

11. Select everything within this folder, and click 'Open'.
    Your files will start loading. Give it a minute or so.
    If you scroll through the files you will notice that our 'docs' folder is missing. Do not worry!

12. To add the folder, lets use the drag and drop method. From your file explorer, navigate to our NAPBook_tutorial folder, and click once on the 'docs' folder to select it.

13. Drag and drop this folder into your github repository. The files should start uploading.

14. After your files finish uploading, scroll down to the 'Commit changes' field; here you may enter a short description for your files. Let's enter the text 'our first NAP book commit'
    When making changes to your files, you may use this field to briefly describe what changes you made, otherwise commonly known as commits.

15. Next, hit the 'Commit changes' button at the end. This is called commiting.

4.1.3 Via Git Bash

1. Download and install Git Bash from this link (https://git-scm.com/downloads).

2. Repeat steps 1 - 7 in the preceding section.

3. On the new window, click on the 'https' tab to reveal the url for your repository.

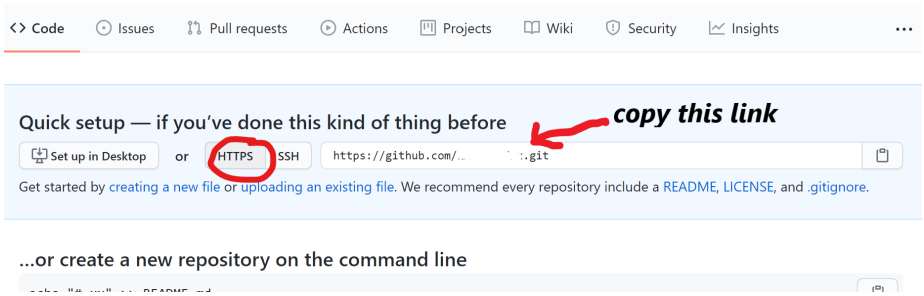


Figure 4.2: Repository url

4. Luanch Git Bash on your pc. This opens up a window with your pc name in the text. This is the command window.

5. Set the working repository to where your project files are located. To do this, type in commamd `cd "path to your project files directory`. Click Enter on your keyboard.

6. Type in `git init` to initialize this directory as your file origin.

7. Type in `git remote add origin` the paste the https link from step 4. Click Enter.

8. Type in `git add .` to add all project files in your pc to your github repository.

9. To add only specific file, type in `git add .insert name of file`

10. Type in `git commit -m`, open quotation marks and insert a commit message. Click Enter.

11. Type in `git push origin master` and click Enter. This pushes/uploads your files plus the commit message to your github repo.
Below is a list of the above commands

```
ls  # check working directory
cd "filepath"  # set working directory
git init  # initialize directory
git remote add orgin https.....(your github repo link)  # clone your local and
 github repositories
git add .  # add all files in local repository to github
git commit -m "your commit message"  # add a commit message
git push origin master  # pushes files from local to github repository
git help  # get help
git status  # check if your local and github repo are upto date.
Files with changes that have not been uploaded to github will be listed here.
```

Figure 4.3: Upload files via git bash

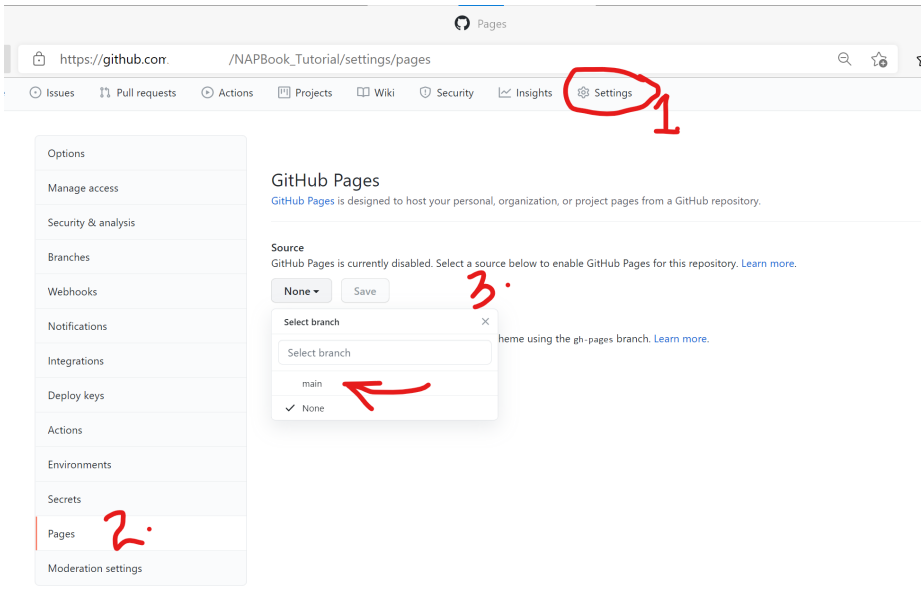
Congratulation once again!

To share your repository with colleagues and friends, just copy the link on your browser and share it with them. The link should take a similar identity as below;

https://github.com/yourusername/NAPBook_Tutorial

4.2 Publishing to GitHub pages

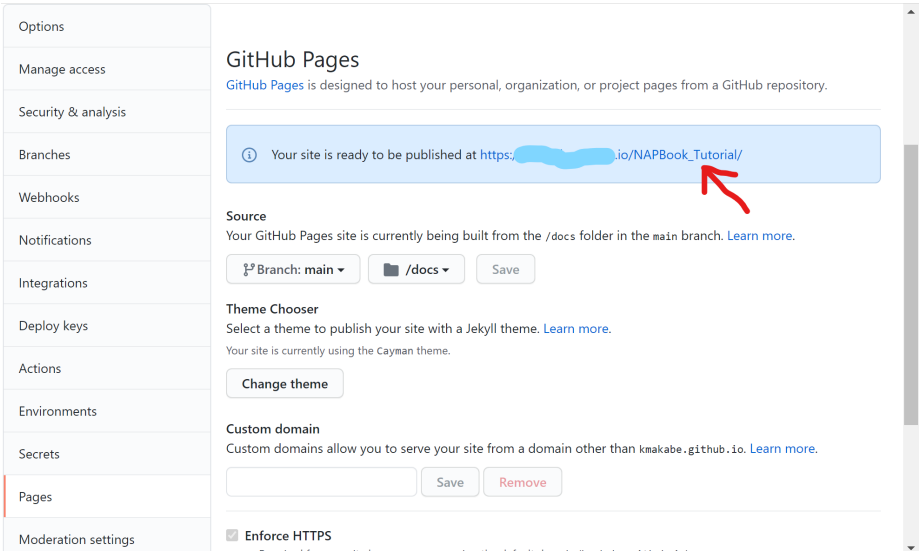
Github pages helps you to create/publish websites in very simple steps. We will publish the NAP book we just created with bookdown into a git-based website. To do this,
1.In your github repository, click on the `Settings` tab (right side of your screen)
2. Scroll down the listed menu items on the left side of the screen until you find menu item `Pages`. Click on it.



3. Scroll down to the 'Source' field. Click on the drop-down arrow and select the **main/master** branch and **docs** folder as your source files for your website. Click Save.
A message with a link to your website will appear, right above the `Source` field.

> Your site is ready to be published at https://yourusername.github.io/repositoryname/

or as seen here:



Use this link to view your newly created website.
4. Alternatively, navigate back to your main repository area, scroll down to your right to find your active `github-pages`. Click to view your website deployments.

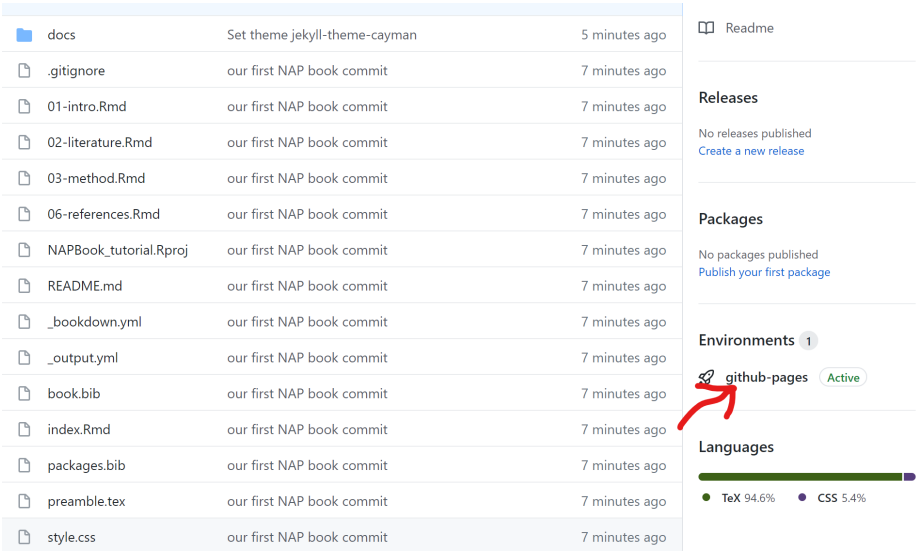


Figure 4.4: gh pages deployment

Now give yourself a pat on the back!

Chapter 5

Working Excel data

5.1 Prep

Steps;

5.1.1 Install packages

```
#install.packages(c("readxl", "magrittr", "knitr", "kableExtra", "DT", "plotly", "dplyr", "ggplot
```

5.1.2 Call libraries for the installed packages

```
library(readxl)
library(magrittr)
library(knitr)
library(kableExtra)
library(DT)
library(plotly)
library(dplyr)
library(ggplot2)
library(treemapify)
```

5.1.3 Import excel table from local drive

3. From the Environment window, click 'Import Dataset'

4. From the drop down menu, select 'From Excel'
5. A new window opens; use this to browse to your target excel file and select it
6. In case of multiple worksheets in your excel file, you may specify
7. the name or number of worksheet you are interested in. Otherwise, the first sheet is imported by default
8. Click 'Import' on the bottom right of the window
9. Your file should appear in the global environment
10. Copy the code from your console into your code environment (you may change the name of your data object to make it shorter or more descriptive)
11. Run code
12. Click on the 'play' icon to view data attributes or
13. Click on the data object or table sign on the far right of your data set name to view the data

```
gcf<-read_excel("C:\\Users\\Esther Makabe\\NAP_Progress_Aggregate\\Open_NAPs_Database.x
```

5.2 Build Tables

5.2.1 Static table

1. Open a new code chunk
2. Call the 'datatable' function by typing in 'datatable ()' # autofill is your friend
3. Fill in details as below (*since* we are only plotting the first 10 rows of our table)
4. Hover over the function to see parameters required or go to the 'Help' tab and search 'datatable' to see further details on its application.

```
kable(head(gcf[,c(2,3,5,8)],10),caption = 'GCF Funding')%>%
kable_styling(bootstrap_options = c('condensed', 'bordered'),fixed_thead = TRUE, font_s
```

5.2.2 Interactive table

1. Open a new code chunk
2. Install packages DT and magrittr
3. Load their libraries

4. Call 'datatable' function by typing in 'datatable ()' # autofill is your friend
5. Hover over the function to see parameters required or go to the 'Help' tab and search 'datatable' to see further details on its application.

Table 5.1: GCF Funding

countryname	Region	Project Name
Antigua and Barbuda	Latin America and the Caribbean	Resilience to hurricanes in the buil
Antigua and BarbudaDominicaGrenada	Latin America and the Caribbean	Integrated physical adaptation and
Bahrain	Asia-Pacific	Enhancing climate resilience of the
Bangladesh	Asia-Pacific	Climate Resilient Infrastructure Ma
Bangladesh	Asia-Pacific	Enhancing adaptive capacities of co
Bangladesh	Asia-Pacific	Extended Community Climate Cha
Belize	Latin America and the Caribbean	Resilient Rural Belize (Be-Resilient
Benin	Africa	Enhanced climate resilience of rura
Bhutan	Asia-Pacific	Supporting Climate Resilience and
Burkina Faso	Africa	Africa Hydromet Program – Streng

6. Fill in details as below

```
gcf_columns<-gcf[,c(2,3,5,8)] # selects only the columns we want to see in our table
datatable(gcf_columns,filter = 'top',rownames = F, editable = F, style = 'jqueryui', class = 'dis
  DT::formatStyle(columns = colnames(gcf_columns),fontSize= '10px')
```

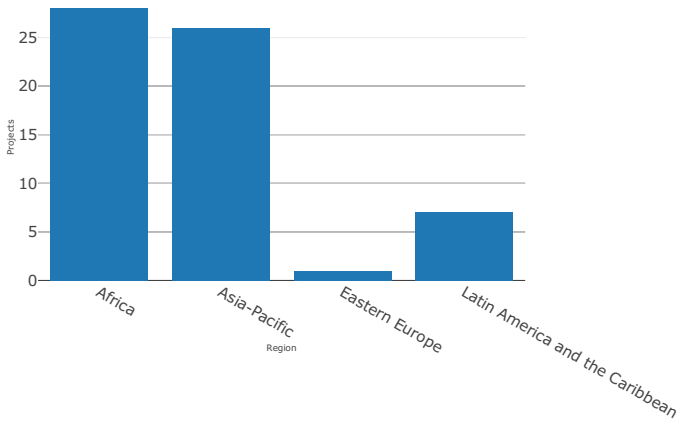
5.3 Generate Other Graphics

5.3.1 A bar chart

1. Open a new code chunk
2. Install packages plotly & dplyr
3. Call their respective libraries
4. To plot the no of projects per region, first group your data by region, then count the no of projects (use a unique identifier, in our case the project ID)
5. Call the function 'plot_ly' and enter parameters as in below

```
gcf %>% group_by(Region)%>% count(gcf$`Project Name`)%>%
  summarise("Projects"=sum(n))%>%   plot_ly(type = "bar",
          y = ~Projects,  x = ~Region

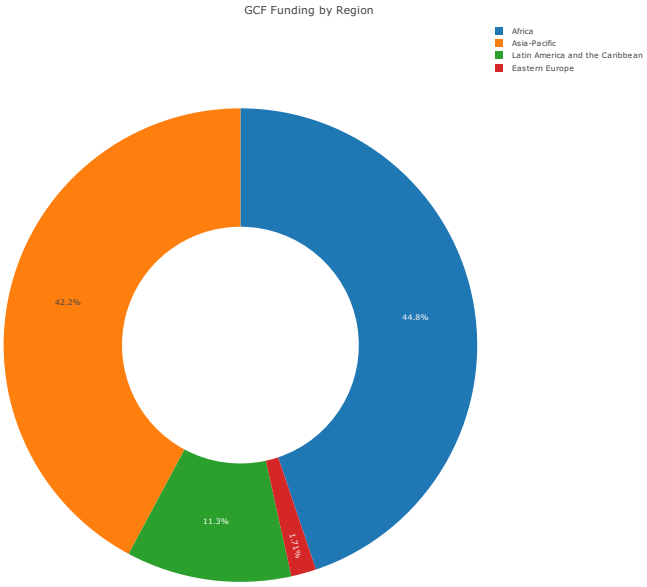
                ) %>%
plotly::layout(yaxis=list(tickfont=list(size=24)),
                 xaxis=list(tickfont=list(size=24)))
```



5.3.2 A donut chart

To plot the same data as above but on a pie chart, the process the same, only the type of chart changes

```
gcf%>%group_by(Region)%>%summarise('Total'=sum(`Total GCF Funding`))%>%plot_ly(labels=
    layout(title="GCF Funding by Region")
```

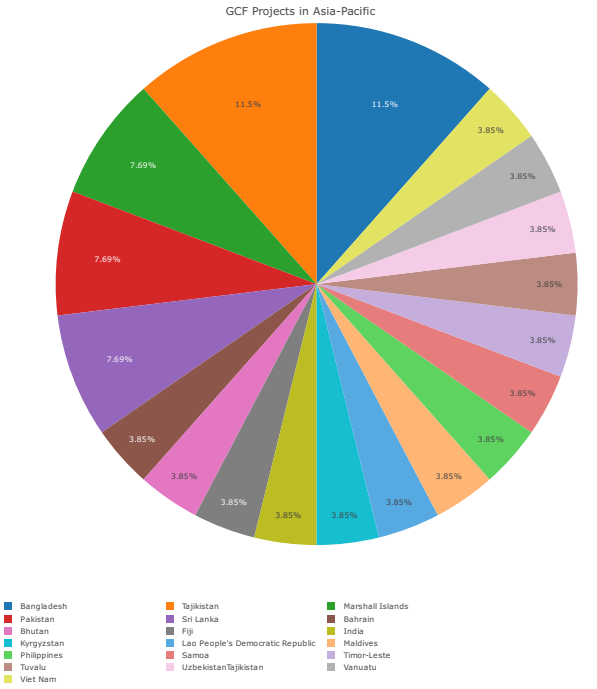


5.4 Filter and plot select data

5.4.1 Pie chart

To plot projects for a select region (s), use the filter function to select only values that match your selection, then group the data by country and create count by unique identifier as above. Repeat the plotting steps as above. you may use any type of charts as may be preferred.

```
gcf%>%filter(Region=="Asia-Pacific")%>%
  group_by(countryname)%>%count(`Project Name`)%>%
        plot_ly(labels=~countryname, values=~n)%>%
    add_pie()%>%
    layout(title=" GCF Projects in Asia-Pacific",
           legend=list(orientation='h'))
```



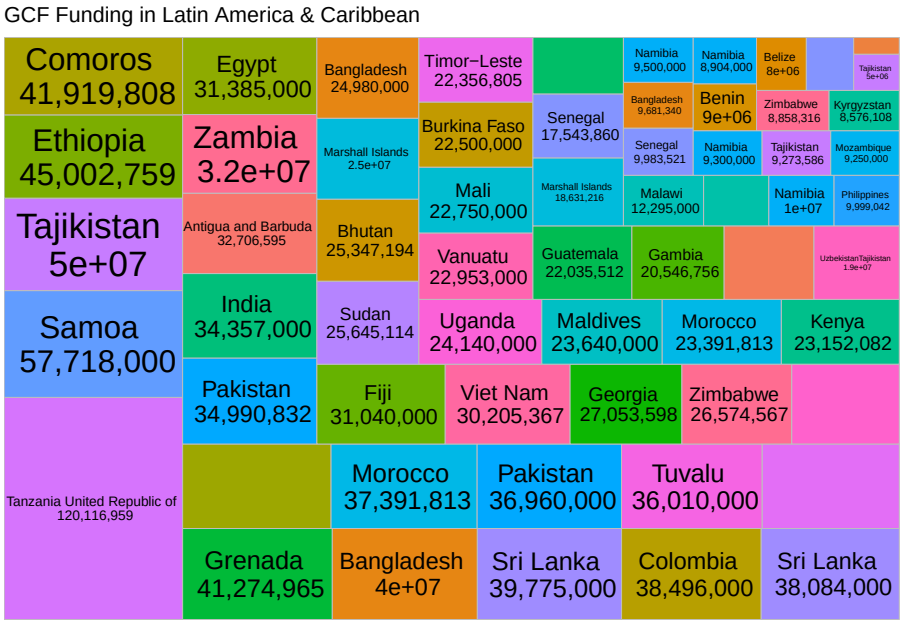
5.4.2 Build Treemap

5.4.2.1 Static

1. Open a new code chunk
2. Install packages ggplot2 & treemapify
3. Call the respective libraries

4. To plot amount of grant in Region x, filter values for only that region, then summarise the data by country using the functions 'group-by' and 'sum'
5. Call the function 'ggplot' and enter parameters as shown in screenshot above
6. Run code

```
ggplot(gcf, aes(fill=countryname,
       area=gcf$`Total GCF Funding`,
       label = paste(countryname,"\n",prettyNum(`Total GCF Funding`,
                                                big.mark = ","))))+
  geom_treemap()+
  geom_treemap_text(colour='black', place='centre')+
  labs(subtitle = 'GCF Funding in Latin America & Caribbean')+
  theme(legend.position = 'none')
```



5.4.2.2 Interactive

```
gcf_sum<-gcf%>%filter(Region=="Africa")%>%
  group_by(countryname, Region)%>%summarise("Total"=sum(`Total GCF Funding`))

plot_ly(
```

```
  data = gcf_sum,
  type= "treemap",
  values = ~Total,
  labels= ~countryname,
  parents= ~Region,
  name = "GCF Funding",
  textinfo="label+value+percent parent")
```

Chapter 6

Working Climate data

The first step is to set your working directory, where your files are stored. You can do this from the toolbar tab session, and choose 'Set Working Directory' from the drop down menu, and navigate to your folder. Or use the command `setwd(enter your path here)`

6.1 Historical climate data

6.1.1 Install Packages and load libraries

```
#install.packages("R.utils","rnaturalearth","reshape","raster","magrittr","dplyr","lubridate")
library(R.utils)
library(rnaturalearth)
library(reshape)
library(raster)
library(magrittr)
library(dplyr)
library(lubridate)
```

6.1.2 Load data from cru website

1. On your browser, navigate to (https://crudata.uea.ac.uk/cru/data/hrg/cru_ts_4.05/cruts.2103051243.v4.05/). This is where all data variables are stored.
2. Select the variable of your choice. In the example below, we will use precipitation and mean temperature data, variable names 'pre' & 'tmp' respectively.

3. On the data folder, find data for the years 1901-2020(cru_ts4.05.1901.2020.pre.dat.nc.gz/ru_ts4.05.
4. Right click on it and copy link address. Open a code chunk and paste your link in an object(as in 'myurl' in below code).
5. Call the function `download.file` and enter details as below.
6. The files are stored as a compressed .gz file; to extract the files, call the function `gunzip` and enter file name with the .gz extension
7. Run code

```
# pre
#preurl<- "https://crudata.uea.ac.uk/cru/data/hrg/cru_ts_4.05/cruts.2103051243.v4.05/p
#download.file(preurl, destfile="cru_ts4.05.1901.2020.pre.dat.nc.gz")
#gunzip("cru_ts4.05.1901.2020.pre.dat.nc.gz")

# tmp
#tmpurl<- "https://crudata.uea.ac.uk/cru/data/hrg/cru_ts_4.05/cruts.2103051243.v4.05/t
#download.file(tmpurl, destfile="cru_ts4.05.1901.2020.tmp.dat.nc.gz")
#gunzip("cru_ts4.05.1901.2020.tmp.dat.nc.gz")
```

6.1.3 Load/define geometry

This is where data to define your area of interest goes. You may upload your own geometry,such as shapefile or use any of the map services available in r. In this case we will use country boundaries from the package `rnaturalearth`

```
malawi<-rnaturalearth::ne_countries(country ='malawi')
```

6.1.4 Load/Import your data into r

Create new object name and call the raster function `stack` Enter the path to your data directory/ where you saved the downloaded file. To extract data exactly to your polygon shape, use the raster functions `crop` and `mask` and the bounding geometry (in our case the Malawi national boundary)

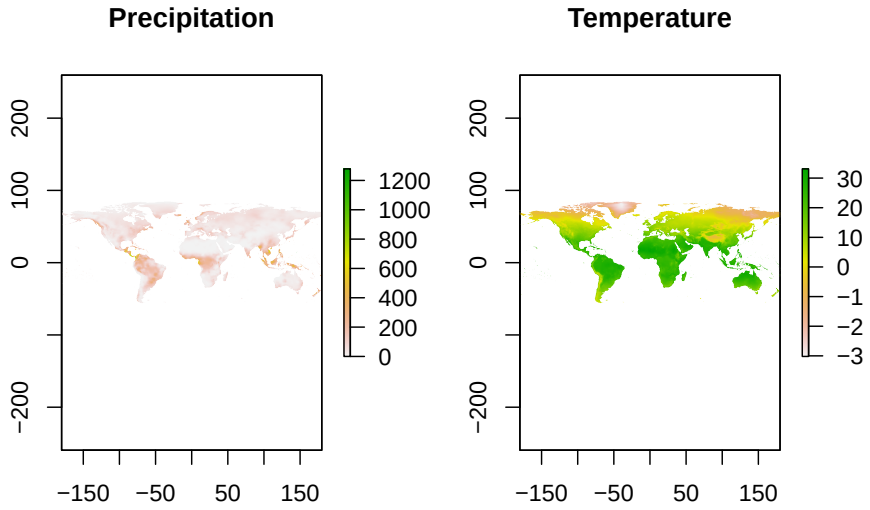
```
prstack<-stack("C:\\Users\\Esther Makabe\\NAPdown\\cru_ts4.05.1901.2020.pre.dat.nc")
temp<-stack("C:\\Users\\Esther Makabe\\NAPdown\\cru_ts4.05.1901.2020.tmp.dat.nc")

pr_crop<-raster::crop(prstack, malawi) # cut out data for malawi
pr_mask<-raster::mask(pr_crop,malawi) # mask data to malawi boundary

tcrop<-raster::crop(temp, malawi)
tmask<-raster::mask(tcrop, malawi)
```

At this point you can already plot individual precipitation and temperature layers using the basic plot function. Below we plot precipitation and temperature for October 2011

```
par(mfrow=c(1,2))
plot(prstack$X2011.10.16, main='Precipitation')
plot(temp$X2011.10.16, main='Temperature')
```



6.1.5 Convert the raster stack created above into a dataframe and format date values

This step extracts raster values at each station (depicted by its latitude and longitude) and stores them as a table.
The `melt` function converts the data from a wide to long table format.
The date values in our data are stored as and within text and the step below is to extract the dates from this text and store them in a new column.
We also, create two more columns to separate Year and Month

Caution!: The process in this code chunk involves modifying the data and data structure and thus extra attention is required. If you need to run the code or part of it more than once, it is strongly recommended to run the whole code chunk rather than in parts.This is to avoid duplication and conflict in executing commands.

```
prdf<-as.data.frame(pr_mask, xy=TRUE, na.rm=TRUE)%>%
  melt(id.vars=c('x','y')) # create dataframe
tmpdf<-as.data.frame(tmask, xy=TRUE, na.rm=TRUE)%>%
  melt(id.vars=c('x', 'y'))

Date<-substr(prdf$variable, 2,11) # extract date values from the dataframe
prdf$Date<-Date # add date column to dataframe
Year<-substr(Date,1,4)
Month<-substr(Date,6,7)
prdf<-cbind(prdf, Year, Month)

colnames(prdf)[colnames(prdf)=="value"]<-"pr"  # change column label


tmpdf$Date<-substr(tmpdf$variable, 2,11)
tmpdf$Year<-substr(tmpdf$Date,1,4)
tmpdf$Month<-substr(tmpdf$Date,6,7)

colnames(tmpdf)[colnames(tmpdf)=="value"]<-"tmean"
```

6.1.6 Plot Monthly data

Our data is now ready for some statistical analysis. we may plot monthly data as below;

```
# summarise data by month and calculate monthly mean
pr_monthly<-prdf%>%dplyr::filter(Year>=1981)%>% group_by(Month)%>%
  summarise(across(contains("pr"), ~mean(pr)))

# rearrange drawing order to plot from July to June
pr_monthly$Month<-factor(pr_monthly$Month,
                         levels =c('07','08','09','10','11','12','01','02','03','04','


temp_monthly<-tmpdf%>%dplyr::filter(Year>=1981)%>%group_by(Month)%>%
  summarise('tmean'=mean(tmean))
temp_monthly$Month<-factor(temp_monthly$Month,
                           levels = c('07','08','09','10','11','12','01','02','03','04

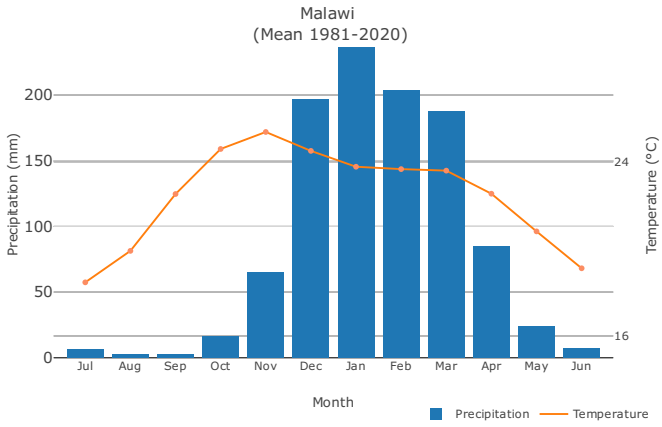
# combine the pr and temp data frames
pr_tmp<-cbind(pr_monthly,temp_monthly)
pr_tmp<-pr_tmp[,-3] # remove duplicate column

pr_tmp$Month<-month.abb
```

```
pr_tmp$Month<-factor(pr_tmp$Month,
                     levels = c("Jul", "Aug", "Sep" ,"Oct" ,"Nov",
                                "Dec", "Jan" ,"Feb", "Mar", "Apr" ,"May" ,"Jun"))

# define properties for secondary axis (only when plotting 2 variables in the same plot area)
ty<-list(overlaying = "y",
  side = "right",
  title = "Temperature (°C)",
  autotick = FALSE,
      dtick = 8,
 range=c(15,30)
  )

# plot
plot_ly(type= 'bar', data= pr_tmp, x= ~Month, y= ~pr, name = 'Precipitation')%>%
 add_lines(x= ~Month, y= ~tmean,name= 'Temperature', yaxis='y2')%>%
  add_markers(x= ~Month, y= ~tmean,color='#D21919', yaxis='y2', name='Temperature', showlegend=FA
    layout(legend=list(orientation='h', y=-0.13,x=0.65), yaxis=list(title='Precipitation (mm)',sh
  subplot(titleX = TRUE, titleY = TRUE)
```



6.1.7 Plot Annual data

or the annual means as below;

- Subset the data to get only values from 1981 on wards (or whichever year is preferred)
- Then, group the data by years
- Calculate mean values for each year
- Combine the data frames
- Plot time series object

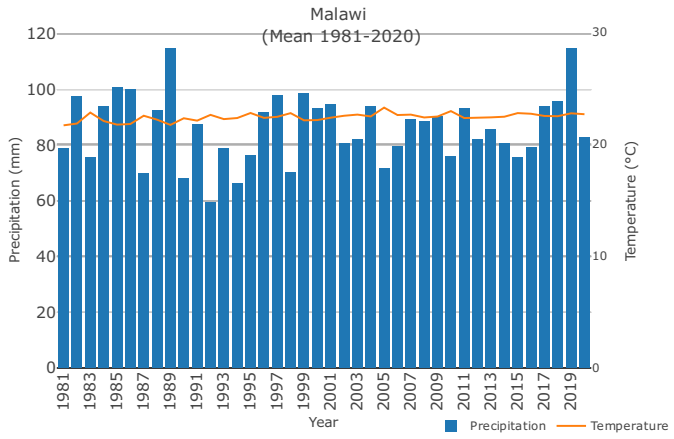
```
pr_ann<-prdf%>%dplyr::filter(Year>=1981)%>% group_by(Year)%>%
  summarise(across(contains("pr"), ~mean(pr)))
```

```
temp_ann<-tmpdf%>%dplyr::filter(Year>=1981)%>%group_by(Year)%>%summarise('tmean'=mean(tmean))

# combine the pr and temp data frames
pr.tmp<-cbind(pr_ann,temp_ann)
pr.tmp<-pr.tmp[,-3] # remove duplicate column

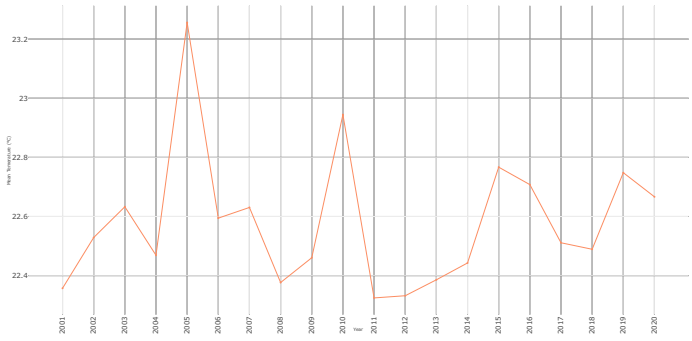
# define properties for secondary axis
ty<-list(overlaying = "y",
  side = "right",
  title = "Temperature (°C)",
  autotick = FALSE,
      dtick = 10,
 range=c(0,30)
  )

# plot
plot_ly(type= 'bar', data= pr.tmp, x= ~Year, y= ~pr, name = 'Precipitation')%>%
 add_lines(x= ~Year, y= ~tmean, mode = 'lines+markers',name= 'Temperature', yaxis='y2')%>%
    layout(legend=list(orientation='h', y=-0.13,x=0.7),
                   yaxis=list(title='Precipitation (mm)',showticklables=F, tickfont=list(size=16)),
                   yaxis2=ty, xaxis=list(tickangle=270,tickfont=list(size=16)))
```



The two preceeding plots are twin-plots, meaning two graphics have been combined in a single plot area. To plot just one you may be choose one of the dataframes say temperature and plot it on its own. See below

```
temp_ann%>%filter(Year>2000)%>% plotly::plot_ly()%>%
 add_lines(x= ~Year, y= ~tmean, color='red', showlegend=FALSE)%>%
  add_markers(x= ~Year, y= ~tmean, color='red', showlegend=FALSE, name='Temperature')%
  layout(yaxis=list(title='Mean Temerature (°C)',tickfont=list(size=22)),xaxis=list(ti
```



Chapter 7

Further Reads

Here we include a list of links to articles and other material that might be useful for this exercise.

1. R mardown basics
2. Bookdown basics
3. GitHub for bookdown

Bibliography

Yihui Xie. *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition, 2015. URL http://yihui.org/knitr/. ISBN 978-1498716963.