**Markdown** is a simple way to create text that rendered into an HTML webpage. In fact, using rmarkdown we can convert it to many different output types. The top three that you might be most likely to use are HTML, PDF, and Microsoft Word. In a nutshell, R Markdown stands on the shoulders of knitr and Pandoc. The former executes the computer code embedded in Markdown, and converts R Markdown to Markdown. The latter renders Markdown to the output format you want (such as PDF, HTML, Word, and so on).

How to convert to HTML, PDF, or Word?

1. Controlling the “knit” button

https://rmd4sci.njtierney.com/figs/rstudio-knit-button.png

1. changing the YAML option

title: "Document"

output: html\_document

title: "Document"

output: pdf\_document

title: "Document"

output: word\_document

rmarkdown provides an environment where you can write your complete analysis, and marries your text, and code together into a rich document. You write your code as code chunks, put your text around that, and then you have a document you can reproduce.

In order to ensure that RMarkdown works perfectly the following packages should be installed.

install.packages("rmarkdown")

install.packages("knitr")

install.packages("tinytex")

install.packages("here")

install.packages("tidyverse")

install.packages("broom")

install.packages("fs")

install.packages("usethis")

TinyTeX is a lightweight, portable, cross-platform, and easy-to-maintain LaTeX distribution. The R companion package tinytex can help automatically install missing LaTeX packages when compiling LaTeX or R Markdown documents to PDF

tinytex::install\_tinytex()

The RStudio layout has the following features:

* On the upper left, the Rmarkdown script
* On the lower left, the R console
* On the lower right, the view for files, plots, packages, help, and viewer.
* On the upper right, the environment / history pane

**The anatomy of an rmarkdown document**

This is an rmarkdown document (demo). It has three parts:

* Metadata (YAML, Ain’t Markup Language)
* Text (markdown formatting)
* Code (code formatting)

**Metadata Section**

The metadata of the document tells you how it is formed - what the title is, what date to put, and other control information.

---

title: "Example document"

author: "Gokarna Aryal"

output: html\_document

---

It starts and ends with three dashes ---, and has fields like the following: title, author, and output.

**Text Section- Inline formattting**

It provides a simple way to markup text

- bullet list

- bullet list

- bullet list

1. numbered list

2. numbered list

3. numbered list

\_\_bold\_\_, \*\*bold\*\*, \_italic\_, \*italic\*

Inline text will be *italic* if surrounded by underscores or asterisks, e.g., \_text\_ or \*text\*. **Bold** text is produced using a pair of double asterisks (\*\*text\*\*). A pair of tildes (~) turn text to a subscript (e.g., H~3~PO~4~ renders H3PO4). A pair of carets (^) produce a superscript (e.g., Cu^2+^ renders Cu2+).

To mark text as inline code, use a pair of backticks, e.g., `code`. To include n literal backticks, use at least backticks outside, e.g., you can use four backticks to preserve three backtick inside: ```` ```code``` ````, which is rendered as ```code```.

Hyperlinks are created using the syntax [text](link), e.g., [RStudio](<https://www.rstudio.com>)

**Code Section**

Code in an rmarkdown document in two ways,

code chunks

inline code

Code chunks are marked by three backticks and curly braces with r inside them:

```{r chunk-name}

# a code chunk

```

You can use the Insert Code Chunk button. Add a chunk label and/or chunk options inside the curly braces after r.

You can also insert an R code chunk by using keyboard shortcut Ctrl + Alt + I

Chunk names

Every chunk should have a name. For example

* one word that describes the action (e.g., “read”)
* one word that describes the thing inside the code (e.g, “gapminder”)
* separate words with “-” or "\_" (e.g., read-gapminder)

```{r}

fit=lm(dist~speed, data=cars)

b=coef(fit)

plot(cars)

abline(fit)

```

The slope of the regression is `r b[1]`.

**Remark:** In RStudio, you can create a new Rmd file from the menu

File -> New File -> R Markdown.

**Chunk options**

You can add arguments at the start of code chunks to change how the code is evaluated and presented. These arguments go in the first line ````{r, Arguments}. For example, you can tell R to echo (that is repeat) the code in the document by including the echo = TRUE argument, or change the size of figures created in the chunk with the fig.width and fig.height arguments.

```{r, echo=FALSE}

# echo - Display code in output document(default =TRUE)

```

```{r eval=FALSE}

# eval - Run code in chunk and evaluate(default = TRUE)

```

```{r include=TRUE}

# This is when include was set to true

```

* include = FALSE prevents code and results from appearing in the finished file. R Markdown still runs the code in the chunk, and the results can be used by other chunks.

*Whether to include anything from a code chunk in the output document. When include = FALSE, this whole code chunk is excluded in the output, but note that it will still be evaluated if eval = TRUE. When you are trying to set echo = FALSE, results = ‘hide’, warning = FALSE, and message = FALSE, chances are you simply mean a single option include = FALSE instead of suppressing different types of text output individually.*

* echo = FALSE prevents code, but not the results from appearing in the finished file. This is a useful way to embed figures.
* message = FALSE prevents messages that are generated by code from appearing in the finished file.
* warning = FALSE prevents warnings that are generated by code from appearing in the finished.
* fig.cap = "..." adds a caption to graphical results.

Example:

```{r,eval=FALSE, echo=TRUE, include=TRUE}

x=c(1,2,3,4,5,6,7)

mean(x)

summary(x)

```

x=c(1,2,3,4,5,6,7)

mean(x)

summary(x)

```{r,eval=TRUE, echo=TRUE, include=TRUE}

x=c(1,2,3,4,5,6,7)

mean(x)

summary(x)

```

x=c(1,2,3,4,5,6,7)

mean(x)

## [1] 4

summary(x)

## Min. 1st Qu. Median Mean 3rd Qu. Max.

## 1.0 2.5 4.0 4.0 5.5 7.0

```{r,eval=TRUE, echo=FALSE, include=TRUE}

x=c(1,2,3,4,5,6,7)

mean(x)

summary(x)

```

## [1] 4

## Min. 1st Qu. Median Mean 3rd Qu. Max.

## 1.0 2.5 4.0 4.0 5.5 7.0

```{r setup , include=FALSE}

knitr::opts\_chunk$set(echo = FALSE,

fig.align = "center",

fig.width = 4,

fig.height = 4,

dev = "png",

cache = TRUE)

```

**Example**

```{r, fig.align='center', fig.height=6,fig.width=4}

plot(cars,pch=18)

```

```{r library}

library(tidyverse)

```

```{r functions}

# A function to scale input to 0-1

scale\_01 <- function(x){

(x - min(x, na.rm = TRUE)) / diff(range(x, na.rm = TRUE))

}

```

```{r read-data}

gapminder <- read\_csv(here::here("data", "gapminder.csv"))

```

**Table of Contents**

We can add a table of contents (TOC) using the *toc* option and specify the depth of headers that it applies to using the toc\_depth option. For example:

---

title: "Example"

output:

html\_document:

toc: true

toc\_depth: 2

---

If the table of contents depth is not explicitly specified, it defaults to 3 (meaning that all level 1, 2, and 3 headers will be included in the table of contents).

We can specify the toc\_float option to float the table of contents to the left of the main document content. The floating table of contents will always be visible even when the document is scrolled. For example:

---

title: "Example"

output:

html\_document:

toc: true

toc\_float: true

---

We can add section numbering to headers using the number\_sections option:

---

title: "Example"

output:

html\_document:

toc: true

number\_sections: true

---