Lesson notes | R Markdown

Created by the GRAPH Courses team

April 2023

This document serves as an accompaniment for a lesson found on https://thegraphcourses.org.

The GRAPH Courses is a project of the Global Research and Analyses for Public Health (GRAPH) Network, a non-profit headquartered at the University of Geneva Global Health Institute, and supported by the World Health Organization (WHO) and other partners

Introduction

The {rmarkdown} package allows you to dynamically generate documents by mixing formatted text and results produced by R code. The generated documents can be in HTML, PDF, Word, and many other formats. It is therefore a very practical tool for exporting, communicating and disseminating analysis results.

There is a whole book on Rmarkdown, so we can only cover some of the essentials here.

This document was itself generated from R Markdown files.

Learning objectives

- You can create and knit an Rmarkdown document containing code and free text.
- You can output documents to multiple formats including HTML, PDF, Word, Powerpoint and flexdashboards.
- You understand basic markdown syntax.
- You can use R chunk options, including *eval*, *echo*, and *message*.
- You know the syntax for in-line R code.
- You recognize some useful packages for table formatting in Rmarkdown.
- You understand how to use the {here} package to force Rmarkdown files to use the project folder as the working directory.

Project setup

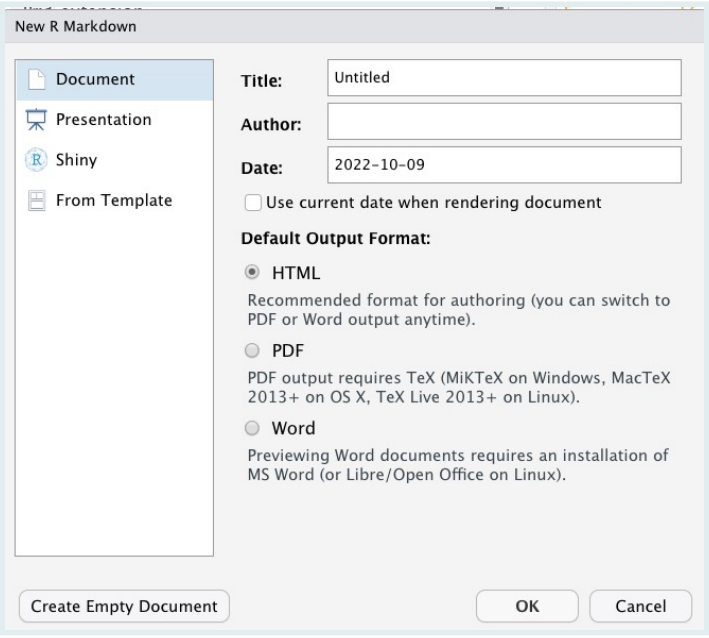
In RStudio, click on the *File* menu, and select *New Project...*. Then click on *New Directory*. Give your project a name, and select a directory into which to place it. (Make sure you remember where you put it!) Once you have these fields filled out, click *Create Project*.

Next we're going to set up some folders inside the project. Go to the *Files* pane, and click on *New Folder*. Name this one "data", and click *OK*. This is where you will put the data related to this project. Create one more called "rmd". R Markdown documents will go here.

Create a new document

An R Markdown document is a simple text file saved with the `.Rmd` extension.

In RStudio, you can create a new document by going to the *File* menu then choosing *New file* then *R Markdown...*. The first time you create an R Markdown document, you may be asked to install several packages. Go ahead and install those. Once RStudio has the appropriate packages, the following dialog box appears :



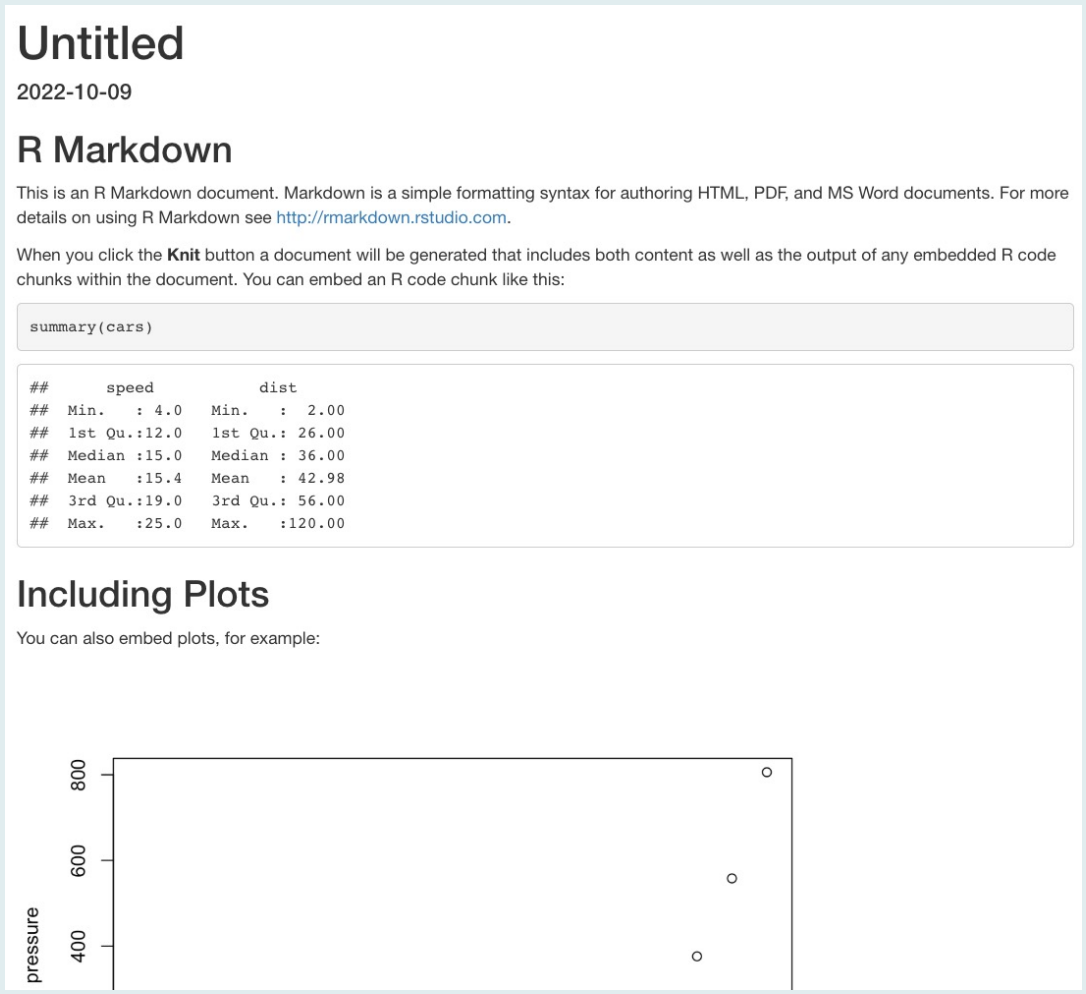
For now, you can leave all the default values and click OK. A file with sample content is then displayed.

Try editing some of the text in the file. Notice that is made up of some free text and some code sections.

Save your file with `Cmd/Ctrl + S`, remembering to give it the extension ".Rmd". E.g. "ebola_analysis.Rmd". Be sure to save it in the "rmd" folder you just created.

You can now try rendering the document by clicking on the "knit" button at the top right:

This will create an HTML output that looks like this:



Untitled

2022-10-09

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed           dist
##  Min.   : 4.0   Min.   :  2.00
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
##  Max.   :25.0   Max.   :120.00
```

Including Plots

You can also embed plots, for example:

This new rendered file is stored in the same directory as your Rmd. It has the same name, except it ends with ".html" instead of ".rmd".

HTML stands for Hyper text markup language and is the format that is used for most documents on the web.

Rmarkdown Header (YAML)

Now let's return to the rest of the Rmd to consider it part by part.

The first part of the document is its *header*. (It is also called "YAML", which stands for "Yet another markup language".) (The name is intended to be humorous.)

```
---
title: "Untitled"
output: html_document
date: "2022-10-09"
---
```

The YAML header must be located at the very beginning of the document, delimited by three dashes (`---`) before and after.

This header contains the document's metadata, such as its title, author, date, plus a whole host of possible options that will allow you to configure or customize the entire document and its rendering. Here, for example, the line `output: html_document` indicates that the generated document must be in HTML format.

We can change the `html_document` text to try out some other formats.

First you can make so

With the output set to "word_document", we get something like this:

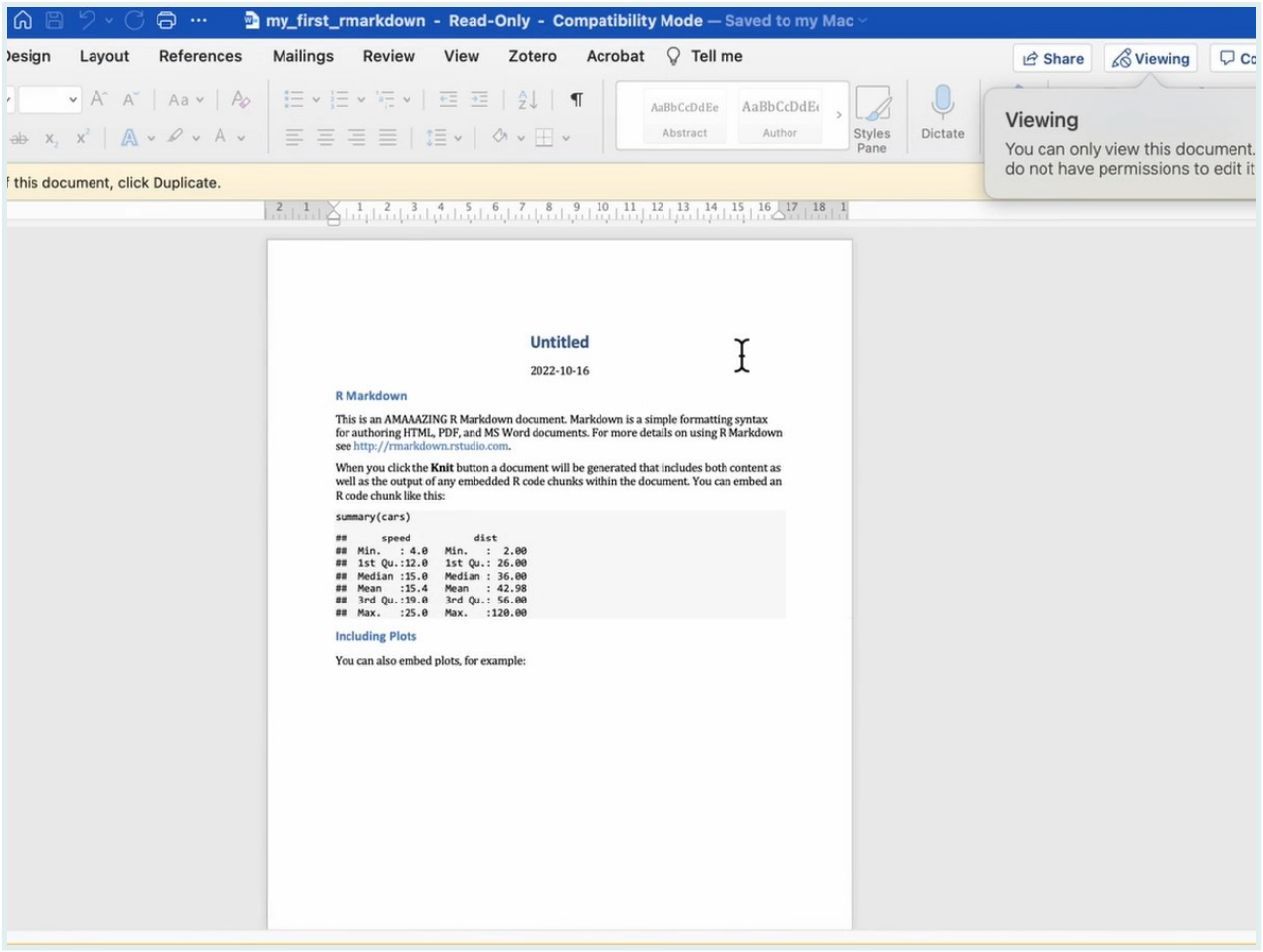


Image of the r markdown document open in the Microsoft Word program

Note that this creates a ".docx" version of our document in the "rmd" folder.

With the output set to "powerpoint_document", it comes out like this:

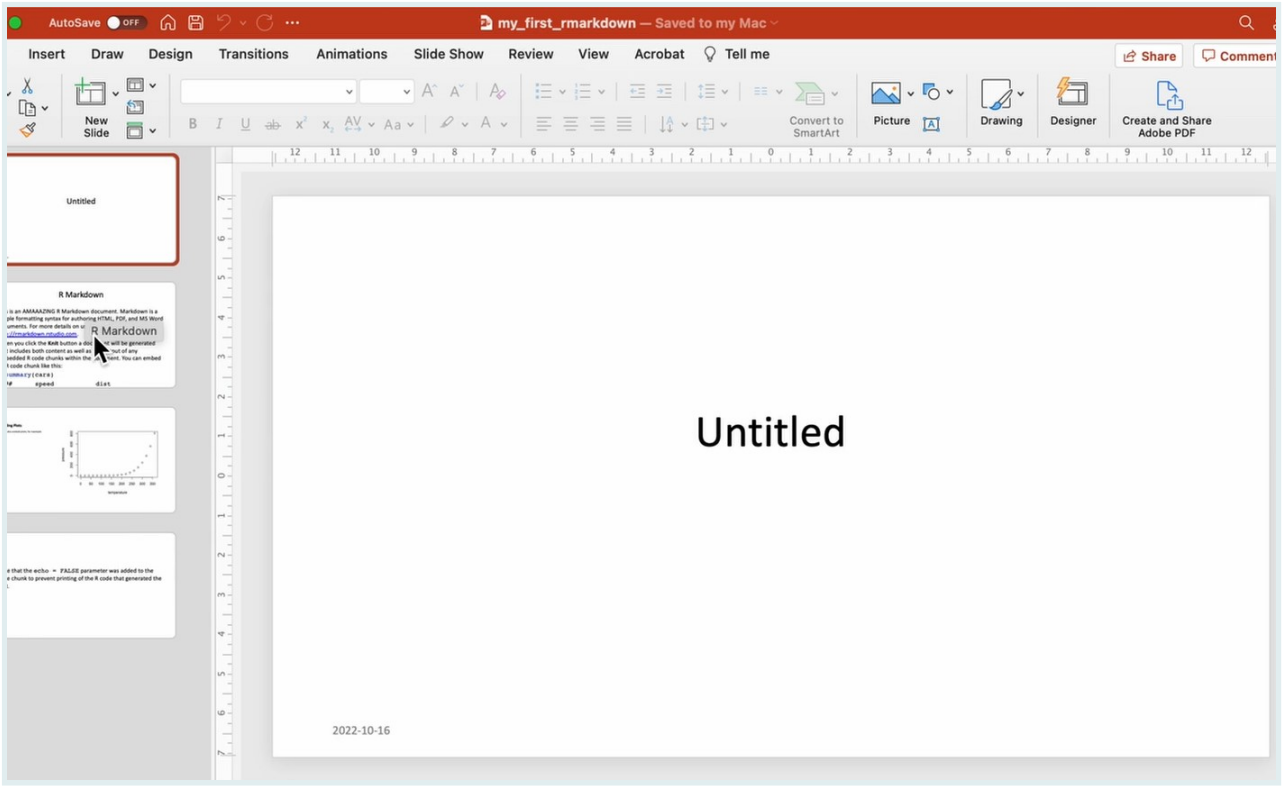


Image of the r markdown document open in the Microsoft Powerpoint program

If we change the output setting to "pdf_document", we can get the same document in PDF format (for this you may be prompted to install tinytex on your computer, see below):

Untitled

2022-10-09

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed           dist
##  Min.   : 4.0   Min.   :  2.00
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
##  Max.   :25.0   Max.   :120.00
```

Including Plots

You can also embed plots, for example:

KEY POINT

For PDF generation, you must have a working `LaTeX` installation on your system. If not, Yihui Xie's `tinytex` extension aims to make it easier to install a minimal `LaTeX` distribution regardless of your machine's

KEY POINT

operating system. To use it, you must first install the extension with `install.packages('tinytex')`, then run the following command in the console (expect a download of about 200MB): `tinytex::install_tinytex()` More information on the tinytex website.

There is also a file format called "prettydoc". To try this out, type `install.packages('prettydoc')` into the console and hit *enter*. The output format for prettydoc is a little different than the previous three we've seen, you need to type `prettydoc::html_pretty` in the `output` section. When you knit a prettydoc, you should see something like this:

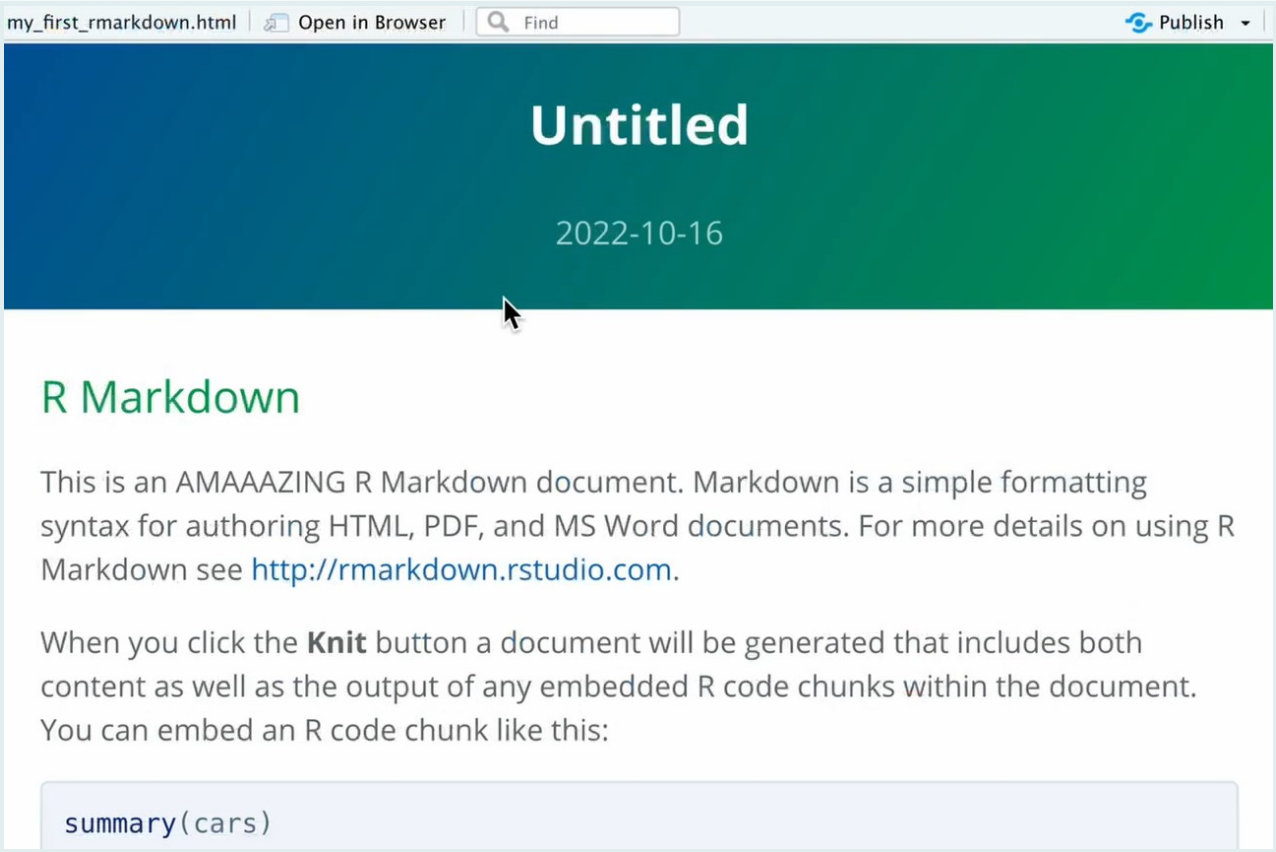


Image of the r markdown document as a prettydoc

We can even get a simple dashboard format. First we need to `install.packages('flexdashboard')` . Then if we set the `output` to `flexdashboard::flex_dashboard`, and knit, we get something like the following:

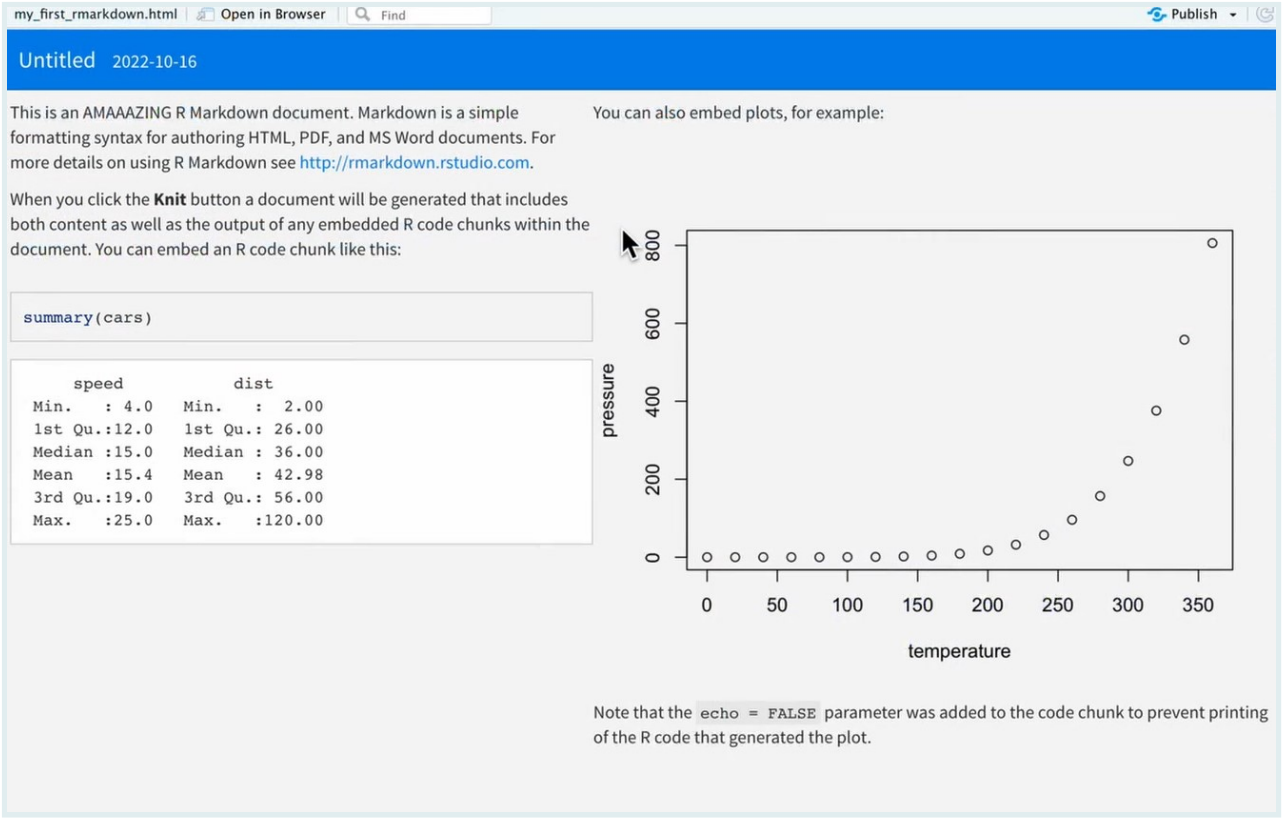


Image of the r markdown document as a flexdashboard

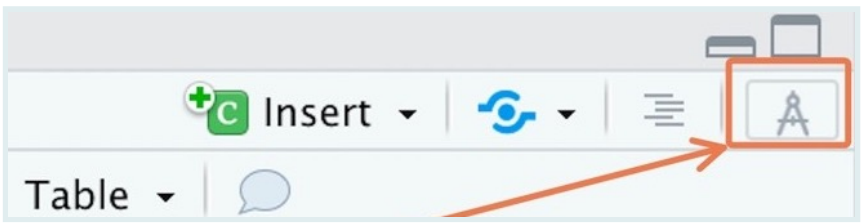
Note that it does not yet have tabs. To create tabs in a flexdashboard, change some of your double hashtags `##` to single hashtags `#`. This will change the header style for those sections, and get flexdashboard to render those headers as tabs instead.

Many other formats are possible, and we encourage you to explore on your own!

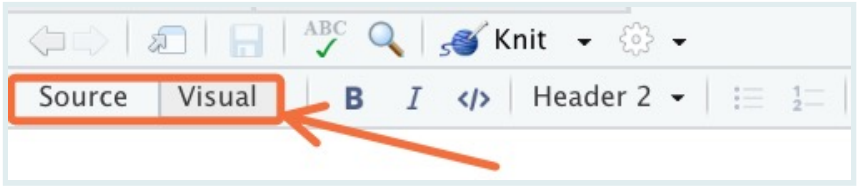
Visual vs Source mode

Rmarkdown documents can be edited in either a "Source" mode or a "Visual" mode.

You can switch into visual mode for a given document using the toolbars. For older RStudio versions, you may have an `A` button at the top-right of the document toolbar



For newer RStudio versions, there is a pair of buttons to toggle between the modes:



What's the difference between these two modes?

In source mode you see the raw markdown syntax.

```
## R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for
authoring HTML, PDF, and MS Word documents. For more details on using R
Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that
includes both content as well as the output of any embedded R code chunks
within the document. You can embed an R code chunk like this:

```{r cars}
summary(cars)
```
```

Markdown is a simple set of conventions for adding formatting to plain text. For example, to italicize text, you wrap it in as asterisk `*text here*`, and to start a new header, you use the pound sign `#` . We will learn these in detail below

But in visual mode, you instead see a Microsoft-word like WYSIWIG view:

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
{r cars}
summary(cars)
```

with a toolbar for easy formatting:

That means you do not have remember the syntax for markdown elements. For example, if you want to make a section of text bold, you can simply highlight that piece of text and click on the bold button in the toolbar.

Now, while visual mode is much easier to use, we will teach you markdown syntax here for three reasons:

- Visual mode is sometimes a buggy experience, and to debug this you'll need to switch to source mode
- Understandin markdown syntax is useful outside of Rmarkdown
- Visual mode is not available in RStudio's collaborative mode, which you may make use of

Markdown syntax

In the "Help" tab, if you look up "Markdown Quick Reference", you will be able to find a wide variety of RMD options available to you.

You can define titles of different levels by starting a line with one or more `#`:

```
# Level 1 title
## Level 2 Title
### Level 3 Title
```

The body of the document consists of text that follows the *Markdown* syntax. A Markdown file is a text file that contains lightweight markup that helps set heading levels or format text. For example, the following text:

```
This is text with *italics* and **bold**.

You can define bulleted lists:

- first element
- second element
```

Will generate the following formatted text:

> This is text with *italics* and **bold**.
>
> You can define bulleted lists:
>
> - first element
> - second element

Note that you need spaces before and after lists, as well as keeping the listed items on separate lines, or else they will all crunch together rather than making a list.

We see that words placed between asterisks are italicized, lines that begin with a dash are transformed into a bulleted list, etc.

The Markdown syntax allows for other formatting, such as the ability to insert links or images. For example, the following code:

```
[Example Link](https://example.com)
```

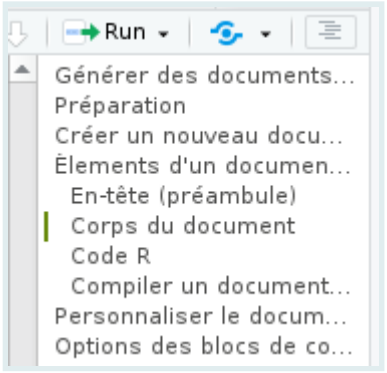
… will give the following link:

Example Link

We can also embed images. If you're in *Source* mode, type:

`[what you want the subtitle to say](images/picture_name.jpg)`, replacing "what you want the subtitle to say" (it can also be blank), "images" with the name of the image folder in your project, and "picture_name.jpg" with the name of the image you want to use. Of course, it is easier to do in *Visual* mode. From here, you can just open the folder that holds your image on your computer and drag-and-drop the image from the folder onto the page you're building. Or you can place the cursor where you want the image, click the button above marked with a "picture" icon, follow the prompts, and insert your image where the cursor is. Note that this will also create an "images" folder in your project (if it doesn't already exist) and put the image file into the "images" folder.

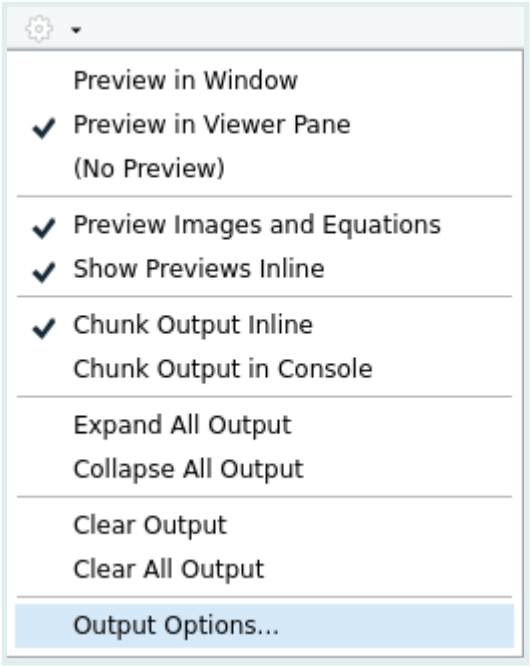
When titles have been defined, if you click on the *Show document outline* icon completely to the right of the toolbar associated with the R Markdown file, a table of contents automatically generated from the titles is displayed and allows you to navigate easily in the document:



Dynamic TOC

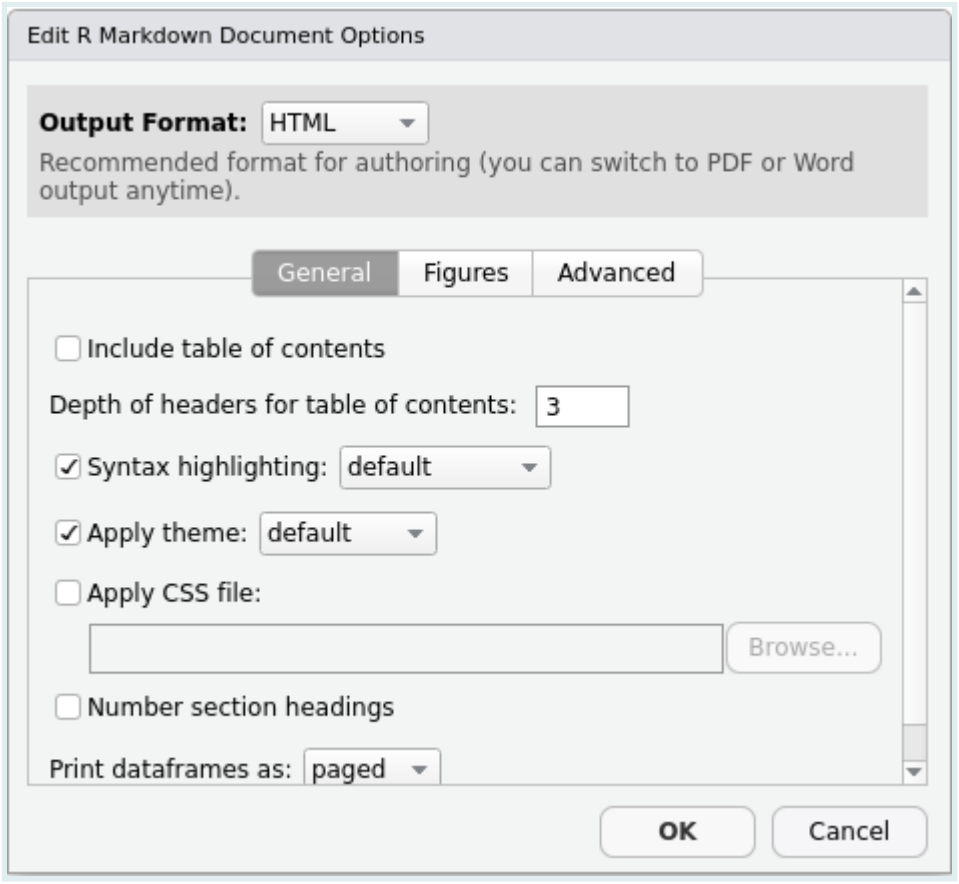
Customizing the generated document

The customization of the generated document is done by modifying options in the preamble of the document. However, RStudio offers a small graphical interface to change these options more easily. To do this, click on the gear icon to the right of the *Knit* button and choose *Output Options...*



R Markdown Output Options

A dialog box appears allowing you to select the desired output format and, depending on the format, different options:



R Markdown Output Options Dialog

For the HTML format for example, the *General* tab allows you to specify if you want a table of contents, its depth, the themes to apply for the document and the syntax highlighting of the R blocks, etc. The *Figures* tab allows you to change the default dimensions of the graphics generated.

When you change options, RStudio will actually change the preamble of your document. So if you choose to show a table of contents and change the syntax highlighting theme, your header will become something like:

```
---
title: "R Markdown Review"
output:
   html_document:
     highlight: kate
     knock: yes
---
```

You can modify the options directly by editing the preamble.

Note that it is possible to specify different options depending on the format, for example:

```
---
title: "R Markdown Review"
output:
  html_document:
    highlight: kate
    knock: yes
  pdf_document:
    fig_caption: yes
    highlight: kate
---
```

The complete list of possible options is present on the official documentation site (very complete and well done) and on the cheat sheet and the reference guide, accessible from RStudio via the *Help* menu then *Cheatsheets*.

R code chunks

In addition to free text in Markdown format, an R Markdown document contains, as its name suggests, R code. This is included in blocks (*chunks*) written the following way in *Source* mode:

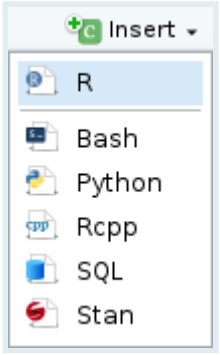
```
```{r}
r_code <- 2+2
```
```

Which will produce the following in *Visual* mode:

```
r_code <- 2+2
```

As this sequence of characters is not very easy to enter, you can use the *Insert* menu of RStudio and choose *R*[^3], or use the keyboard shortcut `Command+Option+i` on Mac or `Ctrl+Alt+i` on Windows.

Note that it is possible to use other languages in code chunks.



Code block insertion menu

In RStudio blocks of R code are usually displayed with a slightly different background color to distinguish them from the rest of the document.

When your cursor is in a block, you can enter the R code you want and execute it with Command + Enter. You can also execute all the code contained in a block by clicking on the green "play" button at the top right of the code chunk.

Chunk output inline vs in condole

In RStudio, by default, the results of a block of code (text, table or graphic) are displayed directly *in* the document editing window, allowing them to be easily viewed and kept for the duration of the session.

This behavior can be changed by clicking the gear icon on the toolbar and choosing *Chunk Output in Console.*

R code chunk options

It is also possible to pass options to each block of R code to modify its behavior.

Remember that a block of code looks like this:

````
```{r}
x <- 1:5
```
````

The options of a code block are to be placed inside the braces `{r}`, with a comma separating each option.

Block name

The first possibility is to give a *name* to the block. This is indicated directly after the `r`:

```
{r block_name}
```

It is not mandatory to name a block, but it can be useful in the event of a compilation error, to identify the block that caused the problem. Be careful, you cannot have two blocks with the same name.

Options

In addition to a name, a block can be passed a series of options in the form `option=value`. Here is an example of a block with a name and options:

````
```{r blockName, echo = FALSE, warning = TRUE}
x <- 1:5
```
````

And an example of an unnamed block with options:

```
```{r echo = FALSE, warning = FALSE}
x <- 1:5
```

One of the useful options is the `echo` option. By default `echo` is `TRUE`, and the block of R code is inserted into the generated document, like this:

```r
x <- 1:5
print(x)
```

```
## [1] 1 2 3 4 5
```

But if we set the `echo=FALSE` option, then the R code is no longer inserted into the document, and only the result is visible:

```
## [1] 1 2 3 4 5
```

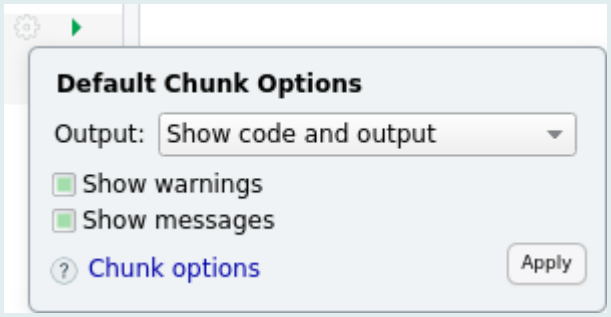
Here is a list of some of the available options:

Option	Values	Description
echo	TRUE/FALSE	Show (or hide) this R code chunk in the resulting knitted document
eval	TRUE/FALSE	Run (or not) the code in this code chunk in the resulting knitted document
include	TRUE/FALSE	Combines the options "echo and eval"; either show and run, or hide and don't run
message	TRUE/FALSE	Show (or hide) any system messages generated by running this code chunk in the resulting knitted document
warning	TRUE/FALSE	Show (or hide) any warnings generated by running this code chunk in the resulting knitted document

There are many other options described in particular in R Markdown reference guide{target = "_blank"} (PDF in English).

Change options

It is possible to modify the options manually by editing the header of the code block, but you can also use a small graphical interface offered by RStudio. To do this, simply click on the gear icon located to the right of the header line of each block:
```



Code Block Options Menu

You can then modify the most common options, and click on *Apply* to apply them.

Global Options

You may want to apply an option to all the blocks in a document. For example, one may wish by default not to display the R code of each block in the final document.

You can set an option globally using the `knitr::opts_chunk$set()` function. For example, inserting `knitr::opts_chunk$set(echo = FALSE)` into a code block will set the `echo = FALSE` option to default for all subsequent blocks.

In general, we place all these global modifications in a special block called `setup` and which is the first block of the document:

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

Inline Code

It is also possible to write code chunks embedded in the text. If you go to *Source* mode and type

“The sum of a pair of 2s is ` r 2+2 `”

and then knit the RMD, the resulting document will evaluate the r code between the backticks. Note that you have to include the “r” at the beginning of your inline code chunk to get it to recognize it as R code.

You could also pass variables around your document just like in a regular R program. For example, on one line you could run,

``` {r} max_height <- max(women$height) ```

“The maximum height in the women data set is ` r max_height ` .”

The advantages of such a system are numerous:

- a single document can show your entire analysis workflow, since the code, results and text explanations are included
- the document can be very easily regenerated and updated, for example if the source data has been modified.
- the variety of output formats (HTML, PDF, Word, slides, dashboards, etc.) makes it easy to present your work to others.

Display tables

There are a number of ways for R Markdown Documents to show data tables. To start, you can see how our RMD displays a table with no formatting:

```
women
```

```
##    height weight
## 1      58    115
## 2      59    117
## 3      60    120
## 4      61    123
## 5      62    126
## 6      63    129
## 7      64    132
## 8      65    135
## 9      66    139
## 10     67    142
## 11     68    146
## 12     69    150
## 13     70    154
## 14     71    159
## 15     72    164
```

It looks pretty basic. Next, to follow along you'll want to load the following packages:

```
pacman::p_load(flextable, gt, reactable)
```

Flextable is better for showing simple tables supported by many formats. GT is better for showing complex tables in HTML documents. Reactable is better for showing very large tables in HTML by giving your audience the option to scroll through the tables.

```
"This is a flextable"
```

```
## [1] "This is a flextable"
```

```
flextable::flextable(women)
```

height	weight
58	115
59	117
60	120
61	123
62	126
63	129
64	132
65	135
66	139
67	142
68	146
69	150
70	154
71	159
72	164

```
"This is a GT table"
```

```
## [1] "This is a GT table"
```

```
gt::gt(women)
```

height	weight
58	115
59	117
60	120
61	123
62	126
63	129
64	132
65	135
66	139
67	142
68	146
69	150
70	154
71	159
72	164

```
"This is a reactable"
```

```
## [1] "This is a reactable"
```

```
reactable::reactable(women)
```

You can see many other types of table formats people have created at https://www.rstudio.com/blog/rstudio-table-contest-2022/

Crosstabs

By default, tables from the `table` function are displayed as they appear in the R console, in plain text:

```
library(question)
```

```
## Error in library(question): there is no package called 'question'
```

```
data(hdv2003)
tab <- lprop(table(hdv2003$qualif, hdv2003$sex))
```

```
## Error in lprop(table(hdv2003$qualif, hdv2003$sex)): could not find
function "lprop"
```

```
table
```

```
## function (..., exclude = if (useNA == "no") c(NA, NaN), useNA = c("no",
##     "ifany", "always"), dnn = list.names(...), deparse.level = 1)
## {
##     list.names <- function(...) {
##         l <- as.list(substitute(list(...)))[-1L]
##         if (length(l) == 1L && is.list(..1) && !is.null(nm <- names(..1)))
##             return(nm)
##         nm <- names(l)
##         fixup <- if (is.null(nm))
##             seq_along(l)
##         else nm == ""
##         dep <- vapply(l[fixup], function(x) switch(deparse.level +
##             1, "", if (is.symbol(x)) as.character(x) else "",
##             deparse(x, nlines = 1)[1L]), "")
##         if (is.null(nm))
##             dep
##         else {
##             nm[fixup] <- dep
##             nm
##         }
##     }
##     miss.use <- missing(useNA)
##     miss.exc <- missing(exclude)
##     useNA <- if (miss.use && !miss.exc && !match(NA, exclude,
##         nomatch = 0L))
##         "ifany"
##     else match.arg(useNA)
##     doNA <- useNA != "no"
##     if (!miss.use && !miss.exc && doNA && match(NA, exclude,
##         nomatch = 0L))
##         warning("'exclude' containing NA and 'useNA' != \"no\"' are a bit
contradicting")
##     args <- list(...)
##     if (length(args) == 1L && is.list(args[[1L]])) {
##         args <- args[[1L]]
##         if (length(dnn) != length(args))
##             dnn <- paste(dnn[1L], seq_along(args), sep = ".")
##     }
##     if (!length(args))
```

```
##         stop("nothing to tabulate")
##     bin <- 0L
##     lens <- NULL
##     dims <- integer()
##     pd <- 1L
##     dn <- NULL
##     for (a in args) {
##         if (is.null(lens))
##             lens <- length(a)
##         else if (length(a) != lens)
##             stop("all arguments must have the same length")
##         fact.a <- is.factor(a)
##         if (doNA)
##             aNA <- anyNA(a)
##         if (!fact.a) {
##             a0 <- a
##             op <- options(warn = 2)
##             a <- factor(a, exclude = exclude)
##             options(op)
##         }
##         add.na <- doNA
##         if (add.na) {
##             ifany <- (useNA == "ifany")
##             anNAc <- anyNA(a)
##             add.na <- if (!ifany || anNAc) {
##                 ll <- levels(a)
##                 if (add.ll <- !anyNA(ll)) {
##                   ll <- c(ll, NA)
##                   TRUE
##                 }
##                 else if (!ifany && !anNAc)
##                   FALSE
##                 else TRUE
##             }
##             else FALSE
##         }
##         if (add.na)
##             a <- factor(a, levels = ll, exclude = NULL)
##         else ll <- levels(a)
##         a <- as.integer(a)
##         if (fact.a && !miss.exc) {
##             ll <- ll[keep <- which(match(ll, exclude, nomatch = 0L) ==
##                 0L)]
##             a <- match(a, keep)
##         }
##         else if (!fact.a && add.na) {
##             if (ifany && !aNA && add.ll) {
##                 ll <- ll[!is.na(ll)]
##                 is.na(a) <- match(a0, c(exclude, NA), nomatch = 0L) >
##                   0L
##             }
##             else {
##                 is.na(a) <- match(a0, exclude, nomatch = 0L) >
##                   0L
##             }
##         }
```

```
##         nl <- length(ll)
##         dims <- c(dims, nl)
##         if (prod(dims) > .Machine$integer.max)
##             stop("attempt to make a table with >= 2^31 elements")
##         dn <- c(dn, list(ll))
##         bin <- bin + pd * (a - 1L)
##         pd <- pd * nl
##     }
##     names(dn) <- dnn
##     bin <- bin[!is.na(bin)]
##     if (length(bin))
##         bin <- bin + 1L
##     y <- array(tabulate(bin, pd), dims, dimnames = dn)
##     class(y) <- "table"
##     y
## }
## <bytecode: 0x14f8d3350>
## <environment: namespace:base>
```

We can improve their presentation by using the `kable` function of the `knitr` extension. This provides suitable formatting depending on the output format. We will therefore have "clean" tables whether in HTML, PDF or word processing formats:

```
library(knitr)
kable(tab)
```

```
## Error in kable(tab): object 'tab' not found
```

Various arguments can be used to modify the output of `kable`. `digits`, for example, lets you specify the number of significant digits to display in number columns:

```
kable(tab, digits = 1)
```

```
## Error in kable(tab, digits = 1): object 'tab' not found
```

Data Tables and Flat Sorts

With regard to data tables (tibble or *data frame*), the default HTML display is content with a text display as in the console, very difficult to read as soon as the table exceeds a certain size.

An alternative is to use the `paged_table` function, which displays a paged HTML representation of the table:

	id	...	sexe	nivetud	poids
	<int>	<int>	<fctr>	<fctr>	<dbl>
1	1	28	Femme	Enseignement superieur y compris technique superieur	2634.39822
2	2	23	Femme	*NA*	9738.39578
3	3	59	Homme	Derniere annee d'etudes primaires	3994.10246
4	4	34	Homme	Enseignement superieur y compris technique superieur	5731.66151
5	5	71	Femme	Derniere annee d'etudes primaires	4329.09400
6	6	35	Femme	Enseignement technique ou professionnel court	8674.69938
7	7	60	Femme	Derniere annee d'etudes primaires	6165.80349
8	8	47	Homme	Enseignement technique ou professionnel court	12891.64076
9	9	20	Femme	*NA*	7808.87206
10	10	28	Homme	Enseignement technique ou professionnel long	2277.16047

1-10 of 2,000 rows | 1-6 of 21 columns Previous 1 2 3 4 5 6 ... 200 Next

Render a table by `paged_table`

An alternative is to use `kable`, as before for crosstabs, or the `datatable` function of the `DT` extension, which offers even more interactivity:

Show 10 entries Search:

	id	age	sexe	nivetud	poids	occup
1	1	28	Femme	Enseignement superieur y compris technique superieur	2634.3982157	Exerce une profession
2	2	23	Femme		9738.3957759	Etudiant, eleve
3	3	59	Homme	Derniere annee d'etudes primaires	3994.1024587	Exerce une profession
4	4	34	Homme	Enseignement superieur y compris technique superieur	5731.6615081	Exerce une profession
5	5	71	Femme	Derniere annee d'etudes primaires	4329.0940022	Retraite
6	6	35	Femme	Enseignement technique ou professionnel court	8674.6993828	Exerce une profession
7	7	60	Femme	Derniere annee d'etudes primaires	6165.8034861	Au foyer
8	8	47	Homme	Enseignement technique ou professionnel court	12891.640759	Exerce une profession
9	9	20	Femme		7808.8720636	Etudiant, eleve
10	10	28	Homme	Enseignement technique ou professionnel long	2277.160471	Exerce une profession

Showing 1 to 10 of 2,000 entries Previous 1 2 3 4 5 ... 200 Next

Render a table by `DT::datatable`

In any case, it is not recommended to display a very large data table in this way, because the resulting HTML file would contain all the data and would therefore be very bulky.

You can define a default display mode for all dataframes by modifying the *Output options* of the HTML format (*General* tab, *Print dataframes as*), or by manually modifying the `df_print` option of the `html_document` entry in the preamble. Note that the tables

resulting from the `freq` function of `questionr` are displayed as data tables (and not as cross tables).

Document Templates

We have seen here the production of "classic" documents, but R Markdown allows you to create many other things.

The extension's documentation site offers a gallery of the different possible outputs. You can create slides, websites or even entire books, like this document.

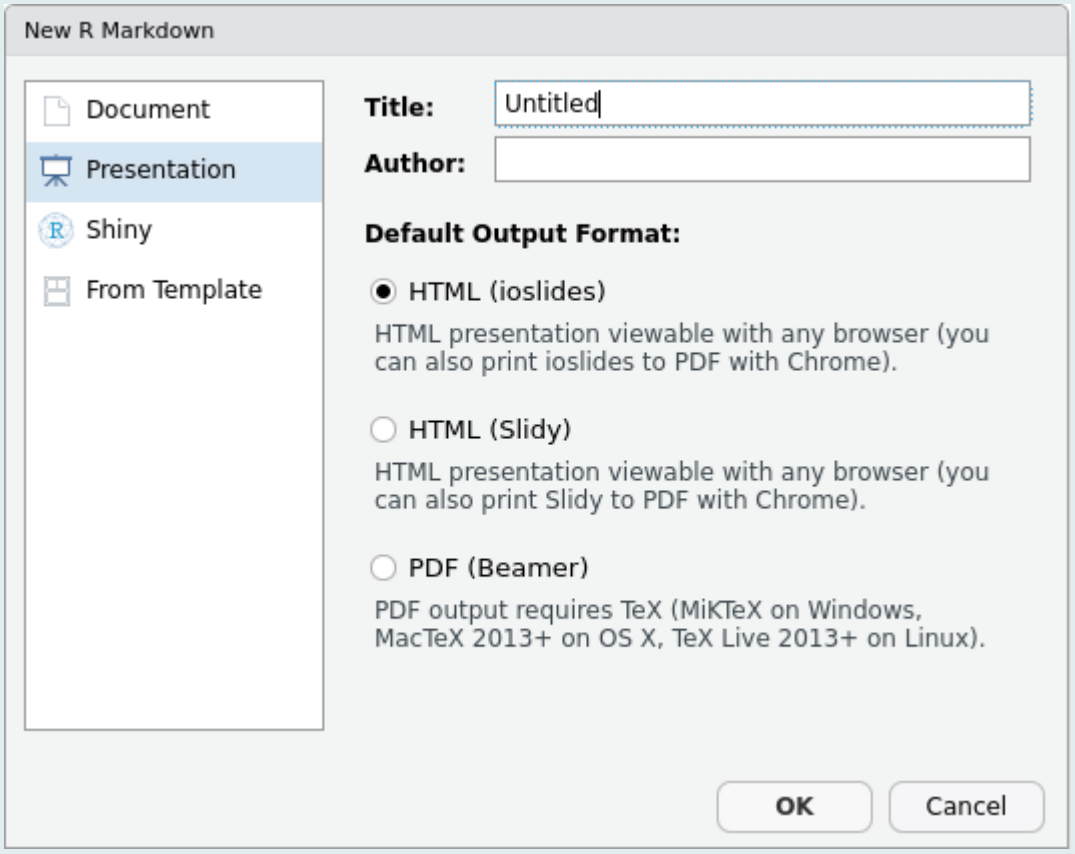
Slides

An interesting use is the creation of slideshows for presentations in the form of slides. The principle remains the same: we mix text in Markdown format and R code, and R Markdown transforms everything into presentations in HTML or PDF format. In general, the different slides are separated at certain heading levels.

Some slide templates are included with R Markdown, including:

- `ioslides` and `Slidy` for HTML presentations
- `beamer` for PDF presentations via `LaTeX`

When you create a new document in RStudio, these templates are accessible via the *Presentation* entry:

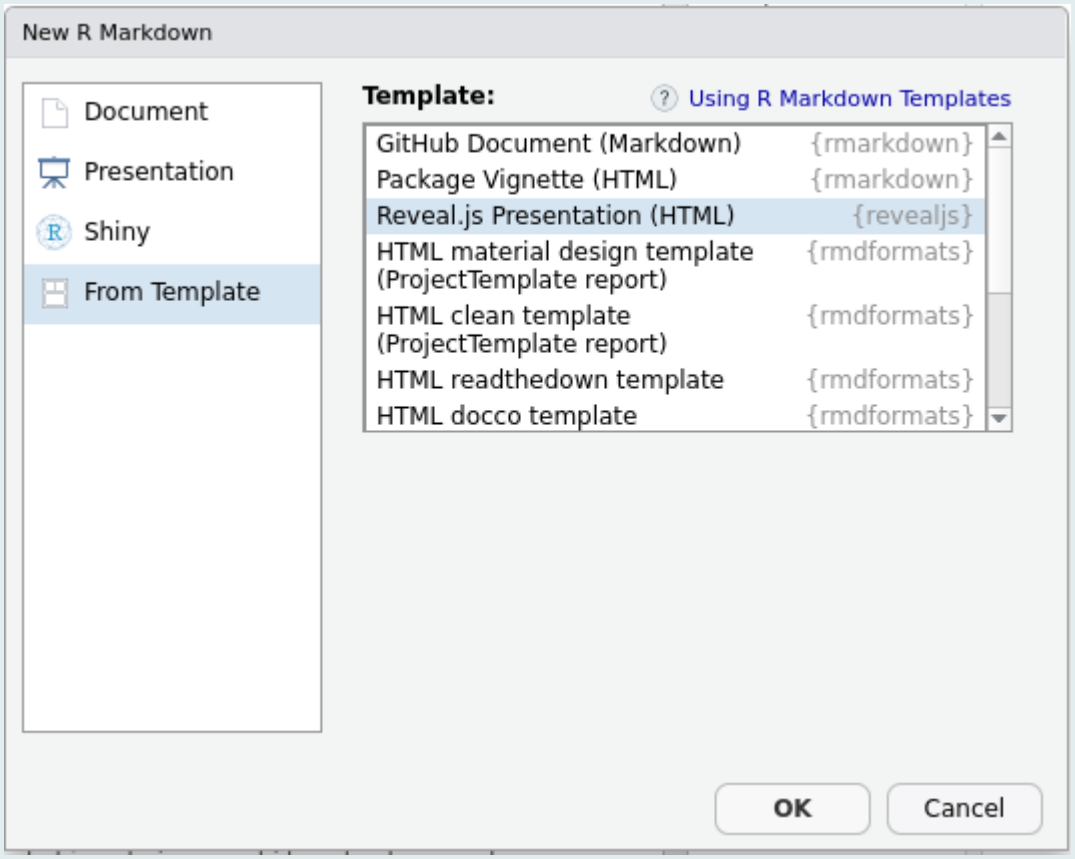


Create an R Markdown presentation

Other extensions, which must be installed separately, also allow slideshows in various formats. These include in particular:

- xaringan for HTML presentations based on remark.js
- revealjs for HTML presentations based on reveal.js
- rmdshower for HTML slideshows based on shower

Once the extension is installed, it generally offers a starting *template* when creating a new document in RStudio. These are accessible from the *From Template* entry.



Create a presentation from a template

Templates

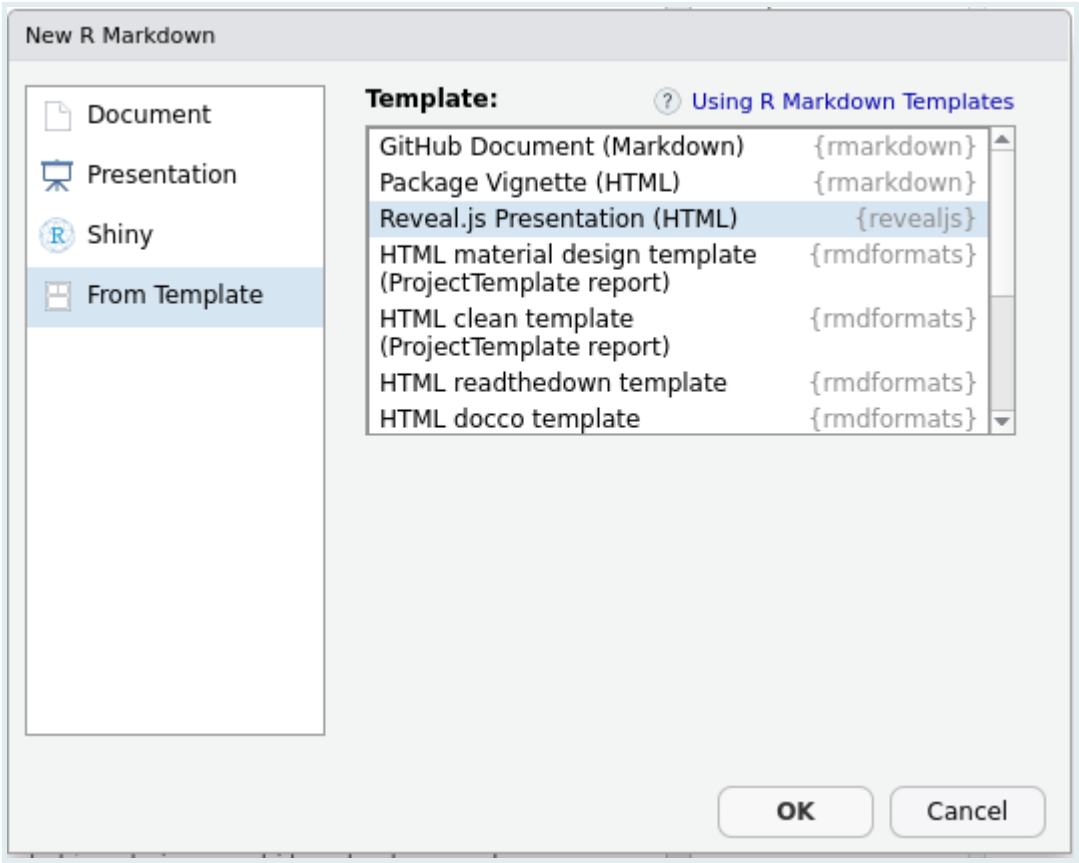
There are also different *templates* allowing you to change the format and presentation of the generated documents. A list of these formats and their associated documentation can be accessed from the formats documentation page.

Note in particular:

- the Distill format, suitable for scientific or technical publications on the Web
- the Tufte Handouts format which allows you to produce PDF or HTML documents in a format similar to that used by Edward Tufte for some of his publications
- rticles, package that offers LaTeX templates for several scientific journals

Finally, the rmdformats extension offers several HTML templates particularly suitable for long documents.

Again, most of the time, these document templates offer a starting *template* when creating a new document in RStudio (entry *From Template*):



Create a document from a template

Resources

The following resources are all in English…

The book *R for data science*, available online, contains a chapter dedicated to R Markdown.

The extension's official site contains very complete documentation, both for beginners and for advanced users.

Finally, the RStudio help (*Help* menu then *Cheatsheets*) provides access to two summary documents: a synthetic "cheat sheet" (*R Markdown Cheat Sheet*) and a more complete "reference guide" (*R Markdown Reference Guide*).

Example analysis in Rmarkdown

Now we can put the tools we just used to work!

First, create a new R Markdown Project in RStudio.

Then open a web browser, go to https://bit.ly/view-ebola-data , and download the CSV. Here are the data you need.

Open your "downloads" folder, and copy or move the CSV to the "data" folder of your new project.

Create a new R Markdown Document in this project.

Open a web browser, go to https://tinyurl.com/ebola-script , highlight the code (under "ebola-script", starting with `1 # Ebola Sierra Leone analysis` and ending with `29 num_vars_plot`), and copy that text.

Paste the text into your new RMD under the `setup` chunk.

We're going to need to find the data, so paste "data/ebola_sierra_leone.csv" inside the readcsv function.

Next we'll try running through the code to make sure it works. Click the *Knit* button above.

It didn't work! One small difficulty with R Markdown is that it has a very limited vision, and only looks in its own folder. You need to tell it more explicitly where to look. See the new package in the `# Load packages` section, `here`? Here will help us change the frame of reference so we can access our data. Change our previous attempt to `readcsv` to the following:

```
ebola_sierra_leone <- readcsv(here("data/ebola_sierra_leone.csv"))
```

Now if you *Knit*, everything should run nicely. However, the document still looks disorganized, with visible code fragments and very basic displays. But we have the tools to change it!

Move the `# Load packages ----` line and all the package loading below it up into the setup code chunk. If you try *Knit*ting again you'll see a bunch of messages spitting out from all these load statements. We don't want everyone to see that, so add `, message=FALSE` to your setup code chunk.
It's also common to do all your data loading in one place, so move your `# Load data ----` and the `readcsv` line up into the setup code chunk as well.

Next let's break this document apart into various sections.

Cut the `# Cases by district ----` line, and paste it into the white space between the code chunks. We probably don't want to show the parts where we're adding data to the tabyl, so in this code chunk add `echo = false`.

Remember how we inserted code chunks earlier? We can use that same command to break code chunks apart, too. Break this code block apart into three individual chunks by using Command+Option+i or CTRL+Alt+i between each section. Pull the other two headers outside of the code.

Knit again and see what happens! You can play around with options here to see how they change the results.

As an example you can go back to the source document and use one of the new tables you learned about. Add `flextable,` inside the `p_load(` function, and try to display `district_tab` as a flextable.

Knit once more. You did it!