R Markdown User Manual

> This manual is created as a summary guide for researchers who want to present their works in a reproducible way.
>
> As a quick-start guide, this manual only contains the basics about how to use R Markdown in reproducible research. For further study, please find the full version of this manual for more advanced learning.

1. Start

There are two ways to create an "R Markdown" file:

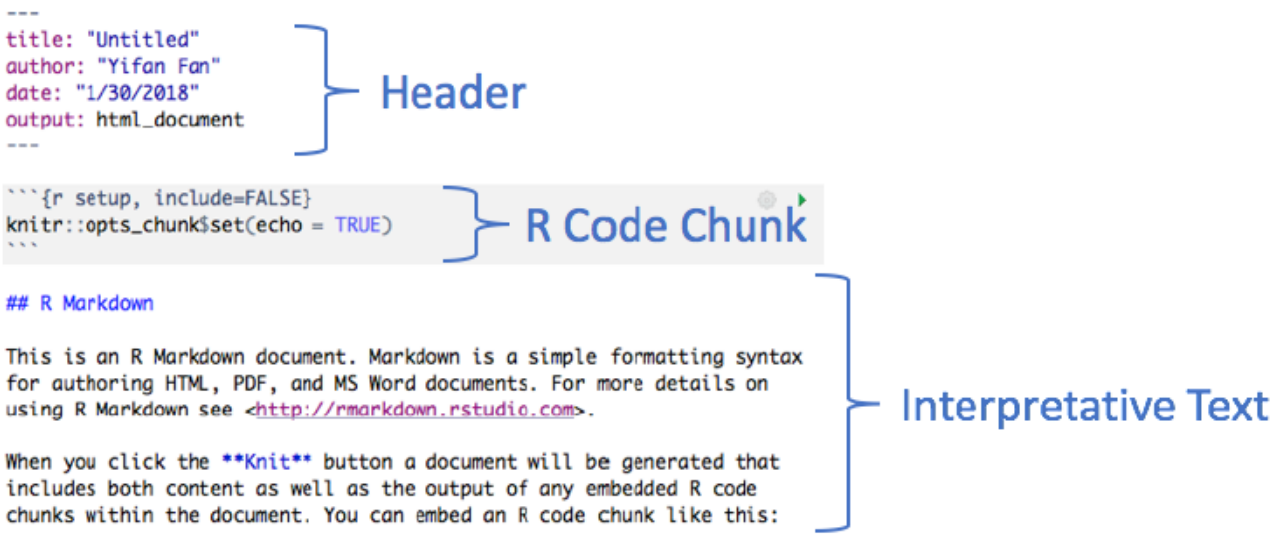
- Create a new empty file, rename its extension as `.rmd`, and then you can input any R Markdown content into this file ;
- Open RStudio, and then click `File -- New File -- R Markdown…`. You can keep the setting as default.

  In the second method, RStudio will automatically provide you an R Markdown example.

This example shows the ***three common parts*** of an R Markdown document:

1. **Header**
2. **R code chunks**
3. **Interpretative text**

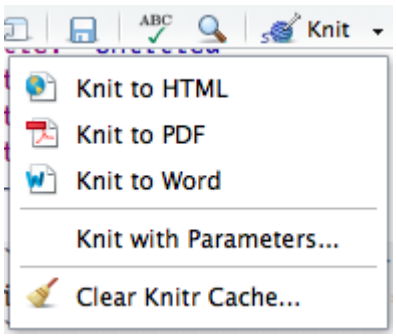
We will introduce them in this order.



2. Header

The header part follows "YAML[1]" formatting language. Basically, we have no need to change its structure.

You can define the title, author and date information in this part, but it is not necessary to manually define the output type. For example, you can keep `output: html_document` here even if you want to output it as a pdf document. When you click `Knit` button on the upper bar and select `Knit to PDF`, the document will automatically change its `output` option to `pdf_document`.



You can add some customized elements in the header to make your article more readable. A classical header template is as follows:

```
---
title:  'This is the title: it contains a colon'
subtitle: <h1><u>This is the subtitle</u></h1>
author:
- Author One^[University of A]
- Author Two^[University of B]
date: "`r format(Sys.time(), '%d %B %Y')`"
tags: [tag1, tag2]
abstract: |
  This is the abstract.

  It consists of two paragraphs.
output: pdf_document
---
```

> The header also supports user to load LaTeX packages to achieve some advanced functions, but it is too complex so that it will not be covered in this manual.

3. R Code Chunk

> You can input and run R code in an R Markdown document, just as you would do it in a common R script.

For example,

```
library(APackage)

a <- 20
b <- 18

a + b
```

To let your audience understand what is happened here, you may add some comments:

```
# load required package(s):
library(APackage)

# Now, John has 20 apples, and Dexen has 18 apples.
a <- 20
b <- 18

# How many apples do they have?
a + b
```

The question is, for readers who doesn't know how to do the addition (here we were dealing with simple addition, but we usually are dealing with more complex calculations), they have to run the code to see the result. However, R's console will only show the result without any comments. Thus when you run this code you will only see:

```
> library(APackage)
## Attaching package: 'APackage'
> a <- 20
> b <- 18
> a + b
[1] 38
```

This may confuse a reader because of the missing interpretative text. However, we can use R Markdown to show the "interpretative text" and "R code & result" together.

To do this, we need to create a `.rmd` file at first. Then we put our R code into a "R code chunk" by covering them with ```` ```{r} ```` and ```` ``` ````, and remove the `#` in front of each comment:

````
load required package(s):
```{r}
library(APackage)
```
````

````
Now, John has 20 apples, and Dexen has 18 apples.
```{r}
a <- 20
b <- 18
```

How many apples do they have?
```{r}
a + b
```
````

When you knit the code, the output document may look like:

```
load required package(s):
library(APackage)
## Attaching package: 'APackage'

Now, John has 20 apples, and Dexen has 18 apples.
a <- 20
b <- 18

How many apples do they have?
a + b
## [1] 38
```

The document now displays both "interpretative text" and "R code & result".

In most cases, the default R code chunk setting does not meet our expectations because it may show too many details. For example, we do not want the knitted document to show the "message" of `library(APackage)` and the "original codes" of the other two parts. We can use "**code chunk options**" to custimize what is displayed.

3a. Code Chunk Options

> To set options, we can add it into the chunk head ` ```{r} ` in forms of "**Option Label = Value**"

- To hide the message of an R code chunk: use ` ```{r message = FALSE} `
- To hide the original code of an R code chunk: use ` ```{r echo = FALSE} `

Thus if we rewrite the `.rmd` file like this:

````
load required package(s):
```{r message = FALSE}
library(APackage)
```

Now, John has 20 apples, and Dexen has 18 apples.
```{r echo = FALSE}
a <- 20
b <- 18
```
````

```
How many apples do they have?
```{r echo = FALSE}
a + b
```
```

the knitted document would look like:

```
load required package(s):
library(APackage)

Now, John has 20 apples, and Dexen has 18 apples.

How many apples do they have?
## [1] 38
```

There are several settings that can be used for code chunk display options including:

- Set `echo = FALSE`, R Markdown will not display the code in the final report, but it will still run the code and display the results unless told otherwise.
- Set `eval = FALSE`, R Markdown will not run the code or include the results, but it will still display the code unless told otherwise.
- Set `results = 'hide'`, R Markdown will not display the results of the code, but it will still run the code and display the code itself unless told otherwise.
- Set `message = FALSE`, R Markdown will not display the related message information.
- Set `warning = FALSE`, R Markdown will not display the related warning information.
- Set `error = FALSE`, R Markdown will not display the related error information.

(For more options setting, please see here.)

To set **more than one options**, separate them by a comma:

```
```{r echo = FALSE, message = FALSE, warning = FALSE}
library(APackage)
```
```

3b. Code Chunk Label

> We can give each code chunk an individual label to recognize them.

You can add **a word** between `r` and `chunk options` in your chunk head to name this code chunk.

```
```{r AWord, message = FALSE, warning = FALSE}
library(APackage)
```
```

A code chunk label can help you to:

- recognize different chunks
- recall any labeled code chunk by using the option `ref.label`

For example:

````
load required package(s):
```{r labelA, message = FALSE}
library(APackage)
```

Now, John has 20 apples, and Dexen has 18 apples.
```{r labelB, echo = FALSE}
a <- 20
b <- 18
```

How many apples do they have?
```{r labelC, echo = FALSE}
a + b
```

Sorry, what's the result again?
```{r ref.label = 'labelC', echo = FALSE}
```
````

the knitted document is

````
load required package(s):
library(APackage)

Now, John has 20 apples, and Dexen has 18 apples.

How many apples do they have?
## [1] 38

Sorry, what's the result again?
## [1] 38
````

> The principle of `ref.label` is to copy your code but not to show the result again.

If you write this

````
The initial a is:
```{r label1}
a = 1
```

Now let a plus one:
```{r label2, echo = FALSE}
a <- a + 1
a
```


Recall it:
````

```
```{r ref.label = 'label2', echo = FALSE}
```
```

the corresponding output is

```
The initial a is:
a = 1

Now let a plus one:
## [1] 2

Recall it:
## [1] 3
```

If you don't want to rerun your code but only want to show the result again, please refer to the usage of `cache` chunk option (not covered in this manual).

3c. In-line R Code

To embed in-line R code, just surround that code with a pair of single backticks and preface the code with a lower case `r`. For example,

```
The sum of 2017.12 and 1991.12 is `r 2017.12 + 1991.12`.
```

R Markdown will execute the code and show its result directly:

```
The sum of 2017.12 and 1991.12 is 4008.24.
```

4. Interpretative Text

> R Markdown also allows users using **Markdown** markup language to format their articles, such as changing font stules, inserting figures and tables, making bolleted or numbered lists.

4a. Markdown Basics

You can use Markdown to embed formatting instructions into text. For example, you can make a word *italicized* by surrounding it in asterisks, **bold** by surrounding it in two asterisks, and `monospaced` (commonly for showing codes in its original form) by surrounding it in backticks:

```
*italics*
**bold**
`source code`
```

To create titles and headers, use leading hashtags. The number of hashtags determines its level:

```
# First level header or titles
## Second level header or titles
### Third level header ot titles
```

To create a web link, surrounding it in hard brackets and lacing its location or addressing behind it in parentheses, like [RStudio](www.rstudio.com)

```
[RStudio](www.rstudio.com)
```

Inserting an image is similar to inserting a web link, but with an extra `!` before the label:

```
![Figure Title](image address)
```

> The image address could be either a local address or a hyper link. For example:
>
> - local address: `![Fig1](C:\Pictures\fig1.jpg)`
> - hyper link: `![Fig1](www.pictures.com/figs)`
>
> You can also leave the figure's title empty.

Sometimes we need to upload the local images to cloud drive so we can browse these images in anywhere. To do this, we need:

1. Move the local image to a cloud drive (e.g. MS OneDrive's sync folder)
2. Right click this image
3. Select `Share`. Then a pop-up window will show the share hyper link.
4. Paste the link to the image address in your R Markdown article.

   `![Figure Title](Paste the link here)`

Then you can access your images in anywhere.

4b. Lists

To make bulleted lists, add an asterisk/a hyphen and a space at the front of a new line:

```
* item 1
* item 2
* item 3
```

or

```
- item 1
- item 2
- item 3
```

You can also build an ordered list by adding a number followed by a period followed by a space at the front of a new line:

```
1. item 1
2. item 2
3. item 3
```

To make sublists within a list, indent each item by four spaces (or press `Tab`). Then add a plus sign (+) and a space at the front of a new line:

```
* item 1
    + subitem 1
    + subitem 2
* item 2
* item 3
```

You can also use an asterisk/a hyphen to make sublists. The plus sign is recommended just because of the convenience of identification when reading your Markdown code.

> Whatever the number you put in the front of the line, Markdown will automatically figure out its order, and assign the “correct” numbers for you presenting list.

Try this and see what will happen:

```
1. item 1
1. item 2
12. item 3
```

4c. Math Equations

You can also use Markdown to embed mathematical expressions into your report.

To do this, surround your equations with two pairs of dollar signs:

```
$$ 1 + 2 = 3$$
```

The syntax used to present equations is called “MathJax”. Here is a basic tutorial about MathJax can be found here.

To embed an equation in-line, surround it with a single pair of dollar signs.

```
Here is an example $1 + 2 = 3$.
```

LaTeX/MathJax can be used to present more compliated expressions.

For example: $\sigma = \sqrt{\frac{1}{N}\sum_i^N (x_i - \mu)^2}$

```
$$ \sigma = \sqrt{\dfrac{1}{N}\sum^{N}_{i}\left(x_{i}-\mu \right)^{2}} $$
```

There are some applications can help you to quickly sort the math expression code out.

For PC users: VisualMathEditor

For mobile users: MyScript-MathPad

4d. Useful Syntax

> R Markdown also supports some limited LaTeX and HTML syntax.
>
> Frequently used syntax includes:

1. Font Color

For HTML output files:

```
Roses are <span style="color:red">red</span>,
Sky is <span style="color:blue">blue</span>.
```

For LaTeX output files:

```
Roses are \textcolor{red}{red},
Sky is \textcolor{blue}{blue}.
```

2. Under Line

HTML: `This is an <u>underline</u>`

LaTeX: `This is an \underline{underline}`

3. Empty Line

HTML: `<br>`

LaTeX: `\newline`

4e. R Markdown Example

Now let us put these three sections together. Starting from our previous `.rmd` file, we reformat the R Markdown document as follows:

````
---
title:  'An Example'
subtitle: <h1><u>Using R Markdown</u></h1>
author:
- YF^[University of UWO]
date: "`2017-01-31`"
abstract: |
  This is the abstract.

  It consists of two paragraphs.
output: pdf_document
---

# section 1
Load required package(s):
```{r labelA, message = FALSE, eval = FALSE}
library(APackage)
```
There are **three** packages used in another file:

- Package 1
- Package 2
  + version 1
  + version 2
- Package 3

Now, John has 20 apples, and Dexen has 18 apples.
```{r labelB, echo = FALSE}
a <- 20
b <- 18
```

We have *three* steps to calculate this:

1. step1
2. step2
3. step3

and the equation is $20 + 18$.

How many apples do they have?
```{r labelC, echo = FALSE}
a + b
```

Sorry, what's the result again?
````

```
```{r ref.label = 'labelC', echo = FALSE}
```
```

To view what this will produce, simply knit it in RStudio.

> For more further Markdown learning, please see Official R Markdown Tutorial.

5. Using R Markdown to create as Slideshows

You can output your R Markdown file as a slideshow by setting its YAML output option to:

```
---
output: beamer_presentation
---
```

which creates a beamer pdf slideshow,

```
---
output: ioslides_presentation
---
```

which creates an ioslides HTML slideshow or

```
---
output: slidy_presentation
---
```

which creates a slidy HTML slideshow.

When you want to split your text into several slides, just **insert an additional breaks**:

```
***
```

Everywhere you add these three asterisks in your text, `Pandoc` will create a new slide.

Here is an example of an R Markdown slides:

```
---
title:  'An Example'
subtitle: <h1><u>Using R Markdown</u></h1>
author:
- YF^[University of UWO]
date: "`2017-01-31`"
abstract: |
  This is the abstract.

  It consists of two paragraphs.
output: beamer_presentation
---

# section 1
Load required package(s):
```{r labelA, message = FALSE, eval = FALSE}
library(APackage)
```
There are **three** packages used in another file:
```

```
- Package 1
- Package 2
  + version 1
  + version 2
- Package 3

***

Now, John has 20 apples, and Dexen has 18 apples.
```{r labelB, echo = FALSE}
a <- 20
b <- 18
```

We have *three* steps to calculate this:

1. step1
2. step2
3. step3

***

and the equation is $20 + 18$.

How many apples do they have?
```{r labelC, echo = FALSE}
a + b
```

Sorry, what's the result again?
```{r ref.label = 'labelC', echo = FALSE}
```
```

6. Editor Recommendation

As an alternative to using RStudio, some authors use *Typora* (click to download).

- It natively supports to edit `.rmd` files
- The interpretative text part written in Markdown syntax will **be transfered immediately** when typing, so it is a good tool to check the output layout in real time without repeatedly knitting and rewriting.
- It supports all operational platforms from Windows to Mac OS.
- Totally free.

Therefore, the basic work-flow of writing in R Markdown is:

1. Use common R scripts (`.r` files) to finish the basic R coding part, run your R codes and debug it.
2. Move the successful tested R codes into an R Markdown file (`.rmd` file).
3. Separate your R codes into several parts based on the way that you want to explain them.

   Put each part into a R code chunk environment and set its individual options.
4. Fill out the header.
5. Use Markdown syntax to finish the interpretative text.
6. Save your file, then use *Typora* to reopen it. Check its spelling, grammar and layout.
7. Save your revised file, and use RStudio to knit your `.rmd` document.

References

1. Gandrud, C. (2013). *Reproducible research with R and R studio*. CRC Press.
2. Roger D. Peng (2016). *Report Writing for Data Science in R*. Leanpub.
3. Xie, Y. (2015). *Dynamic Documents with R and knitr* (Vol. 29). CRC Press.

---

1. YAML, short for “YAML Ain’t Markup Language”, is a human-readable data serizlization language usually used for configuration files.↩