R Notes

Menghan Yuan

2025-05-13

Contents

Chapter 1

About

This is a *sample* book written in **Markdown**. You can use anything that Pandoc's Markdown supports; for example, a math equation $a^2 + b^2 = c^2$.

1.1 Usage

Each **bookdown** chapter is an .Rmd file, and each .Rmd file can contain one (and only one) chapter. A chapter *must* start with a first-level heading: `# A good chapter`, and can contain one (and only one) first-level heading.

Use second-level and higher headings within chapters like: `## A short section` or `### An even shorter section`.

The `index.Rmd` file is required, and is also your first book chapter. It will be the homepage when you render the book.

1.2 Render book

You can render the HTML version of this example book without changing anything:

1. Find the **Build** pane in the RStudio IDE, and
2. Click on **Build Book**, then select your output format, or select "All formats" if you'd like to use multiple formats from the same book source files.

Or build the book from the R console:

```
bookdown::render_book()
```

To render this example to PDF as a `bookdown::pdf_book`, you'll need to install XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): https://yihui.org/tinytex/.

1.3 Preview book

As you work, you may start a local server to live preview this HTML book. This preview will update as you edit the book when you save individual .Rmd files. You can start the server in a work session by using the RStudio add-in "Preview book", or from the R console:

```
bookdown::serve_book()
```

Chapter 2

Rstudio

Rstudio shortcuts

Command Palette: shift+cmd+P, all shortcuts can be accessed via the Command Palette.

keyboard combination	function
opt + _	insert assignment operator `<-`
ESC or ctrl + C	exit + prompt
shift + cmd + M	Add magrittr's pipe operator "%>%"After R4.1, you can set this too native pipe `|>`
ctrl + [/]	indent or unindent
cmd + D	delete one row
cmd + 1	move cursor to console window
cmd + 2	move cursor to editor window
ctrl + shift + S	add 80 hyphens `---` to signal a new chapter (Addin)
ctrl + shift + =	add 80 equals === to signal a new Chapter (Addin)
shift + cmd +N	new R script
cmd + ↑ / ↓	in console, get a list of command history
shift + ↑ / ↓	select one line up/down
fn + F2	`view()` an object, don't select the object
cmd + shift + 1	activate X11() window
ctrl (+ shift) + tab	next (last) tab in scriptor (this applies to all apps); hit ctrl first, then shift if necessary, last tab

Source

keyboard combination	function
cmd + return	Run current line/selection
opt + return	Run current line/selection (retain cursor position)

`Rmd` related

keyboard combination	function
cmd + shift + K	**Knit** rmd
cmd + opt + C	run current code chunk in `Rmd`
cmd + opt + I	insert code chunks in `Rmd`, i.e., ` ```{r} ` and ` ``` `

Q: How to print output in console rather than inline in Rmd?

A: Choose the gear □ in the editor toolbar and choose "Chunk Output in Console".

Q: How to insert Emojis in Rmd?

A: There are several options (only work for html output):

- You can type directly a lot of Emojis, such as □ and □. Try this first, if it doesn't show properly, then try the following solutions.
  - If the emoji can show in the script, then you can use it directly.

- Using a html tag, e.g., `<span> □ </span>` will show like this
  □
  This seems to be the most straightforward solution to me. □
  Note that the emoji won't disply correctly in your Rmd file, but when you render the Rmd and deploy to html pages, the emoji will show properly.

- Using Hexadecimal code. (You need to look up the code somewhere, which is a hassle. □)
  We can add emojis to an HTML document by using their hexadecimal code. These code starts with `&#x` and ends with `;` to specify browser that these are hexadecimal codes. For example,

  ```
  <p>Smily face <span>&#x1F600;</span> </p>
  ```

  will give you
  Smily face □
  Go to this site: https://emojipedia.org/emoji/
  Grab the **codepoint** for the emoji you want (e.g., `U+1F600` for grinning face)
  Replace `U+` with `&#x` so it becomes `&#x1F600`, and add a semicolon `;` at the end.
  Finally, enclose that into an html tag, e.g., `<span>`.

- With RStudio Visual mode. (You need to change mode back and forth. □)
  First change to the Visual mode. To insert an emoji, you can use either the `Insert` menu or the requisite markdown shortcut plus auto-complete:
  I am personally NOT a fan of Visual Mode because it changes your source code silently …

Set working directory

```
# get the dir name of the current script
dir_folder <- dirname(rstudioapi::getSourceEditorContext()$path)
setwd(dir_folder) # set as working dir
```

RStudio projects are associated with R working directories. You can create an RStudio project:

- In a brand new directory
- In an existing directory where you already have R code and data
- By cloning a version control (Git or Subversion) repository

Why using R projects:

1. I don't need to use `setwd` at the start of each script, and if I move the base project folder it will still work.
2. I have a personal package with a custom project, which creates my folders just the way I like them. This makes it so that the basic locations for data, outputs and analysis is the same across my work.

Double-click on a `.Rproj` file to open a fresh instance of RStudio, with the working directory and file browser pointed at the project folder.

Q: What is an **R session**? And when do I use it?

A: Multiple concurrent sessions can be useful when you want to:

- Run multiple analyses in parallel
- Keep multiple sessions open indefinitely
- Participate in one or more shared projects

Launch a new project-less RStudio session

```
# run in console
rstudioapi::terminalExecute("open -n /Applications/RStudio.app", show = FALSE)
```

`-n` Open a new instance of the application(s) even if one is already running.

`rstudioapi::terminalExecute(command, workingDir = NULL, env = character(), show = TRUE)` tells R to run the system command in quotes.

- `command` System command to be invoked, as a character string.
- `workingDir` Working directory for command
- `env` Vector of name=value strings to set environment variables
- `show` If FALSE, terminal won't be brought to front

The `rstudioapi` package provides an interface for interacting with the RStudio IDE with R code. Using`rstudioapi`, you can:

- Examine, manipulate, and save the contents of documents currently open in RStudio,
- Create, open, or re-open RStudio projects,
- Prompt the user with different kinds of dialogs (e.g. for selecting a file or folder, or requesting a password from the user),
- Interact with RStudio terminals,
- Interact with the R session associated with the current RStudio instance.

Set up Development Tools

https://cran.r-project.org/bin/macosx/tools/

- install Xcode command line tools

  ```
  sudo xcode-select --install
  ```

- install GNU Fortran compiler
  Using **Apple silicon** (aka arm64, aarch64, M1) Macs Fortran compiler
- Go to https://www.xquartz.org/, download the .dmg and run the installer.
- Verify that build tools are installed and available by opening an R console and running

  ```
  install.packages("pkgbuild")
  pkgbuild::check_build_tools()
  ```

Insert Code Session

To insert a new code section you can use the **Code** -> **Insert Section** command. Alternatively, any comment line which includes at least four trailing dashes (-), equal signs (=), or pound signs (#) automatically creates a code section.

Define your own shortcuts

https://www.statworx.com/ch/blog/defining-your-own-shortcut-in-rstudio/

https://www.r-bloggers.com/2020/03/defining-your-own-shortcut-in-rstudio/

Install the shortcut packages.

Add code session separators, `---` or `===`.

```
install.packages(
    # same path as above
  "~/Downloads/shoRtcut_0.1.0.tar.gz",
  # indicate it is a local file
  repos = NULL)
install.packages(
    # same path as above
  "~/Downloads/shoRtcut2_0.1.0.tar.gz",
  # indicate it is a local file
  repos = NULL)
```

Now go to Tools > Modify Keyboard Shortcuts and search for "dashes". Here you can define the keyboard combination by clicking inside the empty Shortcut field and pressing the desired key-combination on your keyboard. Click Apply, and that's it!

Tips and Tricks

- In `Rmd` files, send the R code chunk output to the console.
  By default, RStudio enables inline output (Notebook mode) on all R Markdown documents. You can disable notebook mode by clicking the gear button in the editor toolbar, and choosing `Chunk Output in Console`.

  To use the console by default for all your R Markdown documentsm: Tools -> Options -> R Markdown -> Show output inline for all R Markdown documents.

- To add comments to a function, you can type "Roxygen comment" into the Command Palette (shift+cmd+P) while the cursor is in a function and it will automatically add a template structure for writing a comment about your function.

  Keyboard shortcut: shift+opt+cmd+R

- Snippets are a way to make a shortcut for inserting text based on a "code".
  To find the snippets and edit them, use the Palette (`Cmd-Shift-P`) and type "edit snippets". There you will find some predefined snippets. You can also create your own.

  For instance, when in an R script (or code chunk), typing "fun" followed by pressing `Tab`, a template for a function will be inserted that looks like:

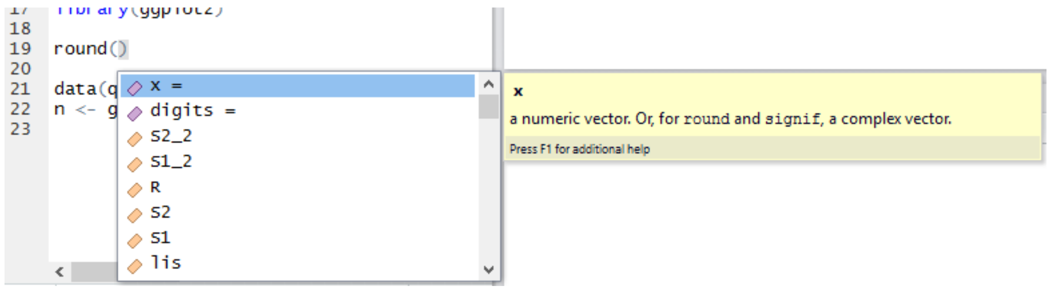
  ```
  name <- function(variables) {

  }
  ```

  You can just fill in the name of the function, then press `Tab` to move to the variables, change the name, then press `Tab` again to move to the function code area and write your function without moving your fingers from the keyboard.

- Show argument definitions as you type functions.

  When you type an existing R function such as `round(`, not only does `tab` give you the options, but there's an explanation beneath each variable, telling you its role in the function:



```
library(ggplot2)

round()

data(q
n <- g
```

x =
digits =
S2_2
S1_2
R
S2
S1
lis

x
a numeric vector. Or, for `round` and `signif`, a complex vector.
Press F1 for additional help

2.1 Dark Theme

https://community.rstudio.com/t/fvaleature-req-word-background-highlight-color-in-find-and-spellcheck/18578/3

https://rstudio.github.io/rstudio-extensions/rstudio-theme-creation.html

https://docs.posit.co/ide/user/ide/guide/ui/appearance.html#creating-custom-themes-for-rstudio

Theme repositories

- `rstudiothemes`: https://github.com/max-alletsee/rstudio-themes
- `rsthemes`: https://www.garrickadenbuie.com/project/rsthemes/

RStudio and Editor themes are two differnt things

- RStudio theme applies to the IDE's framework; including Modern (default), Classic, Sky, and Dark.
  The Sky theme is similar to the Modern theme, except for the tab and toolbar headers. 淡淡的蓝色
  The dark theme is a superset to the Modern and Sky themes that is activated whenever the Editor theme uses a dark palette.
- Editor theme applies to the source pane.
  A useful tool to customize your editor theme: https://tmtheme-editor.glitch.me/#!/editor/theme/Monokai
  Embeded themes can be found here: https://github.com/rstudio/rstudio/tree/87e129853121106a87e92df416363f39da95f82e/src/cpp/session/resources/themes

Useful elements:

`.ace_marker-layer .ace_selection` Changes the color and style of the highlighting for the currently selected line or block of lines.

`.ace_marker-layer .ace_bracket` Changes the color and style of the highlighting on matching brackets.

Recommended highlight color: `rgba(255, 0, 0, 0.47)`

If you really like one of the default themes RStudio provides, but you want to tweak some small things, you can go the theme directory and change the element's appearance.

`RStudio`'s **default editor theme directory** on Mac:

Right click `RStudio.app`, "Show Package Contents" to navigate to the application folder.

`/Applications/RStudio.app/Contents/Resources/resources/themes/ambiance.rstheme` (deprecated)

New editor theme directory: `/Applications/RStudio.app/Contents/Resources/app/resources/themes/ambiance`

You may also find the default themes on GitHub repo: https://github.com/rstudio/rstudio/tree/master/src/cpp/session/resources/themes

If you want to install or create a completely new theme, use the **Custom theme (user-defined) folder**:

- `~/.config/rstudio/themes/idle_fingers_2.rstheme` on mac
- viridis-theme

```
/* yaml tag */
.ace_meta.ace_tag {
  color: #2499DA;
}
/* quoted by $...$ and code chunk options */
.ace_support.ace_function {
  color: #55C667;
}
```

See HERE for common selectors you can use.

A collection of screenshots of default RStudio themes: https://www.trifields.jp/list-of-rstudio-editor-themes-2520

Q: The margin line is too bright.
A: Change the `.ace_print-margin` element.

```
.ace_print-margin {
  width: 1px;
  background: #e8e8e8;
}
```

`#e8e8e8` is the culprit here, and should be darkened. I changed it to `#2F3941`.

Source: https://github.com/rstudio/rstudio/issues/3420#issuecomment-453154475

Install custom themes

- Using `rstudiothemes` pkg
  Go to the repository to see which theme you want to use. Then install the theme. Themes can be applied to RStudio via "Tools" - "Global Options" - "Appearance" - "Add Theme".

  ```
  # install the pseudo-package from this Github repository
  devtools::install_github("max-alletsee/rstudio-themes")

  library(rstudiothemes) # ... then load the library

  # example 1: bulk-install all light themes
  install_rstudio_themes(theme = "all_light")

  # example 2: install two specific light themes
  install_rstudio_themes(theme = c("Ayu Light", "Github {rsthemes}"))

  # examplease 3: install one specific dark theme
  install_rstudio_themes(theme = "49th Parallel")
  ```

- Using `rstudioapi` package's "addTheme" function

  ```
  # create temporary download directory
  theme_49th_parallel <- fs::path_temp("49th_parallel-RStudio",
                                       ext = "rstheme")

  # download theme from github
  ```

```r
download.file("https://raw.githubusercontent.com/wvictor14/rstudio_themes/master/49th%20Paral
              theme_49th_parallel)

# apply the theme
rstudioapi::addTheme(theme_49th_parallel,
                     apply = TRUE)
```

2.2 Update R

Q: How to tell which version of R you are running?
A: In the R terminal, type `R.version`.

The key thing to be aware of is that when you update R, if you just download the latest version from the website, you will lose all your packages! □

On Windows use `installr`

The easiest way to update R and not cause yourself a huge headache is to use the `installr` package. When you use the `updateR()` function, a series of dialogue boxes will appear. These should be fairly self-explanatory but there is a full step-by-step guide available for how to use `installr`, simply select "Yes" when it asks if you would like to copy your packages from the older version of R.

```r
# Install the installr package
install.packages("installr")

# Load installr
library(installr)

# Run the update function
updateR()
```

On Mac, can use `updater`

The package re-installs the packages and does not copy them from the previous R installation library. R packages for minor R releases (e.g. R 4.1 to R 4.2) may not be compatible, which is why its important to re-install the packages and not copy them.

Usage:

1. Find the current location of R by running

```r
> .libPaths()
[1] "/Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/library"
```

2. Install R from https://cran.r-project.org/.

3. Install packages.

   3.1 Open your new version of R and install the `updater` package with `install.packages("updater")`.

   3.2 Run

   ```r
   updater::install_pkgs(lib.loc = c("<location(s) saved in Step 1>"))
   ```

2.3 Packages Management

2.3.1 Load packages

Q: What is the difference btw `library(package)` and `require(package)`?
A:

- `library(package)` returns an error if the package doesn't exist.
- `require(package)` returns `FALSE` if the package is not found and `TRUE` if the packages is loaded. `require` is designed for use inside other functions, such as using the value it returns in some error checking loop, as it outputs a warning and continues if the package is not found.

Q: How to reload a package after updating?
A: Call `detach(package:pkg, unload = TRUE)` or `unloadNamespace` first, then use `library(pkg)` to reload. If you use `library` on a package whose namespace is loaded, it attaches the exports of the already loaded namespace. So detaching and re-attaching a package may not refresh some or all components of the package, and is inadvisable. The most reliable way to completely detach a package is to restart R.

For example, if we want to detach `ggplot2` package, we can use

```
detach(package:ggplot2, unload=TRUE)
```

`requireNamespace` can be used to *test* if a package is installed and loadable because it comes back with either `TRUE` (if found the pkg) or `FALSE` (if failed to find the pkg).

```
> !requireNamespace("ggplot2")
[1] FALSE
> !requireNamespace("ggplot3")
Loading required namespace: ggplot3
Failed with error:  'there is no package called  'ggplot3''
[1] TRUE
```

To see whether need to install some packages:

```
# install the package if it is not available
if (!requireNamespace("devtools")) install.packages("devtools")
# or equivalently
if (!require("devtools")) install.packages("devtools")
```

You can also use `require(devtools)` to check whether the required package is available, but note that it will load the package as a side effect.

Alternatively,

```
# short command
"ggplot2" %in% installed.packages()
# full command
"ggplot2" %in% rownames(installed.packages())
```

`installed.packages()` Finds details of all packages installed in the specified library path `lib.loc`. Returns a matrix of package names, library paths and version numbers.

```
> installed.packages() %>% class()
[1] "matrix" "array"

> installed.packages() %>% str()
 chr [1:355, 1:16] "abind" "alphavantager" "anytime" "askpass" "assertthat" "backports" "ba
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:355] "abind" "alphavantager" "anytime" "askpass" ...
  ..$ : chr [1:16] "Package" "LibPath" "Version" "Priority" ...
```

The following code can be used to load packages for your project and set up the working environment.

```r
# Load the pkg, if not found, install then load
require(dplyr) || {install.packages("dplyr"); require(dplyr)}
require(odbc) || {install.packages("odbc"); require(odbc)}
require(DBI) || {install.packages("DBI"); require(DBI)}
```

If using `library()`, will return error if some package is not installed and interrupt the program.

If it is a list of packages you want to check, use `lapply` to loop through all packages.

```r
## First specify the packages of interest
packages = c("MASS", "nlme")

## Now load or install&load all
package.check <- lapply(
  packages,
  FUN = function(x) {
    if (!require(x, character.only = TRUE)) {
      install.packages(x, dependencies = TRUE)
      library(x, character.only = TRUE)
    }
  }
)
```

You can then use `search()` to determine whether all the packages have loaded.

```r
search()
 [1] ".GlobalEnv"        "package:nlme"      "package:MASS"
 [4] "package:stats"     "package:graphics"  "package:grDevices"
 [7] "package:datasets"  "renv:shims"        "package:utils"
[10] "package:methods"   "Autoloads"         "package:base"
```

Q: `dplyr` has many conflicts with `plyr`.
A: Specify pkg using `::`. Or set library priority by

- changing the order in which you load the packages.

```r
# Load dplyr last so that it has priority
library(plyr)
library(dplyr)
```

- with the {needs} package

```r
library(needs)
# prioritize the functions in dplyr
prioritize(dplyr)
```

Q: How to unload a package without restarting R?
A: `detach("package:ggplot2", unload=TRUE)` or uncheck the checkbox button in `Packages` pane.

Q: How to remove a package?
A: Use `remove.packages("dplyr")` or you can use the package manager pane, click the X mark on the right side of the selected package.

2.3.2 Install packages

Install R packages from source

```r
# From local tarball
install.packages(
  # indicate path of the package source file
  "~/Documents/R/UserPackages/shoRtcut2_0.1.0.tar.gz",
  # indicate it is a local file
  repos = NULL)

# From github
install.packages("Rcpp", repos="https://rcppcore.github.io/drat")
```

Install from GitHub

```r
devtools::install_github(repo, ref="HEAD", subdir = NULL)
```

- `repo` repository address in the format `username/repo[/subdir][@ref|#pull]`. Alternatively, you can specify `subdir` and/or `ref` using the respective parameters. If both are specified, the values in `repo` take precedence.
- `ref` Desired git reference. Could be a commit, tag, or branch name, or a call to `github_pull()` or `github_release()`. Defaults to `"HEAD"`, which means the default branch on GitHub and for git remotes.

Ex

```r
# install version 3.5.1
install_github("tidyverse/ggplot2", ref="ggplot2 3.5.1")
```

Check installed packages

```r
# print all installed packages
rownames(installed.packages())
# check if `ggplot2` is installed
"ggplot2" %in% rownames(installed.packages())
```

`installed.packages(lib.loc=NULL, priority=NULL)`

- `lib.loc` character vector describing the location of **R** library trees to search through
- `priority` used to select packages; `"high"` is equivalent to `c("base", "recommended")`

```r
# list all bases packages from your `R.Version`
> rownames(installed.packages(priority="base"))
 [1] "base"      "compiler"  "datasets"  "graphics"  "grDevices" "grid"
 [7] "methods"   "parallel"  "splines"   "stats"     "stats4"    "tcltk"
[13] "tools"     "utils"
#  what R loads on startup
> c(getOption("defaultPackages"), "base")
[1] "datasets"  "utils"     "grDevices" "graphics"  "stats"     "methods"   "base"
```

`getOption("defaultPackages")` is what R loads on startup although the `base`package is not counted.

Check package version

```r
packageVersion("ggplot2") # check package version
```

Q: How do I know if I have the latest version?
A: You can go to GitHub repo to check release notes. You will find the latest version of packages

there.

2.3.3 Update packages

- Update an individual package
  - Using `install.packages`

    ```r
    install.packages("ggplot2") # update one specific package
    ```

  - Using `update.packages`

    ```r
    update.packages(oldPkgs = "ggplot2")
    ```

    Note that you need to specify `oldPkgs` explicily as it is a named argument.

- Update ALL outdated packages

  ```r
  ## update all installed packages in a stated library location, default to `.libPaths()`
  update.packages(lib.loc = .libPaths(), ask = TRUE)
  ```

  `update.packages` updates ALL outdated packages in a stated library location. That library location is given by the first argument (if not supplied it works on all known library locations for the current R session).
  It will ask you for every package if you want to update.
  To just say `yes` to everything, use `ask = FAlSE`.

  ```r
  update.packages(ask = FALSE)
  ```

  Unfortunately this won't update packages installed by `devtools::install_github()`

Troubleshooting

Q: I ran `update.packages("ggplot2")`, but nothing happened. No output on console, no error, nothing.
A: The first argument specifies the library location you want to search through (and update packages therein). `update.packages("ggplot2")` means you want to update the packages in library location `ggplot2`, which is most unlikely to exist on your R installation.

Q: I tried to update `ggplot2` with `install.packages("ggplot2")`, but nothing happened.
A: If `ggplot2` is already loaded, then you can't install `ggplot2` in the current session now. If you need to, save any objects you can't easily recreate, and quit out of R. Then start a new R session, immediately run `install.packages("ggplot2")`, then once finished, load the package and reload in any previously saved objects.

More about `update.packages`:

- `update.packages(lib.loc = NULL, repos = getOption("repos"), ask = TRUE)`: First a list of all packages found in `lib.loc` is created and compared with those available at the repositories. If `ask = TRUE` (the default) packages with a newer version are reported and for each one the user can specify if it should be updated. If so the packages are downloaded from the repositories and installed in the respective library path (or `instlib` if specified).

- You can specify one specific package to update using `update.packages(oldPkgs = "ggplot2")`. It will check updates only for that package and ask you if you want to update.

  The easiest way to update an individual package is just to use `install.packages`. It is a one step command, compared to `update.packages`, which first checks and then asks.

- `update.packages` returns NULL invisibly.
- Be aware that some package updates may cause your previous code to stop working. For this reason, we recommend updating all your packages once at the beginning of each academic year (or semester) – don't do it before an assessment or deadline just in case!

Reinstall all Packages after R update

R packages are missing after updating. So you have to save the installed packages and re-install them after updating.

- Alternatively, `updater` automatically reinsatll R pakages. □

Here is how to do it manually.

```
## get packages installed
packs <- as.data.frame(installed.packages(.libPaths()[1]), stringsAsFactors = F)
# Save to local
f_name <- "~/Documents/R/packages.csv"
rownames(packs)
write.csv(packs, f_name, row.names = FALSE)
packs <- read_csv(f_name)
packs
## Re-install packages using install.packages() after updating R
install.packages(packs$Package)
```

R library path `/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library`

- use `find.package("ggplot2")` to find the location to where the given package is found.
- alternatively, you can run `.libPaths()`
  - `.libPaths()` without an argument will return a list of all the places R will currently look for loading a package when requested.
  - `.libPaths("a/b/c")` with an argument will add that new directory (`"a/b/c"`) to the ones R was already using. If you use that directory often enough, you may wish to add that call to `.libPaths("a/b/c")` in your `.Rprofile` startup file in your home directory.

2.3.4 Put your R package on GitHub

Reference: https://jennybc.github.io/2014-05-12-ubc/ubc-r/session2.4_github.html

- Change to the package directory
- Initialize the repository with `git init`
- Add and commit everything with
  1. `git add .` stage changes;
  2. `git status` optional check staged changes, but yet to submit;
  3. and `git commit` submit staged changes.
- Create a new repository on GitHub
- Connect your local repository to the GitHub one

  ```
  # add repo name "origin" to the remote repo at the URL
  git remote add origin https://github.com/username/reponame
  ```

- Push everything to github

```
# rename the current local branch to "main"
git branch -M main
# creates a remote branch "origin" and sets it upstream of the "main" branch
git push -u origin main
```

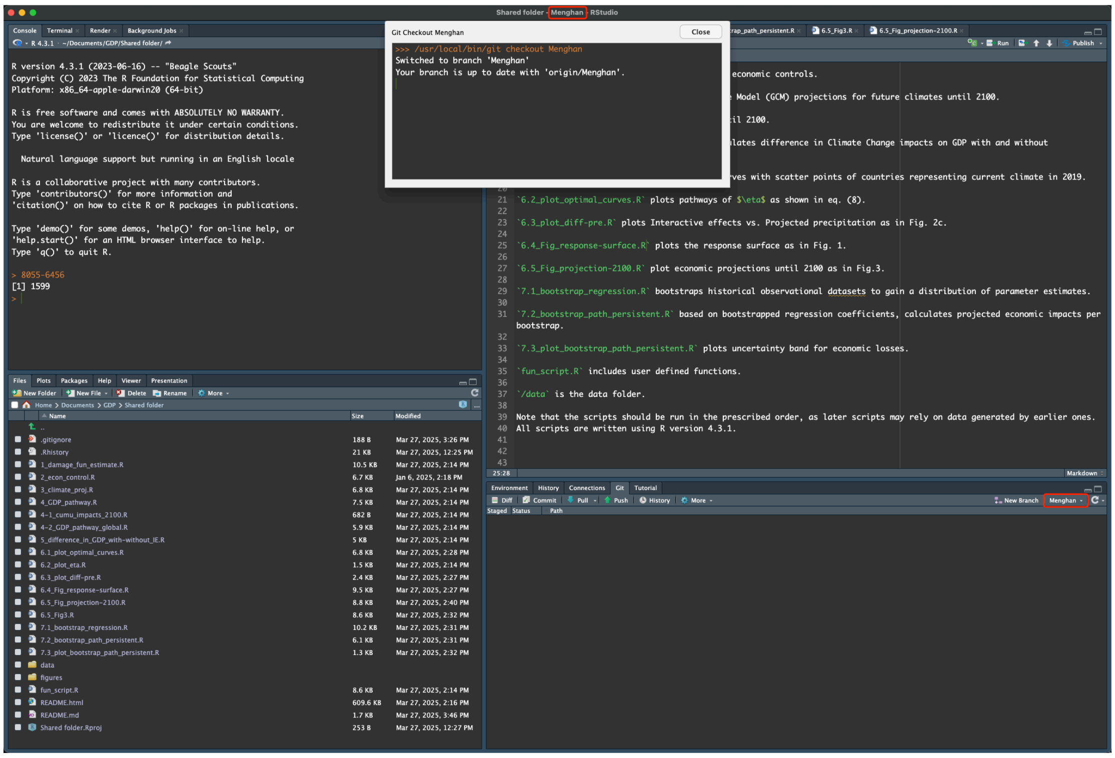
FAQ

Q: What are package vignettes?
A: It's important to write good and clear documentation, but users don't often read it; at best they'll look at the examples, so be sure to include informative examples. In my experience, what users really want are instructive tutorials demonstrating practical uses of the software with discussion of the interpretation of the results. In R packages, such tutorials are called "vignettes."

2.4 Using Git with RStudio

Before you start coding, make sure that you are on the correct branch. You may check

- from the Git tab on the Environment, History, Connections, … pane
- you can also see from the status bar on the very top of the window. The words are formatted as "Projection Name – Branch – RStudio".



```
>>> /usr/local/bin/git checkout Menghan
Switched to branch 'Menghan'
Your branch is up to date with 'origin/Menghan'.
```

```
R version 4.3.1 (2023-06-16) -- "Beagle Scouts"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin20 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 8055-6456
[1] 1599
>
```

```
economic controls.

Model (GCM) projections for future climates until 2100.

il 2100.

lates difference in Climate Change impacts on GDP with and without

ves with scatter points of countries representing current climate in 2019.

`6.2_plot_optimal_curves.R` plots pathways of $\eta$ as shown in eq. (8).

`6.3_plot_diff-pre.R` plots Interactive effects vs. Projected precipitation as in Fig. 2c.

`6.4_Fig_response-surface.R` plots the response surface as in Fig. 1.

`6.5_Fig_projection-2100.R` plot economic projections until 2100 as in Fig.3.

`7.1_bootstrap_regression.R` bootstraps historical observational datasets to gain a distribution of parameter estimates.

`7.2_bootstrap_path_persistent.R` based on bootstrapped regression coefficients, calculates projected economic impacts per
bootstrap.

`7.3_plot_bootstrap_path_persistent.R` plots uncertainty band for economic losses.

`fun_script.R` includes user defined functions.

`/data` is the data folder.

Note that the scripts should be run in the prescribed order, as later scripts may rely on data generated by earlier ones.
All scripts are written using R version 4.3.1.
```

Name	Size	Modified
..		
.gitignore	188 B	Mar 27, 2025, 3:26 PM
.Rhistory	21 KB	Mar 27, 2025, 12:25 PM
1_damage_fun_estimate.R	10.5 KB	Mar 27, 2025, 2:14 PM
2_econ_control.R	6.7 KB	Jan 6, 2025, 2:18 PM
3_climate_proj.R	6.8 KB	Mar 27, 2025, 2:14 PM
4_GDP_pathway.R	7.5 KB	Mar 27, 2025, 2:14 PM
4-1_cumu_impacts_2100.R	682 B	Mar 27, 2025, 2:14 PM
4-2_GDP_pathway_global.R	5.9 KB	Mar 27, 2025, 2:14 PM
5_difference_in_GDP_with-without_IE.R	5 KB	Mar 27, 2025, 2:14 PM
6.1_plot_optimal_curves.R	6.8 KB	Mar 27, 2025, 2:28 PM
6.2_plot_eta.R	1.5 KB	Mar 27, 2025, 2:14 PM
6.3_plot_diff-pre.R	2.4 KB	Mar 27, 2025, 2:27 PM
6.4_Fig_response-surface.R	9.5 KB	Mar 27, 2025, 2:27 PM
6.5_Fig_projection-2100.R	8.8 KB	Mar 27, 2025, 2:40 PM
6.5_Fig3.R	8.6 KB	Mar 27, 2025, 2:32 PM
7.1_bootstrap_regression.R	10.2 KB	Mar 27, 2025, 2:31 PM
7.2_bootstrap_path_persistent.R	6.1 KB	Mar 27, 2025, 2:31 PM
7.3_plot_bootstrap_path_persistent.R	1.3 KB	Mar 27, 2025, 2:32 PM
data		
figures		
fun_script.R	8.6 KB	Mar 27, 2025, 2:14 PM
README.html	609.6 KB	Mar 27, 2025, 2:16 PM
README.md	1.7 KB	Mar 27, 2025, 3:46 PM
Shared folder.Rproj	253 B	Mar 27, 2025, 12:27 PM

Choose a License for your repo

Q: Which open source license is appropriate for my project?
A: See https://opensource.guide/legal/#which-open-source-license-is-appropriate-for-my-project.

Q: How to add a license to my repo?
A: Follow the instructions here.

2.5 Copilot

Copilot offers autocomplete-style suggestions as you code as "ghost text".

GitHub Copilot primarily relies on the context in the file you are actively editing. Any comments, code, or other context provided within the active document will be used as a "prompt" that Copilot will then use to provide a suggested completion.

- To expand the scope of the context used by Copilot beyond just the active document, there is a setting to also index and read from other R, Python, or SQL files in the current project. This setting can be toggled on or off in the Tools > Global Options > Copilot > "Index project files with GitHub Copilot" setting.

- At times, normal autocomplete and Copilot may seem to conflict with each other. In these cases, it is best to review the Copilot suggestion and determine if it is appropriate for the current context. If it is, you can accept the suggestion by pressing `Tab`. If it is not, you can ignore the suggestion and continue typing or force the normal autocomplete to show by pressing `Ctrl+Space`.
  Issue: `Ctrl+Space` crash with spotlight shortcut. □

- Only show Copilot suggestions when evoke mannually using `Ctrl + Backslash (\)`.
  The Copilot suggestions can be very distracting and clutter your script.

For general advice on how to use Copilot, please see:

- RStudio Copilot User Guide

- How to use GitHub Copilot: Prompts, tips, and use cases

2.6 Save R Workspace

If you want to saves all objects in your work space, use `save.image()`. It will creates an image of your current variables and functions, and saves them to a file called `.RData`. When R next loads, objects stored in this image are by default restored.

This sounds convenient, however, you do NOT want to do this because this corrupt reproducibility of your project. □

You want to start from a clean slate very time. □

It is suggested change RStudio Global Options to

- *not* "restore `.RData` into workspace at startup", and

- *never* "save workspace to `.RData` on exit".

In case you do feel the need to save the workspace, use the following cmd.

```
save.image(file = ".RData", version = NULL, ascii = FALSE, compress = !ascii,
safe = TRUE)
```

```r
## save current workspace ##
f_name <- "RImage/TCR_2023-05-09.RData"
f_name
save.image(f_name)
# load(f_name)
```

Q: Can I save the loaded packages in the current session/workspace?
A: The workspace is for *objects* like data and functions. Starting R with particular packages loaded is what your `.Rprofile` file is for, and you can have a different one in each directory. But I'd recommend not saving anything between r sessions and instead recreate it all using code. This is much more likely to lead to reproducible results.

History

When you quit a project, `.Rhistory` is automatically written to the project directory unless you opt out to. It contains a history of all of the commands that you have sent to the R console in this session.

2.7 Pane Layout

Pop out an editor

Click the `Show in New Window` button in any source editor tab.

To return a document to the main window, click the `Return to Main Window` button on the editor toolbar.

Environment Pane

By default, the Environment pane is located in the top-right and includes the Environment, History, Connections, Build, and Version Control System (VCS) tabs.

Version Control System (VCS)

The VCS tab will change based on the version control system you have enabled for that session. For example, using Git will change the tab name to Git and provide some common commands for viewing diffs, committing changes, pull and push … Output pane

The Output pane displays various outputs such as plots, HTML content, or on-disk files. It contains the Files, Plots, R Packages, Help, Viewer, and Presentation tabs.

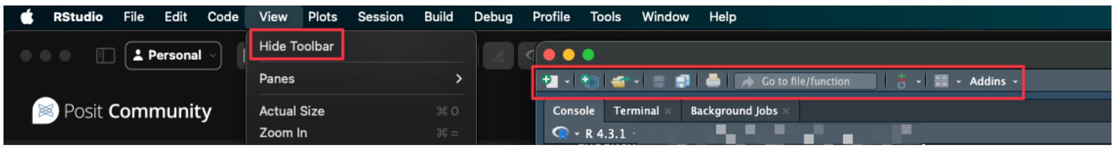
Ref: RStudio Pane Layout

Global Options that make coding easier

- Syntax highlight and matched parentheses.
  Under "Tools -> Global Options -> Code -> Display", under **Syntax section**, check the boxes for **highlight R function calls** and **use rainbow parentheses**. The second is especially useful to mark matching opening and closing brackets.

- Show whitespace characters.
  In "Tools -> Global Options -> Code -> Display", check "Show whitespace characters". This will let you see spaces and newlines in the editor.

Q: How to show Toolbar?
A: View > Show Toolbar.



References:

https://coding-club.rostools.org/posts/tips-and-tricks/

2.8 Options

`getOption(x)` Allow the user to set and examine a variety of global *options* which affect the way in which **R computes and displays its results.** Use `getOption` to check default values of global options.

- `x` a character string holding an option name, must be quoted in quotes
- Can only query one option at a time. If multiple options are given, will return the value of the first option.

`options(...)` query and modify global options.

- `...` any options can be defined, using `name = value`.
  Note that you do NOT need to quote your option name here!
- `options()` with no arguments returns a list with the current values of the options.
- `options("name")` can be used to examine options' current value too; return a *list*, whereas `getOption("name")` returns the value only.
  - Note that you need to quote the option name when you do queries.
  - You can query more than one options at a time.

```
> options("width", "digits")
$width
[1] 90

$digits
[1] 7

> getOption("width", "digits")
[1] 90
```

`?options` to get the help page of global options. To check which options are available and their definitions.

Use examples

```
## Two ways checking default option values
> options("width")
$width
[1] 81

> getOption("width")
[1] 81

## Change option values
# use name=value
> options(width=80, digits=15) # set print width, digits to print for numeric values using
# use a named list
> options(list(width=80, digits=15))
```

Commonly used global options:

Option	Description
width	Controls the maximum number of columns on a line used in printing vectors, matrices and arrays, and when filling by `cat`. Defaults to 80.Don't change this if you want to print more columns. Use `options(tibble.width=400)` instead.

Option	Description
pillar.sigfig	Tibbles print numbers with **three** significant digits by default, switching to scientific notation if the available space is too small.`options(pillar.sigfig = 4)` to increase the number of digits printed out.

2.9 R Startup

`Sys.getenv(x)` get the values of the environment variables. Returns a vector of the same length as `x`.

- `x` a character vector

Environment Variables examples:

```
> Sys.getenv(c("HOME", "R_HOME", "R_PAPERSIZE", "R_PRINTCMD"))
             HOME                                     R_HOME
"/Users/menghan" "/Library/Frameworks/R.framework/Resources"
      R_PAPERSIZE                                 R_PRINTCMD
             "a4"                                      "lpr"
```

Rstudio doesnn't load Rprofile or Renviron

I store my `Rprofile` and `Renviron` in non-default places (i.e. `~/.config/R`). When opening `R` in a normal shell, my environment is loaded perfectly fine. When opening Rstudio, it doesn't load my options, settings or paths.

- Have to wrap your option settings in `rstudio.sessionInit`
  https://damien-datasci-blog.netlify.app/post/2020-12-31-pimp-your-r-startup-message/
  - Open `.Rprofile`

    ```
    usethis::edit_r_profile()
    ```

  - wrap up your options in the following snippet

    ```
    setHook("rstudio.sessionInit", function(newSession) {
      # any code included here will be run at the start of each RStudio session
      options(buildtools.check = function(action) TRUE )
    }, action = "append")
    ```

- Understanding R's startup
  https://rviews.rstudio.com/2017/04/19/r-for-enterprise-understanding-r-s-startup/
  https://docs.posit.co/ide/user/ide/guide/environments/r/managing-r.html

`usethis` is a workflow package: it automates repetitive tasks that arise during project setup and development, both for R packages and non-package projects.

2.9.1 `.Rprofile`

What is `.Rprofile`?

`.Rprofile` is a startup file to set **options** and **environment variables**. `.Rprofile` files can be either at the user or project level.

- User-level `.Rprofile` files live in the base of the user's home directory, and
- project-level `.Rprofile` files live in the base of the project directory.

R will source only one `.Rprofile` file. If there is a project-level `.Rprofile`, the user-level file will NOT be sourced, i.e., the project-level config file take priority.

So if you have both a project-specific `.Rprofile` file and a user `.Rprofile` file that you want to use, you explicitly source the user-level `.Rprofile` at the top of your project-level `.Rprofile` with `source("~/.Rprofile")`.

`.Rprofile` files are sourced as regular R code, so setting environment variables must be done inside a `Sys.setenv(key = "value")` call.

Quitting R will erase the default theme setting. If you load `ggplot2` in a future session it will revert to the default gray theme. If you'd like for `ggplot2` to always use a different theme (either yours or one of the built-in ones), you can set a load hook and put it in your `.Rprofile` file. For example, the following hook sets the default theme to be `theme_minimal()` every time the `ggplot2` package is loaded.

```r
setHook(packageEvent("ggplot2", "onLoad"),
        function(...) ggplot2::theme_set(ggplot2::theme_bw()))
```

Of course, you can always override this default theme by adding a theme object to any of your plots that you construct in `ggplot2`.

2.9.2 .Renviron

`.Renviron` is a user-controllable file that can be used to create **environment variables**. This is especially useful to avoid including credentials like API keys inside R scripts. This file is written in a key-value format, so environment variables are created in the format:

```
Key1=value1
Key2=value2
...additional key=value pairs
```

And then `Sys.getenv("Key1")` will return `"value1"` in an R session.

Like with the `.Rprofile` file, `.Renviron` files can be at either the user or project level. If there is a project-level `.Renviron`, the user-level file will not be sourced. The usethis package includes a helper function for editing `.Renviron` files from an R session with `usethis::edit_r_environ()`.

The `.Renviron` file is most useful for defining sensitive information such as API keys (such as GitHub, Twitter, or Posit Connect) as well as R specific environment variables like the history size (`R_HISTSIZE=100000`) and default library locations `R_LIBS_USER`.

Rcpp compilation breaks in R 4.1.0

```
devtools::build("my_package")
Error: Could not find tools necessary to compile a package
Call `pkgbuild::check_build_tools(debug = TRUE)` to diagnose the problem.
```

- In RStudio, I am continually prompted to install additional build tools and I can't install the build tool. $\rightarrow$ Bypass the option `options(buildtools.check = function(action) TRUE)`.
- Turns out R was pointing to an old clang version in my Makevars.
  I just deleted it using [in Terminal]

  ```bash
  sudo rm ~/.R/Makevars
  ```

Install SDK command line tool

Download from developer.apple.com. Software development kit.

https://developer.apple.com/download/all/

R compiler tools for cpp on MacOS

- https://thecoatlessprofessor.com/programming/cpp/r-compiler-tools-for-rcpp-on-macos/
- install OpenMP enabled `clang` from the terminal
  https://rpubs.com/Kibalnikov/776164

Chapter 3

Rmd

R Markdown is a powerful tool for combining analysis and reporting into the same document. R Markdown has grown substantially from a package that supports a few output formats, to an extensive and diverse ecosystem that supports the creation of books, blogs, scientific articles, websites, and even resumes.

Nice documentations

- **`rmarkdown`** package CRAN
  - Package CRAN page
  - Reference manual
- **`bookdown`** package CRAN
  - Package CRAN page
  - Reference manual
- R markdown: The definitive guide. provides detailed references; GitHub repo HERE.
- R markdown cookbook: concise and covers essential functions, with examples.
- Authoring Books with R Markdown: with a focus on `bookdown`.

Q: What is the difference between Rmd and R script?
A:

- An R script (`.R`) is used for developing and troubleshooting code; a place where you can store reusable code fragments.
- An R Markdown file (`.Rmd`) is used to integrate R commands with explanatory text and output, making it useful for creating reports.

Quick takeaways:

- Can still use horizontal separator `ctrl + shift + S` for dashed lines and `ctrl + shift + =` for equals
- Headers must have one empty line above and below to separate it from other text

---

3.1 YAML metadata

Q: What is YAML?
A: YAML is a human-friendly data serialization language for all programming languages. YAML stands for "Yet Another Markup Language."

Q: What does YAML do?
A: It is placed at the very beginning of the document and is read by each of Pandoc, **rmarkdown**, and **knitr**.

- Provide metadata of the document.
- located at the top of the file.
- adheres to the YAML format and is delimited by lines containing three three dashes (`---`).

YAML also called header and front matter.

See *R Markdown YAML metadata (header) tutorial with examples* by `hao203` HERE for commonly used YAML metadata (header) in different R Markdown output formats.

YAML can set values of the template variables, such as `title`, `author`, and `date` of the document.

- The `output` field is used by rmarkdown to apply the **output format function** `rmarkdown::html_document()` in the rendering process.

  There are two types of output formats in the **rmarkdown** package: documents (e.g., `pdf_document`), and presentations (e.g., `beamer_presentation`).

  Supported output format examples: `html_document`, `pdf_document`.

  R Markdown documents (`html_documents`) and R Notebook documents (`html_notebook`) are very similar; in fact, an R Notebook document is a special type of R Markdown document. The main difference is using R Markdown document (`html_documents`) you have to knit (render) the entire document each time you want to preview the document, even if you have made a minor change. However, using an R Notebook document (`html_notebook`) you can view a preview of the final document without rendering the entire document.

  Troubleshooting

  Issue: `bookdown` always output html, even if specified to pdf.
  Cause: If it produces HTML, the output format must have been provided somewhere.
  Fix: Check if you have a `_output.yml` under the root directory of your book project. If you do, you may delete it. Then bookdown will use the output field that you specified in the YAML frontmatter of your Rmd document.

  If there are two output formats, `rmarkdown::render()` defaults to use the first output type. If you want another, specify the type, e.g., `rmarkdown::render("0100-RStudio.Rmd", 'pdf_document')`.

`bookdown` wrappers of base markdown format

`bookdown` output formats allow numbering and cross-referencing figures/tables/equations. It takes the format `html_document2`, in general, `markdown_document2` is a wrapper for the base format `markdown_document`. With the `bookdown` output format, you can cross-reference sections by their ID's using the same syntax when sections are numbered.

Other bookdown output format examples for single documents: `pdf_document2`, `beamer_presentation2`, `tufte_html2`, `word_document2`. See Page 12 of the reference manual for a complete list of supported format by `bookdown`.

What bookdown is very powerful for is that it compiles books. Book formats:

- HTML:
  - `gitbook`
  - `html_book`
  - `tufte_html_book`
- PDF:
  - `pdf_book`
- e-book:

  - `epub_book`

- Many aspects of the LaTeX template used to create PDF documents can be customized using **top-level** YAML metadata (note that these options do **NOT** appear underneath the `output` section, but rather appear at the top level along with `title`, `author`, and so on). For example:

```
---
title: "Crop Analysis Q3 2013"
output: pdf_document
fontsize: 11pt
geometry: margin=1in
---
```

  A few available metadata variables are displayed in the following (consult the Pandoc manual for the full list):

Top-level YAML Variable	Description
`lang`	Document language code
`fontsize`	Font size (e.g., `10pt`, `11pt`, or `12pt`)
`papersize`	Defines the paper size (e.g., `a4paper`, `letterpaper`)
`documentclass`	LaTeX document class (e.g., `article`, `book`, and `report`)
`classoption`	A list of options to be passed to the document class, e.g., you can create a two-column document with the `twocolumn` option.
`geometry`	Options for `geometry` package (e.g., `margin=1in` set all margins to be 1 inch)
`mainfont`, `sansfont`, `monofont`, `mathfont`	Document fonts (works only with `xelatex` and `lualatex`)
`linkcolor`, `urlcolor`, `citecolor`	Color for internal links (cross references), external links (link to websites), and citation links (bibliography)
`linestretch`	Options for line spacing (e.g. 1, 1.5, 3).

  Pandoc User's Guide: https://www.uv.es/wiki/pandoc_manual_2.7.3.wiki?21

  `classoption`

  - **`onecolumn`, `twocolumn`** - Instructs LaTeX to typeset the document in one column or two columns.
  - **`twoside`, `oneside`**: Specifies whether double or single sided output should be generated. The classes' article and report are single sided and the book class is double sided by default.

    Note that this option concerns the style of the document only. The option two side does *NOT* tell the printer you use that it should actually make a two-sided printout.

    The difference between single-sided and double-sided documents in LaTeX lies in the layout of the page margins and the orientation of the text on the page.

    ⋆ Single-sided documents are printed on only one side of the page, with the text and images aligned to the right-hand side of the page. This type of layout is often used for brochures, flyers, and other types of promotional materials.

    ⋆ Double-sided documents are printed on both sides of the page, with the text and images alternating between right-hand and left-hand margins. This type of layout is often used for **books**, reports, and other types of long-form documents.

      A `twoside` document has different margins and headers/footers for odd and even pages.

The layout of a `twoside` book

Q: Why Inner margin is narrow?
A: The reason for this is that with two pages side by side, you actually have only THREE margins - the left, right and middle. The middle margin is made up from the inside margins of both pages, and so these are smaller because they add together to make the middle margin. If they were bigger, then you would end up with too much whitespace in the middle.

```
o - outside margin
i - inside margin
b - binding offset

Before binding:
------------------    -----------------
|oooo|~~~~~~|ii|b|    | |  |~~~~~~|   |
|    |~~~~~~|  | |    | |  |~~~~~~|   |
|    |~~~~~~|  | |    | |  |~~~~~~|   |
|    |~~~~~~|  | |    | |  |~~~~~~|   |
|    |~~~~~~|  | |    | |  |~~~~~~|   |
|    |~~~~~~|  | |    | |  |~~~~~~|   |
------------------    -----------------

After binding:
-------------------------------
|oooo|~~~~~~|ii|ii|~~~~~~|oooo|
|oooo|~~~~~~|ii|ii|~~~~~~|oooo|
|oooo|~~~~~~|ii|ii|~~~~~~|oooo|
|oooo|~~~~~~|ii|ii|~~~~~~|oooo|
|oooo|~~~~~~|ii|ii|~~~~~~|oooo|
|oooo|~~~~~~|ii|ii|~~~~~~|oooo|
-------------------------------
```

- **`landscape`** - Changes the layout of the document to print in landscape mode.
- **`openright`, `openany`** - Makes chapters begin either only on right hand pages or on the next page available. This does not work with the article class, as it does not know about chapters. The report class by default starts chapters on the next page available and the book class starts them on right hand pages.

• In PDFs, you can use code, typesetting commands (e.g., `\vspace{12pt}`), and specific packages from LaTeX.

1. The `header-includes` option loads LaTeX packages.
   Note that `header-includes` is a top-level option that align with `output`.

```
---
output: pdf_document
header-includes:
  - \usepackage{fancyhdr}
---

\pagestyle{fancy}
\fancyhead[LE,RO]{Holly Zaharchuk}
\fancyhead[LO,RE]{PSY 508}

# Problem Set 12
```

Common `header-includes`:

- Chinese/Japanese support

```
---
output: pdf_document
header-includes:
  - \usepackage{ctex}
---
```

2. Alternatively, use `extra_dependencies` to list a character vector of LaTeX packages. This is useful if you need to load multiple packages:

```
---
title: "Untitled"
output:
  pdf_document:
    extra_dependencies: ["bbm", "threeparttable"]
---
```

If you need to specify options when loading the package, you can add a second-level to the list and provide the options as a list:

```
---
title: "Untitled"
output:
  pdf_document:
    extra_dependencies:
      caption: ["labelfont={bf}"]
      hyperref: ["unicode=true", "breaklinks=true"]
      lmodern: null
---
```

Here are some examples of LaTeX packages you could consider using within your report:

- pdfpages: Include full PDF pages from an external PDF document within your document.
- caption: Change the appearance of caption subtitles. For example, you can make the figure title italic or bold.
- fancyhdr: Change the style of running headers of all pages.

• Some output options are passed to Pandoc, such as `toc`, `toc_depth`, and `number_sections`. You should consult the Pandoc documentation when in doubt.

```
---
output:
  pdf_document:
    toc: true
    keep_tex: true
---
```

- `keep_tex: true` if you want to keep intermediate TeX. Easy to debug. Defaults to `false`.

To learn which arguments a format takes, read the format's help page in R, e.g. **`?html_document`**.

Parameters

We can include variables and R expressions in this header that can be referenced throughout our R Markdown document. For example, the following header defines `start_date` and `end_date` parameters, which will be reflected in a list called `params` later in the R Markdown document.

```
---
title: My RMarkdown
author: Yihui Xie
output: html_document
params:
  start_date: '2020-01-01'
  end_date: '2020-06-01'
---
```

To access a parameter in our R code, call `params$<parameter name>`, e.g., `params$start_date` and `params$end_date`.

Should I use quotes to surround the values?

- Whenever applicable use the unquoted style since it is the most readable.
- Use quotes when the value can be misinterpreted as a data type or the value contains a `:`.

```
# values need quotes
foo: '{{ bar }}' # need quotes to avoid interpreting as `dict` object
foo: '123'       # need quote to avoid interpreting as `int` object
foo: 'yes'           # avoid interpreting as `boolean` object
foo: "bar:baz:bam" # has colon, can be misinterpreted as key

# values need not quotes
foo: bar1baz234
bar: 123baz
```

ref:

- R Markdown anatomy, R Markdown Cookbook
- https://rmarkdown.rstudio.com/lesson-6.html

3.1.1 Render Rmd

When you click the `Knit` button (□□K) in RStudio, generally two processes happen:

1. The `.Rmd` file is fed to `knitr`, which executes all of the R code chunks and creates a new markdown (`.md`) document which includes the R code and its output.
2. The `.md` file is then processed by pandoc which is responsible for creating the finished format, e.g., HTML, PDF, MS_Word.
    - `.md` files can be directly converted to html, but
    - `.md` to pdf is time-consuming. It first generates `.tex`, then call the LaTeX engine to convert to pdf.

There is one function that can do the processes mentioned above: `rmarkdown::render`.

`rmarkdown::render(input, output_format = NULL, output_file = NULL, output_dir = NULL, output_options = NULL, output_yaml = NULL)`

Arguments	Definition
`output_format`	- `"all"` will render all formats define within the file- Name of a format, e.g., `html_document`, will render to that single format- An output format object, e.g., `html_document(toc = TRUE, toc_depth = 2, includes = includes(before_body = "header.htm"))`, where you can pass on the argument
`output_options`	- **List** of output options that can override the options specified in metadata (e.g could be used to force `self_contained` or `mathjax = "local"`). - Note that this is only valid when the output format is read from **metadata** (i.e. not a custom format object passed to output_format).- `output_options` cannot work together with `xxx_document()`.

Arguments	Definition
`output_yaml`	Paths to YAML files specifying output formats and their configurations. The first existing one is used. If none are found, then the function searches YAML files specified to the `output_yaml` top-level parameter in the YAML front matter, `_output.yml` or `_output.yaml`, and then uses the first existing one.

Use example of `render`

```r
render("0208-Rmd-GHpage.Rmd",
    bookdown::pdf_document2(
        latex_engine = "xelatex",
        template = "latex/template.tex",
        includes = includes(
            in_header = "latex/preamble.tex",
            before_body = "latex/before_body.tex")
        ))

# This does NOT work as `output_options` is only valid when the format is not an output format obj
render("0208-Rmd-GHpage.Rmd",
    bookdown::pdf_document2(
        latex_engine = "xelatex",
        template = "latex/template.tex"),
    output_options = list(
        includes = includes(
            in_header = "latex/preamble.tex",
            before_body = "latex/before_body.tex")
        ))
```

You can have more than one output formats for your Rmd. For example, you want both the html and pdf output.

When you render the Rmd with `rmarkdown::render()`, it will use the **first output format you specify in the YAML metadata** (if it is missing, the default is `html_document`).

do not want to use the first one, you can specify the one you want in the second argument, e.g., for an Rmd document `input.Rmd` with the metadata:

```yaml
output:
  html_document:
    toc: true
  pdf_document:
    keep_tex: true
```

You can render it to PDF via:

```r
# Render pdf
rmarkdown::render('input.Rmd', 'pdf_document')

# Render multiple formats
render("input.Rmd", c("html_document", "pdf_document"))

# Render all formats defined
rmarkdown::render('input.Rmd', 'all')
```

- RStudio calls the function `rmarkdown::render()` to render the document in **a new R session**.

  RStudio does this to ensure **reproducibility**.

3.1.2 Document dependency

By default, R Markdown produces standalone HTML files with no external dependencies, using `data:`URIs to incorporate the contents of linked scripts, stylesheets, images, and videos. This means you can share or publish the file just like you share Office documents or PDFs. If you would rather keep dependencies in external files, you can specify `self_contained: false`.

Note that even for self-contained documents, MathJax is still loaded externally (this is necessary because of its big size). If you want to serve MathJax locally, you should specify `mathjax: local` and `self_contained: false`.

One common reason to keep dependencies external is for serving R Markdown documents from a website (external dependencies can be cached separately by browsers, leading to faster page load times). In the case of serving multiple R Markdown documents you may also want to consolidate dependent library files (e.g. Bootstrap, and MathJax, etc.) into a single directory shared by multiple documents. You can use the `lib_dir` option to do this. For example:

```
---
title: "Habits"
output:
  html_document:
    self_contained: false
    lib_dir: libs
---
```

Loading LaTeX packages

We can load additional LaTeX packages using the `extra_dependencies` option **within** the `pdf_document` YAML settings.

This allows us to provide a list of LaTeX packages to be loaded in the intermediate LaTeX output document, e.g.,

```
---
title: "Using more LaTeX packages"
output:
  pdf_document:
    extra_dependencies: ["bbm", "threeparttable"]
---
```

If you need to **specify options** when loading the package, you can add a sub-level to the list and provide the options as a list, e.g.,

```
output:
  pdf_document:
    extra_dependencies:
      caption: ["labelfont={bf}"]
      hyperref: ["unicode=true", "breaklinks=true"]
      lmodern: null
```

For those familiar with LaTeX, this is equivalent to the following LaTeX code:

```
\usepackage[labelfont={bf}]{caption}
\usepackage[unicode=true, breaklinks=true]{hyperref}
\usepackage{lmodern}
```

The advantage of using the `extra_dependencies` argument over the `includes` argument introduced in Section 6.1 is that you do not need to include an external file, so your Rmd document can be **self-contained**.

Includes

HTML Output

You can do more advanced customization of output by including additional HTML content or by replacing the core Pandoc template entirely. To include content in the document header or before/after the document body, you use the `includes` option as follows:

```
---
title: "Habits"
output:
  html_document:
    includes:
      in_header: header.html   # inject CSS and JavaScript code into the <head> tag
      before_body: doc_prefix.html  # include a header that shows a banner or logo.
      after_body: doc_suffix.html  # include a footer
  template: template.html  # custom templates
---
```

An example `header.html` to load a MathJax extension `textmacros`.

```
<script type="text/x-mathjax-config">
  MathJax.Hub.Config({
    loader: {load: ['[tex]/textmacros']},
    tex: {packages: {'[+]': ['textmacros']}}
  });
</script>
```

PDF Output

For example, to support Chinese characters.

You can use `includes` and `preamble.tex` (can be any name, contains any pre-loaded latex code you want to run before your main text code, for setting up environment, loading pkgs, define new commands … Very flexible.)

In the main Rmd:

```
---
output:
  pdf_document:
    includes:
      in_header: latex/preamble.tex
      before_body: latex/before_body.tex
      after_body: latex/after_body.tex
---
```

If you want to add anything to the preamble, you have to use the `includes` option of `pdf_document`. This option has three sub-options:

- `in_header`: loading necessary packages
- `before_body`:
  - Styling that has the highest priority (as it will be loaded latest; if you put in `in_header`, it might be overridden by default settings)
  - Dedication page like "The books is dedicated to …" (此书献给…)

  An example of `before_body.tex`:

```
% Styling that has the highest priority
\let\tightlist\relax % disable `\tightlist`
\setlength{\abovedisplayskip}{-5pt}
\setlength{\abovedisplayshortskip}{-5pt}

% book dedication page
\thispagestyle{empty}

\begin{center}
献给……

呃，爱谁谁吧
\end{center}
```

The default bookdown uses **`\tightlist`** for all bullet lists, setting `itemsep=0pt` and `parskip=0pt`, aim for "compact lists." See the following definition:

```
\providecommand{\tightlist}{%
\setlength{\itemsep}{0pt}\setlength{\parskip}{0pt}}
```

I personally don't like the compact list setting, so I disable it with `\let\tightlist\relax`. To prevent it from being overridden, I put it in `before_body.tex` instead of `preamble.tex`.

- `after_body`.

Each of them takes one or multiple file paths. The file(s) specified in `in_header` will be added to the preamble. The files specified in `before_body` and `after_body` are added before and after the document body, respectively.

```
\documentclass{article}
% preamble
\begin{document}
% before_body
% body
% after_body
\end{document}
```

In `preamble.tex`:

```
\usepackage{xeCJK}
\setCJKmainfont{Noto Sans CJK SC}
```

Alternatively, you can use `header-includes` but with less flexibility to change options:

```
---
output: pdf_document
header-includes:
  - \usepackage{ctex}
---
```

Q: `includes` vs. `header-includes`, which one is better to use for loading LaTeX packages?

A: Another way to add code to the preamble is to pass it directly to the `header-includes` field in the YAML frontmatter. The advantage of using `header-includes`is that you can keep everything in one R Markdown document.

However, if your report is to be generated in *multiple output formats*, we still recommend that you **use the `includes` method**, because the `header-includes` field is unconditional, and will be included in non-LaTeX output documents, too. By comparison, the `includes` option is only applied to the `pdf_document` format.

Ref:

https://github.com/hao203/rmarkdown-YAML?tab=readme-ov-file#chinesejapanese-support https://bookdown.org/yihui/rmarkdown-cookbook/latex-preamble.html

---

header-includes

Tex style and package loading can also put in `header-includes`.

Ex.1

```
---
output: pdf_document
header-includes:
 - \usepackage{fancyhdr}
 - \pagestyle{fancy}
 - \usepackage{ctex} #TeX package for  Chinese
 - \fancyhead[L]{MANUSCRIPT AUTHORS}
 - \fancyhead[R]{MANUSCRIPT SHORT TITLE}
 - \usepackage{lineno} # TeX package for line numbers
 - \linenumbers
---
```

Ex.2

```
---
title: Adding a Logo to LaTeX Title
author: Michael Harper
date: December 7th, 2018
output: pdf_document
header-includes:
  - \usepackage{titling}
  - \pretitle{\begin{center}
    \includegraphics[width=2in,height=2in]{logo.jpg}\LARGE\\}
  - \posttitle{\end{center}}
---
```

Ex.3

To override or extend some CSS for just one document, include for example:

```
---
output: html_document
header-includes: |
  <style>
  blockquote {
    font-style: italic;
  }
  tr.even {
    background-color: #f0f0f0;
  }
  td, th {
    padding: 0.5em 2em 0.5em 0.5em;
  }
  tbody {
    border-bottom: none;
  }
  </style>
---
```

Change Font

The default font is `\usepackage{lmodern}` in bookdown.

Can specify alternative fonts in `preamble.tex` as follows:

```
\usepackage{fontspec}
\setmainfont{Charter}
```

Fonts known to LuaTeX or XeTEX may be loaded by their standard names as you'd speak them out loud, such as Times New Roman or Adobe Garamond. 'Known to' in this case generally means 'exists in a standard fonts location' such as `~/Library/Fonts` on macOS, or `C:\Windows\Fonts` on Windows. In LuaTEX, fonts found in the TEXMF tree can also be loaded by name. In XeTEX, fonts found in the TEXMF tree can be loaded in Windows and Linux, but not on macOS.

3.2 Chunk Options

If you want to set chunk options globally, call `knitr::opts_chunk$set()` in a code chunk (usually the first one in the document), e.g.,

~~~
```{r, label="setup", include=FALSE}
knitr::opts_chunk$set(
  comment = "#>", echo = FALSE, fig.width = 6
)
```
~~~

Full list of chunk options: https://yihui.org/knitr/options/

Chunk options can customize nearly all components of code chunks, such as the source code, text output, plots, and the language of the chunk.

Other languages are supported in `Rmd`

You can list the names of all available engines via:

```
names(knitr::knit_engines$get())
##  [1] "awk"          "bash"         "coffee"
##  [4] "gawk"         "groovy"       "haskell"
##  [7] "lein"         "mysql"        "node"
## [10] "octave"       "perl"         "php"
## [13] "psql"         "Rscript"      "ruby"
## [16] "sas"          "scala"        "sed"
## [19] "sh"           "stata"        "zsh"
## [22] "asis"         "asy"          "block"
## [25] "block2"       "bslib"        "c"
## [28] "cat"          "cc"           "comment"
## [31] "css"          "ditaa"        "dot"
## [34] "embed"        "eviews"       "exec"
## [37] "fortran"      "fortran95"    "go"
## [40] "highlight"    "js"           "julia"
## [43] "python"       "R"            "Rcpp"
## [46] "sass"         "scss"         "sql"
## [49] "stan"         "targets"      "tikz"
## [52] "verbatim"     "theorem"      "lemma"
## [55] "corollary"    "proposition"  "conjecture"
## [58] "definition"   "example"      "exercise"
## [61] "hypothesis"   "proof"        "remark"
## [64] "solution"     "marginfigure"
```

The engines from `theorem` to `solution` are only available when you use the **bookdown** package, and the rest are shipped with the **knitr** package.

To use a different language engine, you can change the language name in the chunk header from `r` to the engine name, e.g.,

~~~
```python
x = 'hello, python world!'
print(x.split(' '))
```
~~~

For engines that rely on external interpreters such as `python`, `perl`, and `ruby`, the default interpreters are obtained from `Sys.which()`, i.e., using the interpreter found via the environment variable `PATH` of the system. If you want to use an alternative interpreter, you may specify its path in the chunk option `engine.path`.

For example, you may want to use Python 3 instead of the default Python 2, and we assume Python 3 is at `/usr/bin/python3`

~~~
```{python, engine.path = '/usr/bin/python3'}
import sys
print(sys.version)
```
~~~

- All outputs support markdown syntax.
- If the output is html, you can write in html syntax.

The **chunk label** for each chunk is assumed to be unique within the document. This is especially important for cache and plot filenames, because these filenames are based on chunk labels. Chunks without labels will be assigned labels like `unnamed-chunk-i`, where `i` is an incremental number.

- Chunk label doesn't need a `tag`, i.e., you only provide the `value`.
- If you prefer the form `tag=value`, you could also use the chunk option `label` explicitly, e.g.,

  ~~~
  ```{r, label='my-chunk'}
  # one code chunk example
  ```
  ~~~

You may use `knitr::opts_chunk$set()` to change the default values of chunk options in a document.

Commonly used chunk options

- Complete list here. Or `?opts_chunk` to get the help page.

Options	Definitions
`echo=TRUE`	Whether to display the **source code** in the output document.Use this when you want to show the output but not the code itself.
`eval=TRUE`	Whether to evaluate the code chunk.
`include=TRUE`	Whether to include the chunk **output** in the output document. If `FALSE`, nothing will be written into the output document, but the code is still evaluated and plot files are generated if there are any plots in the chunk, so you can manually insert figures later.
`message=TRUE`	Whether to preserve messages emitted by `message()`
`warning=TRUE`	Whether to show warnings in the output produced by `warning()`.
`results='markup'`	Controls how to display the text results. When `results='markup'` that is to write text output as-is, i.e., write the raw text results directly into the output document without any markups.Useful when priting `stargazer` tables.

Options	Definitions
`comment='##'`	The prefix to be added before each line of the text output. Set `comment = ''` remove the default ##.
`collapse=FALSE`	Whether to, if possible, collapse all the source and output blocks from one code chunk into a single block (by default, they are written to separate blocks). This option only applies to Markdown documents.
`fig.keep='high'`	How plots in chunks should be kept. `high`: Only keep high-level plots (merge low-level changes into high-level plots). `none`: Discard all plots. `all`: Keep all plots (low-level plot changes may produce new plots). `first`: Only keep the first plot. `last`: Only keep the last plot. If set to a numeric vector, the values are indices of (low-level) plots to keep.If you want to choose the second to the fourth plots, you could use `fig.keep = 2:4` (or remove the first plot via `fig.keep = -1`).
`fig.align="center"`	Figure alignment.
`fig.pos="H"`	A character string for the figure position arrangement to be used in `\begin{figure}[]`.
`fig.cap`	Figure caption.

`results='markup'` note plural form for result**s**.

- `markup`: Default. Mark up text output with the appropriate environments depending on the output format. For example, for R Markdown, if the text output is a character string `"[1] 1 2 3"`, the actual output that **knitr** produces will be:

  ~~~
  ```
  [1] 1 2 3
  ```
  ~~~

  In this case, `results='markup'` means to put the text output in fenced code blocks (“‘).

- `asis`: Write text output as-is, i.e., write the raw text results directly into the output document without any markups.

  ~~~
  ```{r, results='asis'}
  cat("I'm raw **Markdown** content.\n")
  ```
  ~~~

  Sometime, you encounter the following error messages when you have R codes within `enumerate` environment.

  You can’t use `macro parameter character #` in horizontal mode.

  By default, knitr prefixes R output with ##, which can’t be present in your TeX file.

  Solution:

  – specify `results="asis"` in code chunks.

- `hold`: Hold all pieces of text output in a chunk and flush them to the end of the chunk.

- `hide` (or `FALSE`): Hide text output.

`collapse=FALSE` Whether to merge text output and source code into a single code block in the output. The default `FALSE` means R expressions and their text output are separated into different blocks.

`collapse = TRUE` makes the output more compact, since the R source code and its text output are displayed in a single output block. The default `collapse = FALSE` means R expressions and their text output are separated into different blocks.

3.3 Print Verbatim R code chunks

verbatim in line code

- use `knitr::inline_expr`.

````
---
title: "Test inline expr"
output: html_document
---

To use `chunk_reveal("walrus", title = "## Walrus operator")` inline, you can wrap it in R inline
````

Including verbatim R code chunks inside R Markdown

One solution for including verbatim R code chunks (see below for more) is to insert hidden inline R code (`` `r     ''` ``) immediately before or after your R code chunk.

- The hidden inline R code will be evaluated as an inline expression to an empty string by knitr.

Then wrap the whole block within a markdown code block. The rendered output will display the verbatim R code chunk — including backticks.

R code generating the four backticks block:

~~~
output_code <-
"````markdown
```{r}
plot(cars)
```  \n````"
cat(output_code)
~~~

Write this code in your R Markdown document:

~~~
````markdown
`r ''````{r}
plot(cars)
```
````
~~~

or

~~~
````markdown
```{r}`r ''`
plot(cars)
```
````
~~~

Knit the document and the code will render like this in your output:

~~~
```{r}
plot(cars)
```
~~~

This method makes use of **Markdown Syntax** for code.

Q: What is the Markdown Syntax for code?
A:

- Inline code use a pair of backticks, e.g., `` `code` ``. To use n literal backticks, use at least $n+1$ backticks outside. Note that use a space to separate your outside backticks from your literal backtick(s). For example, to generate `` `code` ``, you use ``` ``□`code`□`` ``` (i.e., two backticks + space + one backtick + `code` + one backtick + space + two backticks). Note that you need to write sequentially.

- Plain code blocks can be written either
  - After three or more backticks (fenced code blocks), or

    Can also use tildes (~)
  - Indent the blocks by four spaces (indented code blocks)

    Special characters do not trigger special formatting, and all spaces and line breaks are preserved. Blank lines in the verbatim text need not begin with four spaces.
- Note that code blocks must be separated from surrounding text by blank lines.

If the code itself contains a row of tildes or backticks, just use a longer row of tildes or backticks at the start and end:

````
~~~~~~~~~~~~~~~~
~~~~~~~~~~
code including tildes
~~~~~~~~~~
~~~~~~~~~~~~~~~~
````

These begin with a row of three or more tildes (~) and end with a row of tildes that must be at least as long as the starting row.

A trick if you don't want to type more than three tildes or backticks is that you just use different inner and outer symbols.

````
~~~markdown
```r
print ("hello world")
```
~~~
````

Will be rendered as:

~~~
```r
print ("hello world")
```
~~~

A shortcut form (without braces) can also be used for specifying the language of the code block:

~~~
```haskell
qsort [] = []
```
~~~

This is equivalent to:

~~~
``` {.haskell}
qsort [] = []
```
~~~

`haskell` is the language class.

You can add more classes, such as `numberLines` for adding line numbers.

This shortcut form may be combined with attributes:

~~~
```haskell {.numberLines}
qsort [] = []
```
~~~

Which is equivalent to:

~~~
``` {.haskell .numberLines}
qsort [] = []
```
~~~

and

```
<pre id="mycode" class="haskell numberLines" startFrom="100">
  <code>
  primes = filterPrime [2..] where
  filterPrime (p:xs) =
    p : filterPrime [x | x <- xs, x `mod` p /= 0]
  </code>
</pre>
```

If highlighting is supported for your output format and language, then the code block above will appear highlighted, with numbered lines starting with 100, 101, and go on.

Code chunks within `enumerate`

- Mind the indentation. Rstudio does not automatically adjust indentation for codes.
- specify `results="asis"` if encounter

  You can't use 'macro parameter character #' in horizontal mode.
- cross references using bookdown (`\@ref{fig:scatter-plot}`) might not work.

  Use latex references `\ref{fig:scatter-plot}` (base latex) or `\autoref{fig:scatter-plot}` (from `hyperref` package)
- markdown language does not work well inside latex environments. A possible workaround is use `1` and indent four spaces for contents that follow.

If it is still a pain in the ass, use this solution.

Basically, just copy the output from R condole and paste in Rmd.

References:

https://yihui.org/en/2017/11/knitr-verbatim-code-chunk/

https://support.posit.co/hc/en-us/articles/360018181633-Including-verbatim-R-code-chunks-inside-R-Markdown

https://themockup.blog/posts/2021-08-27-displaying-verbatim-code-chunks-in-xaringan-presentations/

Pandoc's Markdown: https://pandoc.org/MANUAL.html#fenced-code-blocks

3.4 Rmd Basics

To name a chunk, add the name after `r`, it's not necessary to add `label='chunk-name'`, but it is possible to do so if you prefer the form `tag=value`.

The chunk label

- Must be unique within the document. This is especially important for cache and plot filenames, because these filenames are based on chunk labels. Chunks without labels will be assigned labels like `unnamed-chunk-i`, where `i` is an incremental number.
- Avoid spaces (⍽), periods (`.`), and underscores (_) in chunk labels and paths. If you need separators, you are recommended to use hyphens (`-`) instead.

`knitr::opts_chunk$set()` changes the default values of chunk options in a document.

Unnumbered sections

Add {-} at the end of the section title.

```
# Question 1: Variance and Covariance properties {-}
<!-- equivalently, you can use {.unnumbered} -->
# Question 1: Variance and Covariance properties {.unnumbered}
```

Note that the section won't be numbered but will show in the TOC.

If you want to further exclude it from the TOC:

```
# Question 1: Variance and Covariance properties {.unlisted .unnumbered}
```

Headings with # will appear in the file outline, which is a convenient feature. So use this method whenever possible.

One exception is level 2 headings in Bookdown:

- By default Bookdown starts a new page for each level 2 heading. If you want to keep the style wihtout starting a new page, use an html tag. The heading won't be numbered or included in TOC. However, a downside is that the heading won't show up in the file outline either, making them harder to locate.

  ```
  <h2>YAML metadata</h2>
  ```

Knitting in the global environment

```
rmarkdown::render("/Users/menghan/Library/CloudStorage/OneDrive-Norduniversitet/EK369E/Semi
```

Advantages: fast; load and output results in the global environment; easy to inspect afterwards.

Rmd built-in themes for `html` output: https://rstudio4edu.github.io/rstudio4edu-book/rmd-themes.html

`.Rmd` documents can be edited in either source or visual mode. To switch into visual mode for a given document, use the Source or Visual button at the top-left of the document toolbar (or alternatively the `Cmd+Shift+F4` keyboard shortcut).

- Visual mode allows you to preview the effect after having compiled the markdown file.

  But it modifies your code siliently, be cautions with visual mode.
- More user-friendly in terms of providing dropdown menus for editting.
- Visual mode supports both traditional **keyboard shortcuts** (e.g. `Cmd + B` for bold) as well as markdown shortcuts (using markdown syntax directly). For example, enclose `**bold**` text in asterisks or type `##` and press space to create a second level heading.
- One bug for Visual mode is that inside **bullet points**, `$` is automatically escaped as `\$`. In this case, use `cmd+/` and choose inline math to insert an eqn.
- When type inline equations, first type `$` then the equation, then `$` at last. Do not type `$$` at one time. Otherwise, they will be escaped as regular text.

Comments in Rmd

- In both html and pdf outputs, use the following to write true comments you don't want to show in the rendered file.

  ```
  <!-- regular html comment -->
  ```

Link to an external javascript

```
<SCRIPT language="JavaScript" SRC="my_jxscript.js"></SCRIPT>
```

Tips:

- In general, you'd better leave at least one empty line between adjacent but different elements, e.g., a header and a paragraph. This is to avoid ambiguity to the Markdown renderer.

  For example, the `-` in the list below cannot be recognized as a bullet point. You need to add a black line before the bullet list.

  ```
  The result of 5
  - 3 is 2.
  ```

  Different flavors of Markdown may produce different results if there are no blank lines. □□

3.5 Citations

For an overview of including bibliographies in your output document, you may see Section 2.8 of Xie (2016). The basic usage requires us to specify a bibliography file using the `bibliography` metadata field in YAML. For example:

```
---
output: html_document
bibliography: references.bib
---
```

where the BibTeX database is a plain-text file with the `*.bib` extension that consists of bibliography entries.

How to cite in text:

- Use `@citationkey` to cite references in text.
- To put citations in parentheses, use `[@citationkey]`.
- To cite multiple entries, separate the keys by semicolons, e.g., `[@key-1; @key-2; @key-3]`.
- To suppress the mention of the author, add a minus sign before `@`, e.g., `[-@citationkey]`.

Syntax	Result
`@adams1975` concludes that …	Adams (1975) concludes that …
`@adams1975[p.33]` concludes that …	Adams (1975, p. 33) concludes that …
… end of sentence `[@adams1975]`.	… end of sentence (Adams, 1975).
`[see @adams1975,p.33]`.	… end of sentence (see Adams, 1975, p. 33).
delineate multiple authors with colon: `[@adams1975; @aberdeen1958]`	delineate multiple authors with colon: (Aberdeen, 1958; Adams, 1975)
Check Lo and MacKinlay `[-@Lo-Mackinlay1988; -@Lo1989]` for example.	Check Lo and MacKinlay (1988, 1989) for example.

Add an item to bibliography without using it

By default, the bibliography will only display items that are directly referenced in the document.

If you want to include items in the bibliography without actually citing them in the body text, you can define a dummy `nocite` metadata field and put the citations there.

```
---
nocite: |
  @item1, @item2
---
```

3.5.1 Bibliographies

Users may also choose to use either `natbib` (based on bibtex) or `biblatex` as a "citation package". In this case, the bibliographic data files need to be in the bibtex or biblatex format, and the document output format is **limited to PDF**.

```
output:
  pdf_document:
    citation_package: natbib
  bookdown::pdf_book:
    citation_package: biblatex
```

If you use matching styles (e.g., `biblio-style: apa` for `biblatex` along with `csl: apa.csl` for `pandoc-citeproc`), output to PDF and to non-PDF formats will be very similar, though not necessarily identical.

Once you have one or multiple `.bib` files, you may use the field `bibliography` in the YAML metadata of your first R Markdown document (which is typically `index.Rmd`), and you can also specify the bibliography style via `biblio-style` (this only applies to PDF output), e.g.,

```
---
bibliography: ["one.bib", "another.bib", "yet-another.bib"]
biblio-style: "apalike"
link-citations: true
---
```

The field `link-citations` can be used to add internal links from the citation text of the author-year style to the bibliography entry in the HTML output.

For any non-PDF output format, `pandoc-citeproc` is the only available option. If consistency across PDF and non-PDF output formats is important, use `pandoc-citeproc` throughout.

To change the bibliography style, you will need to specify a CSL (Citation Style Language) file in the csl metadata field, e.g.,

```
---
output: html_document
bibliography: references.bib
csl: biomed-central.csl
---
```

3.6 Cross References

3.6.1 Using bookdown

You can number and refer to an equation by adding `\begin{equation}` along with a label, provided with `(\#eq:label)`.

- The position of the label matters.
  - For single-lined equations: First write your equation, then append your label `(\#eq:label)`. Otherwise, your equation won't be rendered.
  - For multi-lined equations: append `(\#eq:label)` after `\end{split}`, `\end{aligned}` …

- Note that `\begin{equation}` must NOT be quoted in `$$...$$` for the equation to be rendered.

  Otherwise, will cause "Bad math delimiter" error at the time of tex compilation for pdf output. Might be alright for html output though.

  Unexpected consequence: Without the `$$...$$`, RStudio won't provide previews for equations.

  - For temporary preview in RStudio at the composing stage, you can enclose the whole math environment in `$$...$$`. But **remember to delete them when you are done** editing the equation.
  - See this post by Kenji Sato for a more efficient workaround.

- You can then refer to the equation in text using `\@ref(eq:CJ)`. Remember to put the label in parentheses `()`.

  General syntax for other environments: `\@ref(type:label)` where `type` is the environment being referenced, and `label` is the chunk label.

```
This is an equation redered using bookdown

\begin{equation} (\#eq:CJ)
y=\beta_0 + \beta_1x + e_t
\end{equation}
```

will render as

$$y = \beta_0 + \beta_1 x + e_t \tag{3.1}$$

You may refer to it using eqn `\@ref(eq:CJ)`, e.g., see eqn (**??**).

```
Multilined equations.

\begin{equation}
\begin{aligned}
y_i &= f(x_{1i}, x_{2i}, \ldots, x_{Ki}) + \varepsilon_i \\
&= x_{1i} \beta_1 + x_{2i} \beta_2 + \cdots + x_{Ki} \beta_K + \varepsilon_i
\end{aligned}(\#eq:scalar-form)
\end{equation}
```

will render as

$$\begin{aligned}
y_i &= f(x_{1i}, x_{2i}, \ldots, x_{Ki}) + \varepsilon_i \\
&= x_{1i} \beta_1 + x_{2i} \beta_2 + \cdots + x_{Ki} \beta_K + \varepsilon_i
\end{aligned} \tag{3.2}$$

You may refer to it using eqn `\@ref(eq:scalar-form)`, e.g., see eqn (**??**) .

Note that

- For HTML output, **bookdown** can only number the equations with labels.

  Please make sure equations without labels are not numbered by either using the `equation*` environment or adding `\nonumber` or `\notag` to your equations.

Troubleshooting

Issue: Bad math environment delimiter on conversion to pdf when using equation or align.

Cause: The error happens because I enclosed `\begin{equation}` environment in `$$`. I did this as the dollar sings enable equation rendering and preview in file.

Fix: remove the double signs.

```
The following equation causes error. Need to remove the dollar signs.
$$
\begin{equation}
y=x+2
\end{equation}
$$
```

More examples:

- **Headers**

  ```
  # Introduction {#intro}

  This is Chapter \@ref(intro)
  ```

- **Figures**

  ~~~
  See Figure \@ref(fig:cars-plot)

  ```{r cars-plot, fig.cap="A plot caption"}
  plot(cars)  # a scatterplot
  ```
  ~~~

- **Tables**

  ~~~
  See Table \@ref(tab:mtcars)

  ```{r mtcars}
  knitr::kable(mtcars[1:5, 1:5], caption = "A caption")
  ```
  ~~~

- **Theorems**

  ~~~
  See Theorem \@ref(thm:boring)

  ```{theorem, boring}
  Here is my theorem.
  ```
  ~~~

- **Equations**

  ```
  See equation \@ref(eq:linear)

  \begin{equation}
  a + bx = c  (\#eq:linear)
  \end{equation}
  ```

3.6.2 Using the LaTeX Way

The **LaTeX** way allows you to assign your own labels by `\tag`. One drawback is that this does not allow preview of equations.

1. Add the following script at the beginning of your document body:

   ```
   <script type="text/x-mathjax-config">
   MathJax.Hub.Config({
     TeX: { equationNumbers: { autoNumber: "AMS" } }
   ```

```
});
</script>
```

It configures MathJax to automatically number equations. Source.

2. In the text, use `label{eq:label}`. If you want to provide a specific number to the equation, you can use `\tag{XX.XX}`.
   - Note that `\begin{equation}` is NOT inside `$$ ...$$`!
3. Cite using `$\ref{eq:label}$` (no parenthesis) or `$\eqref{eq:label}$` (with parenthesis). The dollar sign `$` here around `\ref` and `\eqref` is not essential. Commands work with or without `$`.

```
Without using the bookdown package.

\begin{equation} \label{eq:test} \tag{my custom label}
  Y_i = \beta_0 + \beta_1 x_i + \epsilon_i
\end{equation}

Cite Equation $\eqref{eq:test}$ like this.
```

$$Y_i = \beta_0 + \beta_1 x_i + \epsilon_i \tag{my label}$$

Refer to the eq (**??**)

Reference:

https://bookdown.org/yihui/bookdown/markdown-extensions-by-bookdown.html#equations

3.7 Equations

Can use `$...$` (`$$...$$` for blocks) or `\(...\)` (`\[...\]` for blocks) to enclose equations. Difference:

- `$...$` provides rendered equation previews in RStudio.
- `\(...\)` does not have previews.

Rstudio equation previews do NOT work well with indented equations. $\rightarrow$ reduce indentation

如果公式缩进，Rstudio 公式预览功能可能不识别。在不影响理解的前提下，减少不必要的缩进以便预览公式。

Multi-case functions using `\begin{cases}`

```
\begin{align*}
I_t =
\begin{cases}
1 & \text{if } r_t>0 \\
0 & \text{if } r_t\leq0
\end{cases}
\end{align*}
```

will render as

$$I_t = \begin{cases} 1 & \text{if } r_t > 0 \\ 0 & \text{if } r_t \leq 0 \end{cases}$$

For equation numbering support in `bookdown` you need to assign labels.

You may refer to an equation using Eq. `\@ref(eq:eq01)`.

```
\begin{align} (\#eq:eq01)
\frac{p(x)}{1-p(x)} = \exp (\beta_0+\beta_1 x) \,.
\end{align}
```

If you want to provide a specific number to the equation, you can use `\tag{XX.XX}`.

- With LaTeX
  LaTex allows custom labels.

$$\frac{p(x)}{1-p(x)} = \exp(\beta_0 + \beta_1 x)\,. \tag{my label latex}$$

  My specific label here, see eq (**??**) (`\eqref{eq:my-label-latex}`).

- With `bookdown`
  `bookdown` does NOT support custom tag though.

$$\frac{p(x)}{1-p(x)} = \exp(\beta_0 + \beta_1 x)\,. \tag{3.3}$$

  My specific label here, see eq (**??**)

Color eqns using `\color{#00CC66}{...}`.

But sometime everything follows gets colored. You may want to use `{\color{#00CC66} ... }` instead.

```
$$
\color{#008B45}{Y_t} = I_tI_{t-1} + (1-I_t)(1-I_{t-1})
$$
```

$$\color{red}{Y_t} = I_tI_{t-1} + (1-I_t)(1-I_{t-1})$$

- This only works for color names, not hex codes starting with #, because html requires the # followed by 6 characters to define a color, but LaTeX package `xcolor` specifically excludes # in color specifications.

- Here is an inline colored example for LaTeX output (only works for LaTeX).

A workaround: We can write a custom R function to insert the correct syntax depending on the output format using the `is_latex_output()` and `is_html_output()` functions in knitr as follows:

```
colorize <- function(x, color) {
  if (knitr::is_latex_output()) {
    sprintf("\\textcolor{%s}{%s}", color, x)
  } else if (knitr::is_html_output()) {
    sprintf("<span style='color: %s;'>%s</span>", color,
      x)
  } else x
}
```

We can then use the code in an inline R expression `` `r colorize("some words in red", "red")` ``, which will create some words in red, which works for both html and .

Mathjax

https://bookdown.org/yihui/rmarkdown/html-document.html#mathjax-equations

Default configuration used by the rmarkdown package is given by `rmarkdown:::mathjax_config()`. As of rmarkdown v2.1, the function returns "MathJax.js?config=TeX-AMS-MML_HTMLorMML". This configures `Mathjax` to `HTML-CSS`.

Change `Mathjax` configuration to `CommonHTML` using the following codes.

```
---
title: "Trouble with MathJax"
output:
  html_document:
    mathjax: "https://cdnjs.cloudflare.com/ajax/libs/mathjax/2.7.5/MathJax.js?config=TeX-AMS_CHTML
    self_contained: false
---
```

By default, MathJax scripts are included in HTML documents for rendering LaTeX and MathML equations. You can use the `mathjax` option to control how MathJax is included:

- Specify `"default"` to use an HTTPS URL from a CDN host (currently provided by RStudio).
- Specify `"local"` to use a local version of MathJax (which is copied into the output directory). Note that when using `"local"` you also need to set the `self_contained` option to `false`.
- Specify an alternate URL to load MathJax from another location. To use a self-hosted copy of MathJax.
- Specify `null` to exclude MathJax entirely.

Q: Why my eqns are not rendered?

A: MathJax is unlikely to work offline. Check internet connection.

You load MathJax into a web page by including its main JavaScript file into the page. That is done via a `<script>`tag that links to the `MathJax.js` file. To do that, place the following line in the `<head>` section of your document.

For example, if you are using the MathJax distributed network service, the tag might be

```
<script type="text/javascript"
   src="http://cdn.mathjax.org/mathjax/latest/MathJax.js">
</script>
```

MathJax is available as a web service from `cdn.mathjax.org`, so you can obtain MathJax from there without needing to install it on your own server. The CDN is part of a distributed "cloud" network, so it is handled by servers around the world. That means that you should get access to a server geographically near you, for a fast, reliable connection.

The CDN hosts the most current version of MathJax, as well as older versions, so you can either link to a version that stays up-to-date as MathJax is improved, or you can stay with one of the release versions so that your pages always use the same version of MathJax.

For equation numbering support in `bookdown::pdf_document2` you need to assign labels. Defualt behavior is not adding numbering.

- Use `\begin{equation}...\end{equations}` or `\begin{align}...\end{align}` environments.
  - Use `(\#eq:eq1)` or `\label{eq:eq1}` to add labels.
  - Automatically add numbering.
  - Drawback is that rmd does not have preview of equations.
- Do NOT enclose the environments in double dollar signs `$$`. Otherwise, no label is added, but cross-references still show up.
  - `$$` do not add numbering automatically.
  - But in `bookdown::html_document2`, it is ok to use

    ```
    $$
    \begin{equation} (\#eq:simple-lm)
    \hat{\beta}_{\text{OLS}} = \left(\sum_{i=1}^n x_i x_i' \right)^{-1} \left(\sum_{i=
    \end{equation}
    $$
    ```

    Then reference with `\@ref(eq:simple-lm)`.
- Use `\@ref(eq:eq1)` (note this use parentheses) or the Latex command `\eqref{eq:eq1}` (this uses curly braces) to cite the equation.

```
Load the dataset and calculate the monthly return in month $r$ ($r_t$) as

\begin{equation}
r_t = \frac{P_t-P_{t-1}}{P_{t-1}} = \frac{P_t}{P_{t-1}}-1 ,
(\#eq:eq1)
\end{equation}

where $P_t$ is the adjusted price in month $t$.

Test equation1 \@ref(eq:eq1).

Test equation2 \eqref{eq:eq1}.
```

3.8 Theorems

https://stackoverflow.com/questions/50379923/bookdown-remark-environment

Language internationalization: https://bookdown.org/yihui/bookdown/internationalization.html

Theorem environments in the `bookdown` package.

Environment	Printed Name	Label Prefix
`theorem`	Theorem	thm
`lemma`	Lemma	lem
`corollary`	Corollary	cor
`proposition`	Proposition	prp
`conjecture`	Conjecture	cnj
`definition`	Definition	def
`example`	Example	exm
`exercise`	Exercise	exr
`hypothesis`	Hypothesis	hyp

- Definition : an explanation of the mathematical meaning of a word.
- Theorem : A statement that has been proven to be true.

- Proposition : A less important but nonetheless interesting true statement.
- Lemma: A true statement used in proving other true statements (that is, a less important theorem that is helpful in the proof of other results).
  - Lemmas are considered to be less important than propositions. But the distinction between categories is rather *blurred*.
  - There is no formal distinction among a lemma, a proposition, and a theorem.
- Corollary: A true statment that is a simple deduction from a theorem or proposition.
- Proof: The explanation of why a statement is true.
- Conjecture: A statement believed to be true, but for which we have no proof. (a statement that is being proposed to be a true statement).
- Axiom: A basic assumption about a mathematical situation. (a statement we assume to be true).

Usage

Theorems and proofs provide environments that are commonly used within articles and books in mathematics. To write a theorem, you can use the syntax below:

~~~
```{theorem, label, name="Theorem name"}
Here is my first theorem.
```
~~~

will be rendered as:

Theorem 3.1 (Theorem name). *Here is my first theorem.*

Refer to the theorem using `\@ref(thm:label)`, e.g., see theorem **??**.

Another example

~~~
```{theorem, thm-py, name="Pythagorean theorem"}
For a right triangle, if $c$ denotes the length of the hypotenuse and $a$ and $b$ denote the lengt

  \begin{align*}
  c^2 = a^2+b^2
  \end{align*}
```
~~~

will be rendered as:

Theorem 3.2 (Pythagorean theorem). *For a right triangle, if c denotes the length of the hypotenuse and a and b denote the lengths of the other two sides, we have*

$$c^2 = a^2 + b^2$$

- Variants of the `theorem` environments include: `lemma`, `corollary`, `proposition`, `conjecture`, `definition`, `example`, and `exercise`. The syntax for these environments is similar to the `theorem` environment, e.g., “‘`{lemma}`.
- The `proof` environment behaves similarly to theorem environments but is unnumbered. Variants of the `proof` environments include `remark` and `solution`.

  The `proof`environment behaves similarly to theorem environments but is unnumbered.

Customize math environment labels

You need to create a file `_bookdown.yml` in the same directory as your `.Rmd`.

In the configuration file **`_bookdown.yml`**

For example, if you want `FIGURE x.x` instead of `Figure x.x`, you can change `fig` to `"FIGURE "`:

```
language:
  label:
    fig: "FIGURE "
```

If you want to number `proof`,

1. choose one of the predefined theorem like environments that you are not using otherwise, e.g. `example` or `exercise`.
2. Redefine the printed name for that environment in `_bookdown.yml` (c.f. https://bookdown.org/yihui/bookdown/internationalization.html) via:

   ```
   language:
     label:
       exr: 'Proof '
   ```

   Here I changed the `exercise` environment leading word to “Proof”.
3. In your `Rmd` files use `{exercise, mylabel}` environment.

   ~~~
   ```{exercise, mylabel}
   my comment
   ```

   In Remark \@ref(exr:mylabel) we discussed...
   ~~~

   Note that you have to use `exercise` and the corresponding label prefix `exr`.

Can specify environment style in `style.css`

```
.exercise {
    margin: 10px 5px 20px 5px;
}
/* define a boxed environment */
.boxed {
    border: 1px solid #535353;
    padding-bottom: 20px;
}
```

~~~
<div class = "boxed">
```{exercise, proof2}
Show $\pi=\Phi \left(\frac{\mu}{\sigma}\right)$.
$$
\begin{aligned}[b]
P(r_t>0) &= P(\mu+e_t>0) \\
&= P(e_t>-\mu) \quad\quad\quad (\sigma>0, \text{dividing by a pos. number, inequality uncha
&= P\left( \frac{e_t}{\sigma} > -\frac{\mu}{\sigma}\right) \quad\;\; e_t\sim N(0, \sigma^2)
&= P \left( \frac{e_t}{\sigma} < \frac{\mu}{\sigma} \right) \\
&= \Phi \left(\frac{\mu}{\sigma} \right)
\end{aligned}  \square
$$
```
</div>
~~~

3.9 Figures

The idea is to generate the figure, output to local, then reload using the following code.

~~~
```{r car-plot, eval=TRUE, fig.asp = 0.62, echo=FALSE, out.width="80%", fig.cap="Caption here." }
knitr::include_graphics(img1_path)
~~~
```

Use code chunk label to cross reference, e.g., `Fig. \@ref(fig:car-plot)`.

- Note that you must specify `fig.cap` to enable labeling and cross references. Otherwise, the cross reference will show `Fig. ??`.

---

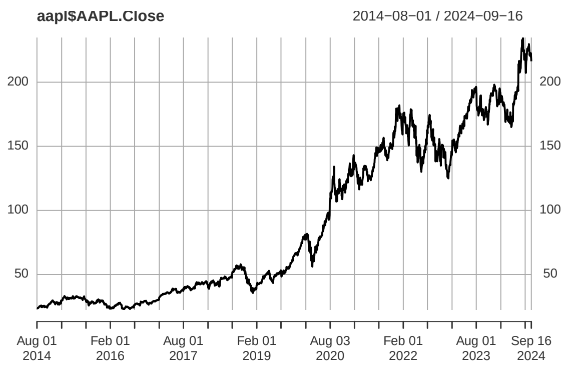
- You can let the code output to document directly, i.e., not generating a file and reload.

  But in this case, scale the figure will change the plot text too. The text might be scaled unexpectedly too small/large. Just be careful with it.

**Output directly to document**

```
library(quantmod)
aapl <- getSymbols("AAPL",
          src = 'yahoo',
          from = "2014-08-01",
          to = "2024-09-17",
          auto.assign = FALSE
          )
```

```
```{r out.width="50%", fig.asp = 0.62, fig.cap="`out.width=\"50%\"`, fig.asp set to 0.62."}
# plot text is scaled too
plot(aapl$AAPL.Close)
```
```

\begin{figure} 
aapl$AAPL.Close
2014−08−01 / 2024−09−16
200
150
100
50
Aug 01 2014
Feb 01 2016
Aug 01 2017
Feb 01 2019
Aug 03 2020
Feb 01 2022
Aug 01 2023
Sep 16 2024
\caption{out.width="50%", fig.asp set to 0.62. Note that text font scales too, hard to read.} \end{figure}

---

```
```{r fig.width=6, fig.asp=0.6}
# Text font does NOT scale, but figure title got cropped
plot(aapl$AAPL.Close)
```
```

---

```
```{r out.width="100%", fig.asp = 0.6, fig.cap="`out.width=\"100%\"`."}
plot(aapl$AAPL.Close)
```
```
```

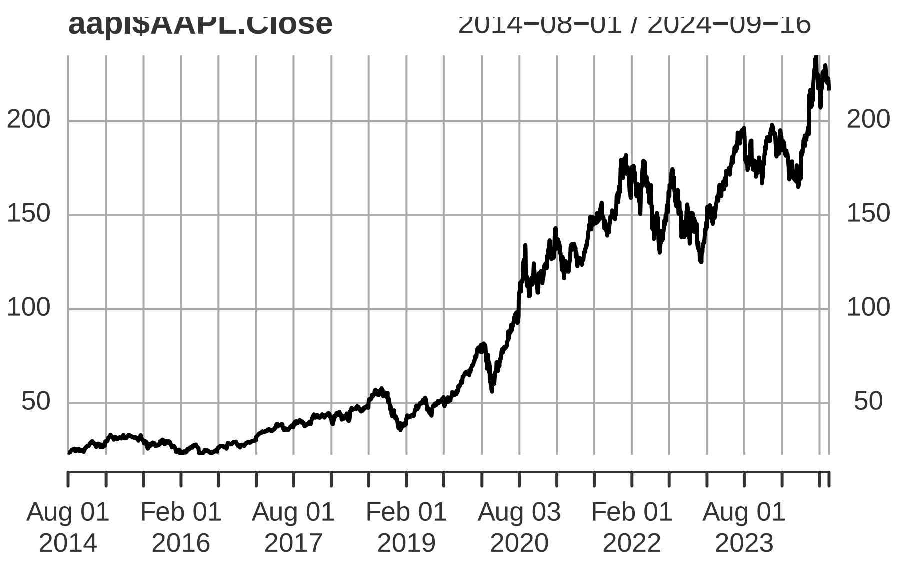
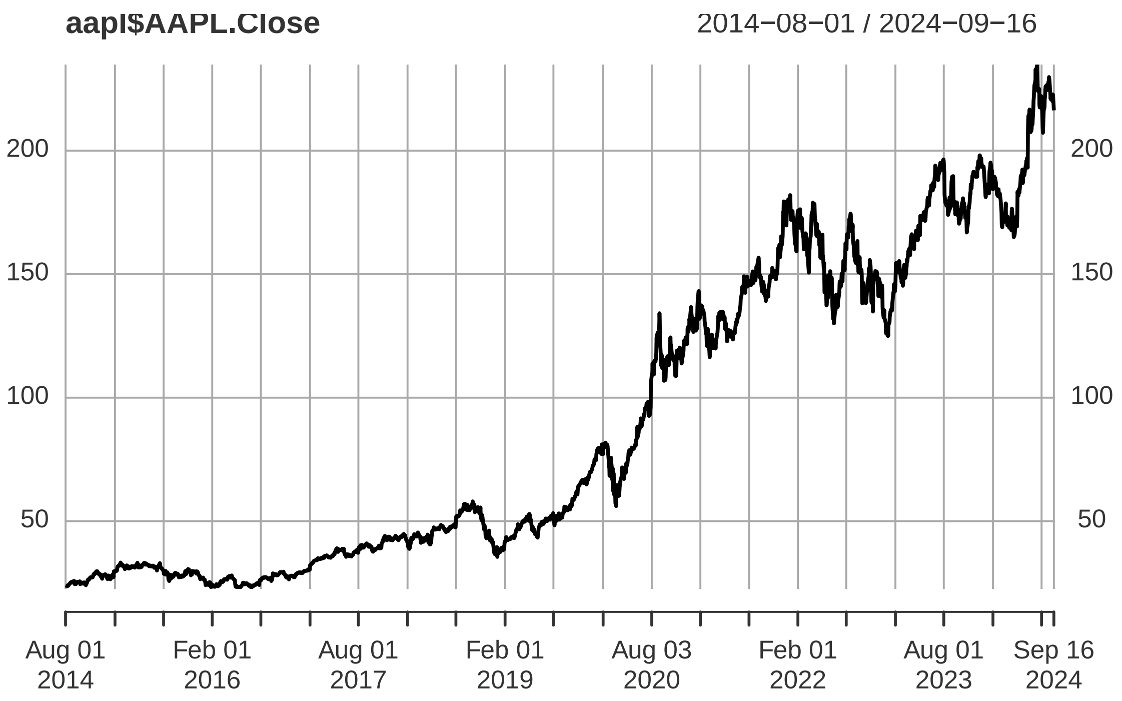


Figure 3.1: Set `fig.width`. Note that text font does NOT scale with figure, BUT the figure title got cropped.



\begin{figure}
\caption{`out.width="100%"`, fig.asp set to 0.6. Note that the plot text got zoomed too, can be too large.} \end{figure}

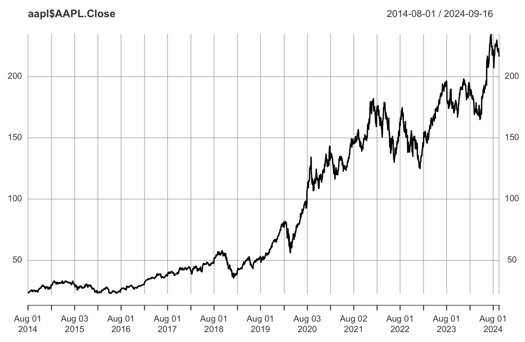
Save and reload

This approach preserves your preference better, maintains the relative size of your figure and the text.

No cropping, no fuss.

```r
f_name <- "images/aapl.png"
png(f_name, width=2594, height=1600, res=300)
plot(aapl$AAPL.Close)
invisible(dev.off())
```

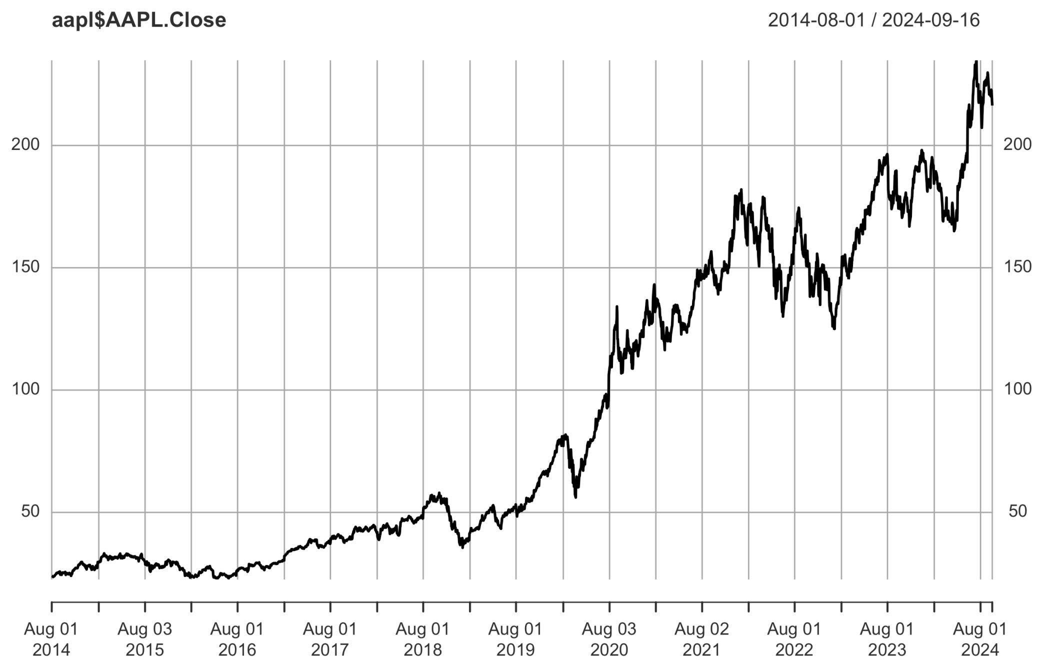
~~~
```{r out.width="50%", fig.cap="include_graphics with `out.width=\"50%\"`." }
knitr::include_graphics(f_name)
~~~
```

\begin{figure}  \caption{include_graphics with out.width="50%".} \end{figure}

aapl$AAPL.Close
2014-08-01 / 2024-09-16
200
150
100
50
Aug 01 2014
Aug 03 2015
Aug 01 2016
Aug 01 2017
Aug 01 2018
Aug 01 2019
Aug 03 2020
Aug 02 2021
Aug 01 2022
Aug 01 2023
Aug 01 2024

```
```{r out.width="100%", fig.cap="include_graphics with `out.width=\"100%\"`." }
knitr::include_graphics(f_name)
```
~~~

```r
knitr::include_graphics(f_name)
~~~



\begin{figure}
\caption{include_graphics with out.width="100%".} \end{figure}

---

Q: How to suppress the following `dev.off()` messages generated by code chunks in `Rmd`?

```
## quartz_off_screen
##                 2
```

A: Enclose `dev.off()` within `invisible()`, or dump the result of `dev.off()` to a garbage variable.

```r
invisible(dev.off())     # opt1
whatever <- dev.off()    # opt2
```

Specify code chunk options `fig.width` and `fig.height` for R-generated figures only.

- Default is `fig.width = 7` and `fig.height = 5` (**in inches**, though actual width will depend on screen resolution). Remember that these settings will default to `rmarkdown` values, not `knitr` values.
- If don't know what size is suitable, can right-click the Plots Viewer and choose "Copy Image Address". Scale by `/100` (in inches) and fill the values to chunk options.

`out.width` and `out.height` apply to both existing images and R-generated figures.

- note that the percentage need to be put in quotes.
- `fig.width` do not scale font, it shows the original font size.
- `out.width` scales the whole figure. Better to use this one. If you want to fix aspect ratio, use `fig.asp=0.6` to set height:width = 6:10.
  - `out.width` keeps the original aspect ratio of the figure and scale the text in the figure too.

    But what most people want is to scale the figure but not the text. For instance, you want to scale your figure to 70% width of page, but you want to keep the original size of text so it is readable.
  - A caveat with `out.width`is that the axis labels and ticks will be so small and hard to read.

Other chunk options related to figures:

`fig.cap=NULL` specify figure captions. Must provide `fig.cap` if you need to cross reference the figure.

See `Fig. \@ref(fig:car-plot)` use code chunk label to cross reference. The chunk label (`car-plot`) provides the identifier for referencing the figure generated by the chunk.

- `Fig.&nbsp;\@ref(fig:logit-regression)` use `&nbsp;` to insert a non-breaking space.

`fig.align="center"` to set figure alignment.

`fig.pos="H"` fix placement.

`fig.asp=0.6` aspect ratio height:width=6:10.

Suggested practice so that you have correct aspect ratio and automatically scaled text and labels in figures. ▯

1. Generate the figure and save to local

   The benefit is that you have full control to adjust the figure as needed, such as font size, and could reuse it later.

   ~~~
   ```{r echo=FALSE, include=FALSE}
   p <- ggplot(contingency_table %>%
               as_tibble() %>%
               mutate(chd69=factor(chd69, levels=c("non-developed", "developed"))),
            aes(x=smoke, y=n, fill=chd69)) +
       geom_bar(position="stack", stat="identity", color="black", linewidth=0.1) +
       scale_fill_grey(start=0.88, end=0.7) +
       labs(y="Frequency") +
       theme(axis.title.x = element_blank(), legend.position = "bottom")
   f_name <- "images/stacked_bar.png"
   ~~~

```
plot_png(p, f_name, 5.17, 5)
```
```

Specify chunk options `include=FALSE` (Do not include code output) to suppress the graphic window information like the following.

```
## quartz_off_screen
##                 2
```

2. Add the figure using

```
```{r scatter-plot, echo=FALSE, fig.cap="Scatter plot of avearge wage against experience.", ou
include_graphics(f_name)
```
```

3. Cross reference

   - `pdf_document`: using `\autoref{fig:scatter-plot}` from `hyperref` package or `Fig. \ref{fig:scatter-plot}` from base latex.

     `hyperref` uses `Figure`, could be changed to `Fig.` by putting the following cmd at the begin of the Rmd.

     ```
     \renewcommand\figureautorefname{Fig.}
     ```

   - `bookdown::html_document2`: using `\@ref(fig:scatter-plot)`.

Latex symbols in Fig. caption

The R code block approach.

- `\\Phi` works. You need to escape the `\` in `\Phi` .
- If there are quotation marks (`"`) in the figure caption, need to escape them using `\"...\"` to distinguish from the outer quotes of the caption parameter.
- You can use regular Markdown syntax in Fig captions, such as using `**Bold**` to make text bold.
- Better to use R code blocks to include figures.

  Note that `include_graphics("https://link-to-Google-drive")` does NOT work for pdf output. Works for html output though.

  If using html tag `<figure>`, the numbering will be messed up. There is only automatic numbering with R code figures.

  Use example:

  ```
  ```{r fig.cap="The $\\Phi$ and $\\phi$ ($f_Z(.)$) functions (CDF and pdf of standard normal)."
  include_graphics("images/Phi_b.png")
  ```
  ```

  Will generate the following Fig **??**.

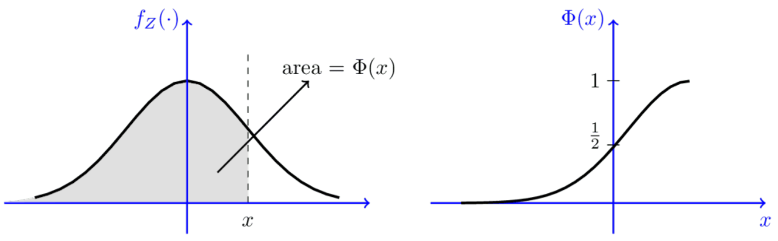


Figure 3.2: The Φ and ϕ ($f_Z(.)$) functions (CDF and pdf of standard normal).
  ```

Alternatively, use **the HTML approach**, and enclose the caption inside `<figcaption>`.

- Benefit: You can type equations as you normally do. Don't need to escape backslashes as using the R code blocks in the example above.
- Drawback: You need to manually add figure numbering.

▯ That means, when you change the order of sections or figures in your webpage, the numbering will be a mess. You need to change all capitals manually.

```
<figure>
<img src="https://drive.google.com/thumbnail?id=1nxfdIKXgZvOqXVSeA3h_hf0yxmsM361l&sz=w1000"
<figcaption>Fig.1 The $\Phi$ and $\phi$ ($f_Z(.)$) functions (CDF and pdf of standard norma
</figure>
```

Fig.1 The $\Phi$ and $\phi$ ($f_Z(.)$) functions (CDF and pdf of standard normal).

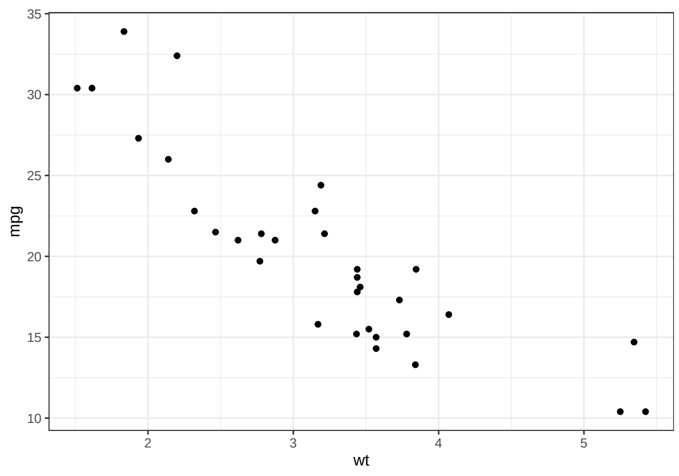
Refer to another figure in figure caption

Just need to use double backslash `\\@ref(fig:xxx)` in the figure caption.

Use example:

We first generate the figure to be referenced.

~~~
```{r firstplot, out.width="60%", fig.cap="Source Figure to be referred to."}
library(ggplot2)
p <- ggplot(mtcars, aes(wt, mpg))
plot_A <- p + geom_point()
plot_A
```
~~~



\begin{figure} \caption{Source Figure to be referenced. **Note that when specifying `out.width="60%"`, the text in the figure is scaled too small.**} \end{figure}

Now a second plot with a reference to Fig.: **??**.

~~~
```{r secondplot, fig.cap = "This is the same as Fig.: \\@ref(fig:firstplot) but now with a
plot_A + geom_line(alpha = .75,col = "red")
```
~~~

3.10 Tables

Cross reference tables

Using bookdown cmd: `\@ref(tab:chunk-label)`.

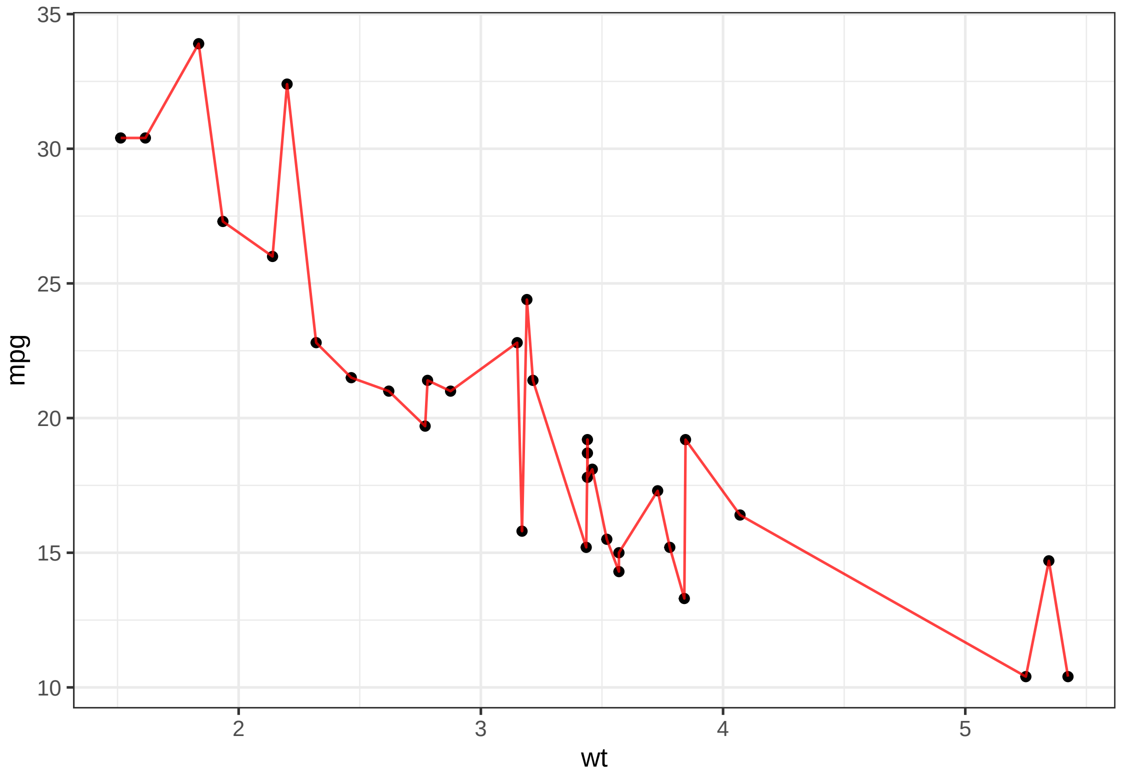


Figure 3.3: This is the same as Fig.: **??** but now with a red line and `out.width="100%"`.

Table 3.6: The mtcars data.

	mpg	cyl	disp	hp	drat
Mazda RX4	21.0	6	160	110	3.90
Mazda RX4 Wag	21.0	6	160	110	3.90
Datsun 710	22.8	4	108	93	3.85
Hornet 4 Drive	21.4	6	258	110	3.08
Hornet Sportabout	18.7	8	360	175	3.15

Note that you must provide `caption` option in `knitr::kable()`. Otherwise the table won't be numbered.

~~~
And see Table \@ref(tab:mtcars).

```{r mtcars, echo=FALSE}
knitr::kable(mtcars[1:5, 1:5], caption = "The mtcars data.")
```
~~~

Refer to the Table **??**.

`knitr::kable(x, format="pipe")` is useful when you want to copy-and-paste R output from console to other document, e.g., markdown.

```
knitr::kable(mtcars[1:5, 1:5], format = "pipe")
|                  |  mpg| cyl| disp|  hp| drat|
|:-----------------|----:|---:|----:|---:|----:|
|Mazda RX4         | 21.0|   6|  160| 110| 3.90|
|Mazda RX4 Wag     | 21.0|   6|  160| 110| 3.90|
|Datsun 710        | 22.8|   4|  108|  93| 3.85|
|Hornet 4 Drive    | 21.4|   6|  258| 110| 3.08|
|Hornet Sportabout | 18.7|   8|  360| 175| 3.15|
```

3.10.1 knitr::kable

`knitr::kable(x, digits, caption=NULL, escape=TRUE)` Create tables in LaTeX, HTML, Markdown and reStructuredText.

- `caption` The table caption. In order to number the table, mut specify the `caption` argument.
- `format` Possible values are `latex`, `html`, `pipe` (Pandoc's pipe tables), `simple` (Pandoc's simple tables), `rst`, and `jira`.

  The value of this argument will be automatically determined if the function is called within a **knitr** document.
- `digits` Maximum number of digits for numeric columns, passed to `round()`.
- `col.names` Rename columns.
- `escape=TRUE` Whether to escape special characters when producing HTML or LaTeX tables. Default is `TRUE`, special characters will either be escaped or substituted. For example, `$` is escaped as `\$`, `_` is escaped as `\_`, and `\` is substituted with `\textbackslash{}`
  - When set to `FALSE`, you have to make sure **yourself** that special characters will not trigger syntax errors in LaTeX or HTML.
  - Common special LaTeX characters include `#`, `%`, `&`, `{`, and `}`. Common special HTML characters include `&`, `<`, `>`, and `"`. You need to be cautious when generating tables with `escape = FALSE`, and make sure you are using the special characters in the right way. It is a very common mistake to use `escape = FALSE` and include `%` or `_` in column names or the caption of a LaTeX table without realizing that they are special.
- `align` Column alignment: a character **vector** consisting of `'l'` (left), `'c'` (center) and/or `'r'` (right).
  - By default or if `align = NULL`, numeric columns are right-aligned, and other columns are left-aligned.
  - If only one character is provided, that will apply to all columns.
  - If a vector is provided, will map to each individual column specifically.
- Missing values (`NA`) in the table are displayed as `NA` by default. If you want to display them with other characters, you can set the option `knitr.kable.NA`, e.g. `options(knitr.kable.NA = '')` in the YAML to hide `NA` values.
- `booktabs = TRUE` use the booktabs package
  - `linesep = ""` remove the extra space after every five rows in kable output (with `booktabs` option)

```
# For Markdown tables, use `pipe` format
> knitr::kable(head(mtcars[, 1:4]), format = "pipe")
|                  |  mpg| cyl| disp|  hp|
|:-----------------|----:|---:|----:|---:|
|Mazda RX4         | 21.0|   6|  160| 110|
|Mazda RX4 Wag     | 21.0|   6|  160| 110|
|Datsun 710        | 22.8|   4|  108|  93|
|Hornet 4 Drive    | 21.4|   6|  258| 110|
|Hornet Sportabout | 18.7|   8|  360| 175|
|Valiant           | 18.1|   6|  225| 105|

# For Plain tables in txt, `simple` is useful
> knitr::kable(head(mtcars[, 1:4]), format = "simple")
                      mpg   cyl   disp    hp
```

```
------------------  -----  ----  -----  ----
Mazda RX4            21.0     6    160   110
Mazda RX4 Wag        21.0     6    160   110
Datsun 710           22.8     4    108    93
Hornet 4 Drive       21.4     6    258   110
Hornet Sportabout    18.7     8    360   175
Valiant              18.1     6    225   105
```

3.10.2 Data frame printing

To show the `tibble` information (number of row/columns, and group information) along with paged output, we can write a custom function by modifying the `print.paged_df` function (which is used internally by rmarkdown for the `df_print` feature) and use CSS to nicely format the output.

https://stackoverflow.com/a/76014674/10108921

Paged df

- https://bookdown.org/yihui/rmarkdown/html-document.html#tab:paged
- https://github.com/rstudio/rmarkdown/issues/1403

```
---
title: "Use caption with df_print set to page"
date: "2025-05-13"
output:
  bookdown::html_document2:
    df_print: paged
---
```

When the `df_print` option is set to `paged`, tables are printed as HTML tables with support for pagination over rows and columns.

The possible values of the `df_print` option for the `html_document` format.

Option	Description
`default`	Call the `print.data.frame` generic method; console output prefixed by `##`;
`kable`	Use the `knitr::kable` function; looks nice but with no navigation for rows and columns, neither column types.
`tibble`	Use the `tibble::print.tbl_df` function, this provides groups and counts of rows and columns info as if printing a `tibble`.
`paged`	Use `rmarkdown::paged_table` to create a pageable table; `paged` looks best but slows down compilation significantly;
A custom function	Use the function to create the table

The possible values of the `df_print` option for the `pdf_document` format: `default`, `kable`, `tibble`, `paged`, or a custom function.

~~~
paged print

```{r echo=TRUE, paged.print=TRUE}
ggplot2::diamonds
```
~~~

~~~
default output

```{r echo=TRUE, paged.print=FALSE}
ggplot2::diamonds
```

kable output

```{r echo=TRUE}
knitr::kable(ggplot2::diamonds[1:10, ])
```
~~~

Note that `kable` output doesn't provide tibble information.

Available options for `paged` tables:

Option	Description
max.print	The number of rows to print.
rows.print	The number of rows to display.
cols.print	The number of columns to display.
cols.min.print	The minimum number of columns to display.
pages.print	The number of pages to display under page navigation.
paged.print	When set to `FALSE` turns off paged tables.
rownames.print	When set to `FALSE` turns off row names.

These options are specified in each chunk like below:

~~~
```{r cols.print=3, rows.print=3}
mtcars
```
~~~

For **pdf_document**, it is possible to write LaTex code directly.

~~~
```{=latex}
\begin{tabular}{ll}
A & B \\
A & B \\
\end{tabular}
```
~~~

Do not forget the equal sign before `latex`, i.e., it is `=latex` instead of `latex`.

3.10.3 Stargazer

`stargazer` print nice tables in `Rmd` documents and `R` scripts:

- Passing a data frame to stargazer package creates a **summary statistic table**.
- Passing a regression object creates a nice **regression table**.
- Support tables output in multiple formats: `text`, `latex`, and `html`.
  - In `R` scripts, use `type = "text"` for a quick view of results.
- `stargaer` does NOT work with `anova` table, use `pander::pander` instead.

Text table

Specify `stargazer(type = "text")`