

# R and R Studio Reference Guides and "Cheat Sheet" Compilation

The following is a set of reference guides and "cheat sheets" that has been gathered from various resources on the Internet. All of these sheets are used for academic purposes and should not be reproduced, distributed, or copied without permission from the sheet designer / author (s). These are intended as shortcuts and aides for the use of R and R Studio, along with R Script, R Markdown, and some of the most popular R packages.

- 2 to 6 • R Markdown Reference Guide | [rmarkdown.rstudio.com](http://rmarkdown.rstudio.com) | updated 2014-10
- 7 to 8 • Base R Cheat Sheet | [rstudio.com](http://rstudio.com) | updated 2015-03
- 9 to 10 • Data Import with tidyverse Cheat Sheet | [readr.tidyverse.org](http://readr.tidyverse.org) | updated 2021-08
- 11 to 12 • rmarkdown Cheat Sheet | [rmarkdown.rstudio.com](http://rmarkdown.rstudio.com) | updated 2021-08
- 13 to 14 • R Studio IDE Cheat Sheet | [rstudio.com](http://rstudio.com) | updated 2021-07
- 15 to 16 • Data transformation with dplyr Cheat Sheet | [dplyr.tidyverse.org](http://dplyr.tidyverse.org) | updated 2021-07
- 17 to 18 • Data tidying with dplyr Cheat Sheet | [dplyr.tidyverse.org](http://dplyr.tidyverse.org) | Updated 2021-07
- 19 to 20 • Data visualization with ggplot Cheat Sheet | [ggplot2.tidyverse.org](http://ggplot2.tidyverse.org) | Updated 2021-08
- 21 to 22 • How Big is Your Graph | <https://www.rstudio.com/resources/cheatsheets/> | 2017-07
- 23 to 24 • R Syntax Comparison Cheat Sheet | [science.smith.edu/~amcnamara](http://science.smith.edu/~amcnamara) | Updated 2018-01
- 25 to 26 • Tabular reporting with flextable Cheat Sheet | [ardata-fr.github.io/flextable-book/](http://ardata-fr.github.io/flextable-book/) | 2021-03



# R Markdown Reference Guide

Learn more about R Markdown at [rmarkdown.rstudio.com](http://rmarkdown.rstudio.com)

Learn more about Interactive Docs at [shiny.rstudio.com/articles](http://shiny.rstudio.com/articles)

Contents:

1. [Markdown Syntax](#)
2. [Knitr chunk options](#)
3. [Pandoc options](#)

## Syntax

Plain text

End a line with two spaces to start a new paragraph.

\*italics\* and \_italics\_

\*\*bold\*\* and \_bold\_

superscript<sup>2</sup>

~~strikethrough~~

[link] ([www.rstudio.com](http://www.rstudio.com))

# Header 1

## Header 2

### Header 3

#### Header 4

##### Header 5

##### Header 6

endash: --

emdash: ---

ellipsis: ...

inline equation: \$A = \pi \* r^2\$

image: )

horizontal rule (or slide break):

\*\*\*

> block quote

\* unordered list  
\* item 2

+ sub-item 1  
+ sub-item 2

1. ordered list

2. item 2

+ sub-item 1  
+ sub-item 2

Table Header | Second Header

----- | -----

Table Cell | Cell 2

Cell 3 | Cell 4

## Becomes

Plain text

End a line with two spaces to start a new paragraph.

*italics* and *italics*

**bold** and **bold**

superscript<sup>2</sup>

~~strikethrough~~

[link](#)

## Header 1

## Header 2

## Header 3

## Header 4

## Header 5

## Header 6

endash: –

emdash: —

ellipsis: ...

inline equation:  $A = \pi * r^2$



horizontal rule (or slide break):

## block quote

- unordered list
- item 2
  - sub-item 1
  - sub-item 2

1. ordered list

2. item 2
- sub-item 1
  - sub-item 2

**Table Header**

**Second Header**

Table Cell

Cell 2

Cell 3

Cell 4



# R Markdown Reference Guide

Learn more about R Markdown at [rmarkdown.rstudio.com](http://rmarkdown.rstudio.com)

Learn more about Interactive Docs at [shiny.rstudio.com/articles](http://shiny.rstudio.com/articles)

## Contents:

1. Markdown Syntax
2. Knitr chunk options
3. Pandoc options

## Syntax

Make a code chunk with three back ticks followed by an r in braces. End the chunk with three back ticks:

```
```{r}
paste("Hello", "World!")
```
```

Place code inline with a single back ticks. The first back tick must be followed by an R, like this `r paste("Hello", "World!")`.

Add chunk options within braces. For example, `echo=FALSE` will prevent source code from being displayed:

```
```{r eval=TRUE, echo=FALSE}
paste("Hello", "World!")
```
```

Learn more about chunk options at <http://yihui.name/knitr/options>

## Becomes

Make a code chunk with three back ticks followed by an r in braces. End the chunk with three back ticks:

```
paste("Hello", "World!")
```

```
## [1] "Hello World!"
```

Place code inline with a single back ticks. The first back tick must be followed by an R, like this Hello World!.

Add chunk options within braces. For example, `echo=FALSE` will prevent source code from being displayed:

```
## [1] "Hello World!"
```

| option                   | default value | description   |
|--------------------------|---------------|---|
| <b>Code evaluation</b>   |               |   |
| <code>child</code>       | NULL          | A character vector of filenames. Knitr will knit the files and place them into the main document.   |
| <code>code</code>        | NULL          | Set to R code. Knitr will replace the code in the chunk with the code in the code option.   |
| <code>engine</code>      | 'R'           | Knitr will evaluate the chunk in the named language, e.g. <code>engine = 'python'</code> . Run <code>names(knitr::knit_engines\$get())</code> to see supported languages.   |
| <code>eval</code>        | TRUE          | If FALSE, knitr will not run the code in the code chunk.  |
| <code>include</code>     | TRUE          | If FALSE, knitr will run the chunk but not include the chunk in the final document.   |
| <code>purl</code>        | TRUE          | If FALSE, knitr will not include the chunk when running <code>purl()</code> to extract the source code.   |
| <b>Results</b>           |               |   |
| <code>collapse</code>    | FALSE         | If TRUE, knitr will collapse all the source and output blocks created by the chunk into a single block.   |
| <code>echo</code>        | TRUE          | If FALSE, knitr will not display the code in the code chunk above it's results in the final document.   |
| <code>results</code>     | 'markup'      | If 'hide', knitr will not display the code's results in the final document. If 'hold', knitr will delay displaying all output pieces until the end of the chunk. If 'asis', knitr will pass through results without reformatting them (useful if results return raw HTML, etc.) |
| <code>error</code>       | TRUE          | If FALSE, knitr will not display any error messages generated by the code.  |
| <code>message</code>     | TRUE          | If FALSE, knitr will not display any messages generated by the code.  |
| <code>warning</code>     | TRUE          | If FALSE, knitr will not display any warning messages generated by the code.  |
| <b>Code Decoration</b>   |               |   |
| <code>comment</code>     | '##'          | A character string. Knitr will append the string to the start of each line of results in the final document.  |
| <code>highlight</code>   | TRUE          | If TRUE, knitr will highlight the source code in the final output.  |
| <code>prompt</code>      | FALSE         | If TRUE, knitr will add > to the start of each line of code displayed in the final document.  |
| <code>strip.white</code> | TRUE          | If TRUE, knitr will remove white spaces that appear at the beginning or end of a code chunk.  |
| <code>tidy</code>        | FALSE         | If TRUE, knitr will tidy code chunks for display with the <code>tidy_source()</code> function in the <code>formatR</code> package.  |



# R Markdown Reference Guide

Learn more about R Markdown at [rmarkdown.rstudio.com](http://rmarkdown.rstudio.com)

Learn more about Interactive Docs at [shiny.rstudio.com/articles](http://shiny.rstudio.com/articles)

## Contents:

1. **Markdown Syntax**
2. **Knitr chunk options**
3. **Pandoc options**

## Chunk options (Continued)

| option                                   | default value   | description  |
|--|-----------------|--|
| <b>Chunks</b>                            |                 |  |
| <code>opts.label</code>                  | NULL            | The label of options set in <code>knitr:: opts_template()</code> to use with the chunk.  |
| <code>R.options</code>                   | NULL            | Local R options to use with the chunk. Options are set with <code>options()</code> at start of chunk. Defaults are restored at end.  |
| <code>ref.label</code>                   | NULL            | A character vector of labels of the chunks from which the code of the current chunk is inherited.  |
| <b>Cache</b>                             |                 |  |
| <code>autodep</code>                     | FALSE           | If TRUE, knitr will attempt to figure out dependencies between chunks automatically by analyzing object names.   |
| <code>cache</code>                       | FALSE           | If TRUE, knitr will cache the results to reuse in future knits. Knitr will reuse the results until the code chunk is altered.  |
| <code>cache.comments</code>              | NULL            | If FALSE, knitr will not rerun the chunk if only a code comment has changed.   |
| <code>cache.lazy</code>                  | TRUE            | If TRUE, knitr will use <code>lazylload()</code> to load objects in chunk. If FALSE, knitr will use <code>load()</code> to load objects in chunk.  |
| <code>cache.path</code>                  | 'cache/'        | A file path to the directory to store cached results in. Path should begin in the directory that the .Rmd file is saved in.  |
| <code>cache.vars</code>                  | NULL            | A character vector of object names to cache if you do not wish to cache each object in the chunk.  |
| <code>dependson</code>                   | NULL            | A character vector of chunk labels to specify which other chunks a chunk depends on. Knitr will update a cached chunk if its dependencies change.  |
| <b>Animation</b>                         |                 |  |
| <code>anipots</code>                     | 'controls,loop' | Extra options for animations (see the <code>animate</code> package).   |
| <code>interval</code>                    | 1               | The number of seconds to pause between animation frames.   |
| <b>Plots</b>                             |                 |  |
| <code>dev</code>                         | 'png'           | The R function name that will be used as a graphical device to record plots, e.g. <code>dev='CairoPDF'</code> .  |
| <code>dev.args</code>                    | NULL            | Arguments to be passed to the device, e.g. <code>dev.args=list(bg='yellow', pointsize=10)</code> .   |
| <code>dpi</code>                         | 72              | A number for knitr to use as the dots per inch (dpi) in graphics (when applicable).  |
| <code>external</code>                    | TRUE            | If TRUE, knitr will externalize tikz graphics to save LaTex compilation time (only for the <code>tikzDevice::tikz()</code> device).  |
| <code>fig.align</code>                   | 'default'       | How to align graphics in the final document. One of 'left', 'right', or 'center'.  |
| <code>fig.cap</code>                     | NULL            | A character string to be used as a figure caption in LaTex.  |
| <code>fig.env</code>                     | 'figure'        | The Latex environment for figures.   |
| <code>fig.ext</code>                     | NULL            | The file extension for figure output, e.g. <code>fig.ext='png'</code> .  |
| <code>fig.height, fig.width</code>       | 7               | The width and height to use in R for plots created by the chunk (in inches).   |
| <code>fig.keep</code>                    | 'high'          | If 'high', knitr will merge low-level changes into high level plots. If 'all', knitr will keep all plots (low-level changes may produce new plots). If 'first', knitr will keep the first plot only. If 'last', knitr will keep the last plot only. If 'none', knitr will discard all plots.           |
| <code>fig.lp</code>                      | 'fig:'          | A prefix to be used for figure labels in latex.  |
| <code>fig.path</code>                    | 'figure/'       | A file path to the directory where knitr should store the graphics files created by the chunk.   |
| <code>fig.pos</code>                     | "               | A character string to be used as the figure position arrangement in LaTex.   |
| <code>fig.process</code>                 | NULL            | A function to post-process a figure file. Should take a filename and return a filename of a new figure source.   |
| <code>fig.retina</code>                  | 1               | Dpi multiplier for displaying HTML output on retina screens.   |
| <code>fig.scap</code>                    | NULL            | A character string to be used as a short figure caption.   |
| <code>fig.subcap</code>                  | NULL            | A character string to be used as captions in sub-figures in LaTex.   |
| <code>fig.show</code>                    | 'asis'          | If 'hide', knitr will generate the plots created in the chunk, but not include them in the final document. If 'hold', knitr will delay displaying the plots created by the chunk until the end of the chunk. If 'animate', knitr will combine all of the plots created by the chunk into an animation. |
| <code>fig.showtext</code>                | NULL            | If TRUE, knitr will call <code>showtext::showtext.begin()</code> before drawing plots.   |
| <code>out.extra</code>                   | NULL            | A character string of extra options for figures to be passed to LaTex or HTML.   |
| <code>out.height, out.width</code>       | NULL            | The width and height to scale plots to in the final output. Can be in units recognized by output, e.g. <code>8\\ linewidth, 50px</code>  |
| <code>resize.height, resize.width</code> | NULL            | The width and height to resize like graphics in LaTex, passed to <code>\resizebox{ }{ }</code> .   |
| <code>sanitize</code>                    | FALSE           | If TRUE, knitr will sanitize like graphics for LaTex.  |



# R Markdown Reference Guide

Learn more about R Markdown at [rmarkdown.rstudio.com](http://rmarkdown.rstudio.com)

Learn more about Interactive Docs at [shiny.rstudio.com/articles](http://shiny.rstudio.com/articles)

## Contents:

1. Markdown Syntax
2. Knitr chunk options
3. Pandoc options

| Templates   | Basic YAML | Template options | Latex options | Interactive Docs |
|---|------------|------------------|---------------|------------------|
| html_document<br>pdf_document<br>word_document<br>md_document<br>ioslides_presentation<br>slidy_presentation<br>beamer_presentation | ---        | ---              | ---           | ---              |

## Syntax for slide formats (ioslides, slidy, beamer)

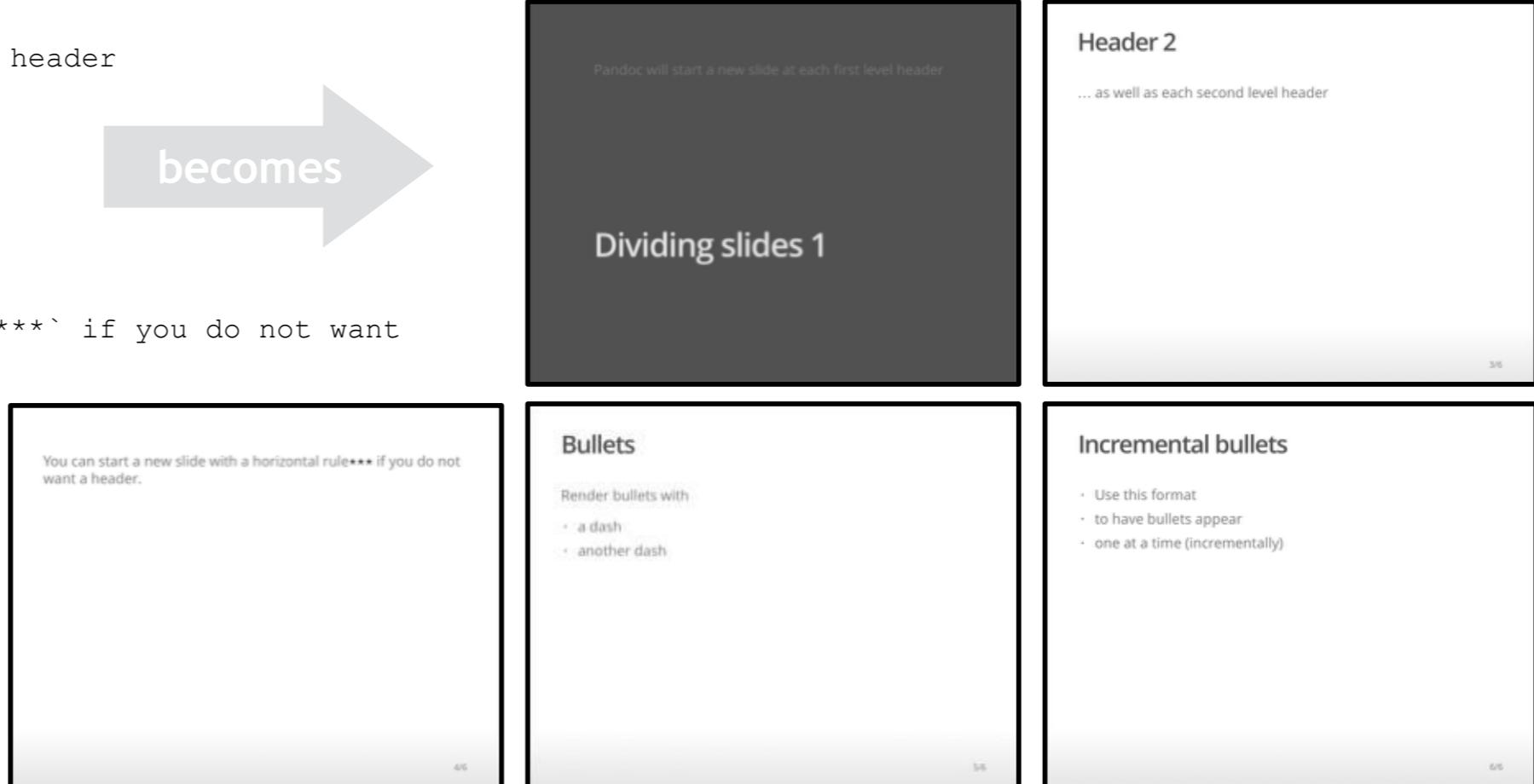
```
# Dividing slides 1
Pandoc will start a new slide at each first level header
## Header 2
... as well as each second level header
***

You can start a new slide with a horizontal rule`***` if you do not want
a header.

## Bullets
Render bullets with
- a dash
- another dash

## Incremental bullets
>- Use this format
>- to have bullets appear
>- one at a time (incrementally)
```

becomes



## Slide display modes

Press a key below during presentation to enter display mode. Press `esc` to exit display mode.

### ioslides

- f** - enable fullscreen mode
- w** - toggle widescreen mode
- o** - enable overview mode
- h** - enable code highlight mode
- p** - show presenter notes

### slidy

- C** - show table of contents
- F** - toggle display of the footer
- A** - toggle display of current vs all slides
- S** - make fonts smaller
- B** - make fonts bigger

## Top level options to customize LaTex (pdf) output

| option  | description   |
|---|---|
| <b>lang</b>                                   | Document language code  |
| <b>fontsize</b>                               | Font size (e.g. 10pt, 11pt, 12 pt)  |
| <b>documentclass</b>                          | Latex document class (e.g. article)   |
| <b>classoption</b>                            | Option for document class (e.g. oneside); may be repeated                                       |
| <b>geometry</b>                               | Options for geometry class (e.g. margin=1in); may be repeated                                   |
| <b>mainfont, sansfont, monofont, mathfont</b> | Document fonts (works only with xelatex and lualatex, see the <code>latex_engine</code> option) |
| <b>linkcolor, urlcolor, citecolor</b>         | Color for internal, external, and citation links (red, green, magenta, cyan, blue, black)       |



# R Markdown Reference Guide

Learn more about R Markdown at [rmarkdown.rstudio.com](http://rmarkdown.rstudio.com)

Learn more about Interactive Docs at [shiny.rstudio.com/articles](http://shiny.rstudio.com/articles)

## Contents:

1. Markdown Syntax
2. Knitr chunk options
3. Pandoc options

| option                 | html | pdf | word | md | ioslides | slidy | beamer | description   |
|------------------------|------|-----|------|----|----------|-------|--------|---|
| <b>colortheme</b>      |      |     |      |    |          | X     |        | Beamer color theme to use (e.g., <code>colortheme: "dolphin"</code> ).  |
| <b>css</b>             | X    |     |      |    | X        | X     |        | Filepath to CSS style to use to style document (e.g., <code>css: styles.css</code> ).   |
| <b>duration</b>        |      |     |      |    |          | X     |        | Add a countdown timer (in minutes) to footer of slides (e.g., <code>duration: 45</code> ).  |
| <b>fig_caption</b>     | X    | X   | X    |    | X        | X     | X      | Should figures be rendered with captions?   |
| <b>fig_crop</b>        |      | X   |      |    |          | X     |        | Should pdfcrop utility be automatically applied to figures (when available)?  |
| <b>fig_height</b>      | X    | X   | X    | X  | X        | X     | X      | Default figure height (in inches) for document.   |
| <b>fig_retina</b>      | X    |     |      | X  | X        | X     |        | Scaling to perform for retina displays (e.g., <code>fig_retina: 2</code> ).   |
| <b>fig_width</b>       | X    | X   | X    | X  | X        | X     | X      | Default figure width (in inches) for document.  |
| <b>font_adjustment</b> |      |     |      |    |          | X     |        | Increase or decrease font size for entire presentation (e.g., <code>font_adjustment: -1</code> ).   |
| <b>fonttheme</b>       |      |     |      |    |          | X     |        | Beamer font theme to use (e.g., <code>fonttheme: "structurebold"</code> ).  |
| <b>footer</b>          |      |     |      |    | X        |       |        | Text to add to footer of each slide (e.g., <code>footer: "Copyright (c) 2014 RStudio"</code> ).   |
| <b>highlight</b>       | X    | X   |      |    | X        | X     |        | Syntax highlighting style (e.g. <code>"tango"</code> , <code>"pygments"</code> , <code>"kate"</code> , <code>"zenburn"</code> , and <code>"rstudio"</code> ).   |
| <b>includes</b>        | X    | X   |      | X  | X        | X     |        | See below   |
| <b>-in_header</b>      | X    | X   |      |    | X        | X     | X      | File of content to place in document header (e.g., <code>in_header: header.html</code> ).   |
| <b>-before_body</b>    | X    | X   |      |    | X        | X     | X      | File of content to place before document body (e.g., <code>before_body:</code>  |
| <b>-after_body</b>     | X    | X   |      |    | X        | X     | X      | File of content to place after document body (e.g., <code>after_body: doc_suffix.html</code> ).   |
| <b>incremental</b>     |      |     |      |    | X        | X     | X      | Should bullets appear one at a time (on presenter mouse clicks)?  |
| <b>keep_md</b>         | X    |     |      |    | X        | X     |        | Save a copy of .md file that contains knitr output (in addition to the .Rmd and HTML files)?  |
| <b>keep_tex</b>        |      | X   |      |    |          | X     |        | Save a copy of .tex file that contains knitr output (in addition to the .Rmd and PDF files)?  |
| <b>latex_engine</b>    |      | X   |      |    |          |       |        | Engine to render latex. Should be one of <code>"pdflatex"</code> , <code>"xelatex"</code> , and <code>"lualatex"</code> .   |
| <b>lib_dir</b>         | X    |     |      |    | X        | X     |        | Directory of dependency files to use (Bootstrap, MathJax, etc.) (e.g., <code>lib_dir: libs</code> ).  |
| <b>logo</b>            |      |     |      |    | X        |       |        | File path to a logo (at least 128 x 128) to add to presentation (e.g., <code>logo: logo.png</code> ).   |
| <b>mathjax</b>         | X    |     |      |    | X        | X     |        | Set to <code>local</code> or a URL to use a local/URL version of MathJax to render equations  |
| <b>number_section</b>  | X    | X   |      |    |          |       |        | Add section numbering to headers (e.g., <code>number_sections: true</code> ).   |
| <b>pandoc_args</b>     | X    | X   | X    | X  | X        | X     | X      | Arguments to pass to Pandoc (e.g., <code>pandoc_args: ["--title-prefix", "Foo"]</code> ).   |
| <b>preserve_yaml</b>   |      |     |      | X  |          |       |        | Preserve YAML front matter in final document?   |
| <b>reference_docx</b>  |      |     | X    |    |          |       |        | A .docx file whose styles should be copied to use (e.g., <code>reference_docx:</code>   |
| <b>self_contained</b>  | X    |     |      |    | X        | X     |        | Embed dependencies into the doc? Set to <code>false</code> to keep dependencies in external files.  |
| <b>slide_level</b>     |      |     |      |    | X        |       |        | The lowest heading level that defines individual slides (e.g., <code>slide_level: 2</code> ).   |
| <b>smaller</b>         |      |     |      |    | X        |       |        | Use the smaller font size in the presentation?  |
| <b>smart</b>           | X    |     |      |    | X        | X     |        | Convert straight quotes to curly, dashes to em-dashes, ... to ellipses, and so on?  |
| <b>template</b>        | X    | X   |      |    | X        | X     |        | Pandoc template to use when rendering file (e.g., <code>template:</code>  |
| <b>theme</b>           | X    |     |      |    |          | X     |        | Bootswatch or Beamer theme to use for page. Valid bootswatch themes include <code>"cerulean"</code> , <code>"journal"</code> , <code>"flatly"</code> , <code>"readable"</code> , <code>"spacelab"</code> , <code>"united"</code> , and <code>"cosmo"</code> . |
| <b>toc</b>             | X    | X   | X    |    |          | X     |        | Add a table of contents at start of document? (e.g., <code>toc: true</code> ).  |
| <b>toc_depth</b>       | X    | X   | X    |    |          |       |        | The lowest level of headings to add to table of contents (e.g., <code>toc_depth: 2</code> ).  |
| <b>transition</b>      |      |     |      |    | X        |       |        | Speed of slide transitions should be <code>"slower"</code> , <code>"faster"</code> or a number in seconds.  |
| <b>variant</b>         |      |     | X    |    |          |       |        | The flavor of markdown to use; one of <code>"markdown"</code> , <code>"markdown_strict"</code> , <code>"markdown_github"</code> , <code>"markdown_mmd"</code> , and <code>"markdown_phpextra"</code>  |
| <b>widescreen</b>      |      |     |      | X  |          |       |        | Display presentation in widescreen format?  |

# Base R Cheat Sheet

## Getting Help

### Accessing the help files

?mean

Get help of a particular function.

help.search('weighted mean')

Search the help files for a word or phrase.

help(package = 'dplyr')

Find help for a package.

### More about an object

str(iris)

Get a summary of an object's structure.

class(iris)

Find the class an object belongs to.

## Using Packages

install.packages('dplyr')

Download and install a package from CRAN.

library(dplyr)

Load the package into the session, making all its functions available to use.

dplyr::select

Use a particular function from a package.

data(iris)

Load a built-in dataset into the environment.

## Working Directory

getwd()

Find the current working directory (where inputs are found and outputs are sent).

setwd('C://file/path')

Change the current working directory.

Use projects in RStudio to set the working directory to the folder you are working in.

| Vectors                   |                                  |                             | Programming                  |                             |                                       |                                  |                         |                          |
|---------------------------|----------------------------------|-----------------------------|------------------------------|-----------------------------|---------------------------------------|----------------------------------|-------------------------|--------------------------|
| Creating Vectors          |                                  |                             | For Loop                     |                             |                                       | While Loop                       |                         |                          |
| c(2, 4, 6)                | 2 4 6                            | Join elements into a vector | for (variable in sequence) { | Do something                | }                                     | while (condition) {              | Do something            | }                        |
| 2:6                       | 2 3 4 5 6                        | An integer sequence         |                              |                             |                                       |                                  |                         |                          |
| seq(2, 3, by=0.5)         | 2.0 2.5 3.0                      | A complex sequence          |                              |                             |                                       |                                  |                         |                          |
| rep(1:2, times=3)         | 1 2 1 2 1 2                      | Repeat a vector             | for (i in 1:4){              | j <- i + 10                 | print(j)                              | while (i < 5){                   | print(i)                | i <- i + 1               |
| rep(1:2, each=3)          | 1 1 1 2 2 2                      | Repeat elements of a vector |                              |                             | }                                     |                                  |                         | }                        |
| Vector Functions          |                                  |                             |                              |                             |                                       |                                  |                         |                          |
| sort(x)                   | rev(x)                           |                             | if (condition) {             | Do something                |                                       | function_name <- function(var) { | Do something            |                          |
| Return x sorted.          | Return x reversed.               |                             | } else {                     | Do something different      |                                       | return(new_variable)             |                         |                          |
| table(x)                  | unique(x)                        | See counts of values.       |                              |                             |                                       |                                  |                         |                          |
| Selecting Vector Elements |                                  |                             |                              |                             |                                       |                                  |                         |                          |
| By Position               |                                  |                             | If Statements                |                             |                                       | Functions                        |                         |                          |
| x[4]                      | The fourth element.              |                             | if (i > 3){                  | print('Yes')                |                                       | function_name <- function(x) {   |                         |                          |
| x[-4]                     | All but the fourth.              |                             | } else {                     | print('No')                 |                                       | Do something                     |                         |                          |
| x[2:4]                    | Elements two to four.            |                             |                              |                             |                                       | return(new_variable)             |                         |                          |
| x[-(2:4)]                 | All elements except two to four. |                             |                              |                             |                                       |                                  |                         |                          |
| x[c(1, 5)]                | Elements one and five.           |                             |                              |                             |                                       |                                  |                         |                          |
| By Value                  |                                  |                             |                              |                             |                                       |                                  |                         |                          |
| x[x == 10]                | Elements which are equal to 10.  |                             | Reading and Writing Data     |                             |                                       | Example                          |                         |                          |
| x[x < 0]                  | All elements less than zero.     |                             | Input                        | Ouput                       | Description                           | if (i > 3){                      | square <- function(x) { |                          |
| x[x %in% c(1, 2, 5)]      | Elements in the set 1, 2, 5.     |                             | df <- read.table('file.txt') | write.table(df, 'file.txt') | Read and write a delimited text file. | print('Yes')                     | squared <- x*x          |                          |
| Named Vectors             |                                  |                             |                              |                             |                                       | } else {                         | return(squared)         |                          |
| x['apple']                | Element with name 'apple'.       |                             |                              |                             |                                       |                                  |                         |                          |
| Conditions                |                                  |                             | Also see the readr package.  |                             |                                       | Description                      |                         |                          |
| a == b                    | Are equal                        | a > b                       | a == b                       | Are equal                   | a > b                                 | Greater than                     | a >= b                  | Greater than or equal to |
| a != b                    | Not equal                        | a < b                       | a > b                        | Greater than                | a >= b                                | Less than                        | a <= b                  | Less than or equal to    |
|                           |                                  |                             | a < b                        | Less than                   | a <= b                                |                                  | is.na(a)                | Is missing               |
|                           |                                  |                             |                              |                             |                                       |                                  | is.null(a)              | Is null                  |

## Types

Converting between common data types in R. Can always go from a higher value in the table to a lower value.

|              |                                 |   |
|--------------|---------------------------------|---|
| as.logical   | TRUE, FALSE, TRUE               | Boolean values (TRUE or FALSE).   |
| as.numeric   | 1, 0, 1                         | Integers or floating point numbers.                                       |
| as.character | '1', '0', '1'                   | Character strings. Generally preferred to factors.                        |
| as.factor    | '1', '0', '1', levels: '1', '0' | Character strings with preset levels. Needed for some statistical models. |

## Maths Functions

|              |                                 |             |                         |
|--------------|---------------------------------|-------------|-------------------------|
| log(x)       | Natural log.                    | sum(x)      | Sum.                    |
| exp(x)       | Exponential.                    | mean(x)     | Mean.                   |
| max(x)       | Largest element.                | median(x)   | Median.                 |
| min(x)       | Smallest element.               | quantile(x) | Percentage quantiles.   |
| round(x, n)  | Round to n decimal places.      | rank(x)     | Rank of elements.       |
| signif(x, n) | Round to n significant figures. | var(x)      | The variance.           |
| cor(x, y)    | Correlation.                    | sd(x)       | The standard deviation. |

## Variable Assignment

```
> a <- 'apple'  
> a  
[1] 'apple'
```

## The Environment

|                 |  |
|-----------------|--|
| ls()            | List all variables in the environment.     |
| rm(x)           | Remove x from the environment.             |
| rm(list = ls()) | Remove all variables from the environment. |

You can use the environment panel in RStudio to browse variables in your environment.

## Matrices

m <- matrix(x, nrow = 3, ncol = 3)  
Create a matrix from x.

|  |                             |                                  |
|--|-----------------------------|----------------------------------|
|  | m[2, ] - Select a row       | t(m) Transpose                   |
|  | m[, 1] - Select a column    | m %*% n Matrix Multiplication    |
|  | m[2, 3] - Select an element | solve(m, n) Find x in: m * x = n |

## Lists

l <- list(x = 1:5, y = c('a', 'b'))  
A list is a collection of elements which can be of different types.

|                      |                                       |                  |                                     |
|----------------------|---------------------------------------|------------------|-------------------------------------|
| l[[2]]               | l[1]                                  | l\$x             | l['y']                              |
| Second element of l. | New list with only the first element. | Element named x. | New list with only element named y. |

Also see the [dplyr package](#).

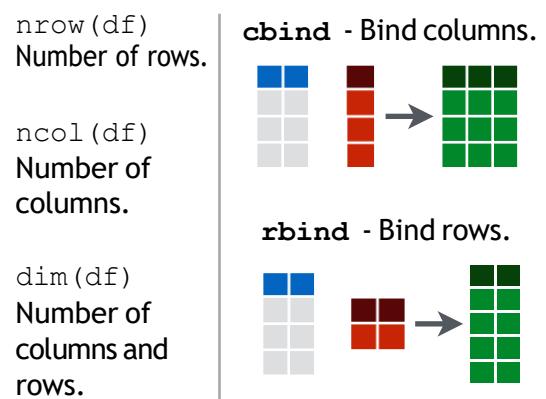
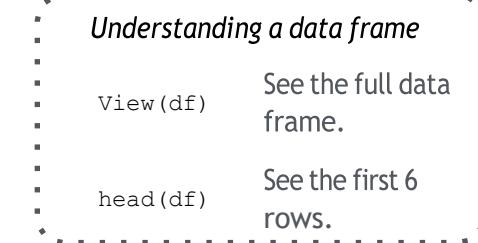
## Data Frames

df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))  
A special case of a list where all elements are the same length.

| x | y |
|---|---|
| 1 | a |
| 2 | b |
| 3 | c |

### Matrix subsetting

|          |  |
|----------|--|
| df[, 2]  |  |
| df[2, ]  |  |
| df[2, 2] |  |



## Strings

|                           |                                       |
|---------------------------|---------------------------------------|
| paste(x, y, sep = ' ')    | Join multiple vectors together.       |
| paste(x, collapse = ' ')  | Join elements of a vector together.   |
| grep(pattern, x)          | Find regular expression matches in x. |
| gsub(pattern, replace, x) | Replace matches in x with a string.   |
| toupper(x)                | Convert to uppercase.                 |
| tolower(x)                | Convert to lowercase.                 |
| nchar(x)                  | Number of characters in a string.     |

Also see the [stringr package](#).

## Factors

|                    |  |
|--------------------|--|
| factor(x)          | Turn a vector into a factor. Can set the levels of the factor and the order. |
| cut(x, breaks = 4) | Turn a numeric vector into a factor by 'cutting' into sections.              |

## Statistics

|                     |  |                                      |
|---------------------|--|--------------------------------------|
| lm(y ~ x, data=df)  | Linear model.                              | prop.test                            |
| glm(y ~ x, data=df) | Generalised linear model.                  | Test for a difference between means. |
| summary             | Get more detailed information out a model. | pairwise.t.test                      |
|                     |  | Perform a t-test for paired data.    |
|                     |  | aov                                  |
|                     |  | Analysis of variance.                |

## Distributions

| Random Variates | Density Function | Cumulative Distribution | Quantile |
|-----------------|------------------|-------------------------|----------|
| Normal          | rnorm            | dnorm                   | pnorm    |
| Poisson         | rpois            | dpois                   | ppois    |
| Binomial        | rbinom           | dbinom                  | pbinom   |
| Uniform         | runif            | dunif                   | qunif    |

Also see the [ggplot2 package](#).

## Plotting

|  |            |                        |  |
|--|------------|------------------------|--|
|  | plot(x)    | Values of x in order.  |  |
|  | plot(x, y) | Values of x against y. |  |
|  | hist(x)    | Histogram of x.        |  |

## Dates

See the [lubridate package](#).



# Data import with the tidyverse : : CHEAT SHEET

## Read Tabular Data with readr

```
read_*(file, col_names = TRUE, col_types = NULL, col_select = NULL, id = NULL, locale, n_max = Inf,
      skip = 0, na = c("", "NA"), guess_max = min(1000, n_max), show_col_types = TRUE) See ?read_delim
```

| A B C | 1 2 3 | 4 5 NA |
|-------|-------|--------|
| 1     | 2     | 3      |
| 4     | 5     | NA     |

`read_delim("file.txt", delim = "|")` Read files with any delimiter. If no delimiter is specified, it will automatically guess.

To make file.txt, run: `write_file("A|B|C\n1|2|3\n4|5|NA", file = "file.txt")`

| A,B,C | 1,2,3 | 4,5,NA |
|-------|-------|--------|
| 1     | 2     | 3      |
| 4     | 5     | NA     |

`read_csv("file.csv")` Read a comma delimited file with period decimal marks.

`write_file("A,B,C\n1,2,3\n4,5,NA", file = "file.csv")`

| A;B;C | 1,5;2;3 | 4,5;5;NA |
|-------|---------|----------|
| 1     | 5       | 2        |
| 4     | 5       | NA       |

`read_csv2("file2.csv")` Read semicolon delimited files with comma decimal marks.

`write_file("A;B;C\n1,5;2;3\n4,5;5;NA", file = "file2.csv")`

| A B C | 1 2 3 | 4 5 NA |
|-------|-------|--------|
| 1     | 2     | 3      |
| 4     | 5     | NA     |

`read_tsv("file.tsv")` Read a tab delimited file. Also `read_table()`.  
`read_fwf("file.tsv", fwf_widths(c(2, 2, NA)))` Read a fixed width file.

`write_file("A\tB\tC\n1\t2\t3\n4\t5\tNA", file = "file.tsv")`

### USEFUL READ ARGUMENTS

| A | B | C  |
|---|---|----|
| 1 | 2 | 3  |
| 4 | 5 | NA |

#### No header

`read_csv("file.csv", col_names = FALSE)`

| 1 | 2 | 3  |
|---|---|----|
| 4 | 5 | NA |

#### Skip lines

`read_csv("file.csv", skip = 1)`

| A | B | C |
|---|---|---|
| 1 | 2 | 3 |

#### Read a subset of lines

`read_csv("file.csv", n_max = 1)`

| A  | B | C  |
|----|---|----|
| NA | 2 | 3  |
| 4  | 5 | NA |

#### Read values as missing

`read_csv("file.csv", na = c("1"))`

| A;B;C | 1,5;2;3,0 |
|-------|-----------|
| 1     | 5         |
| 4     | 5         |
| NA    | 0         |

#### Specify decimal marks

`read_delim("file2.csv", locale = locale(decimal_mark = ","))`

## Save Data with readr

```
write_*(x, file, na = "NA", append, col_names, quote, escape, eol, num_threads, progress)
```

| A | B | C  |
|---|---|----|
| 1 | 2 | 3  |
| 4 | 5 | NA |

| A,B,C | 1,2,3 | 4,5,NA |
|-------|-------|--------|
| 1     | 2     | 3      |
| 4     | 5     | NA     |

`write_delim(x, file, delim = " ")` Write files with any delimiter.

`write_csv(x, file)` Write a comma delimited file.

`write_csv2(x, file)` Write a semicolon delimited file.

`write_tsv(x, file)` Write a tab delimited file.

One of the first steps of a project is to import outside data into R. Data is often stored in tabular formats, like csv files or spreadsheets.



The front page of this sheet shows how to import and save text files into R using `readr`.



The back page shows how to import spreadsheet data from Excel files using `readxl` or Google Sheets using `googlesheets4`.

### OTHER TYPES OF DATA

Try one of the following packages to import other types of files:

- `haven` - SPSS, Stata, and SAS files
- `DBI` - databases
- `jsonlite` - json
- `xml2` - XML
- `httr` - Web APIs
- `rvest` - HTML (Web Scraping)
- `readr::read_lines()` - text data

## Column Specification with readr

Column specifications define what data type each

column of a file will be imported as. By default `readr` will generate a column spec when a file is read and output a summary.

`spec(x)` Extract the full column specification for the given imported data frame.

```
spec(x)
# cols(
#   age = col_integer(),
#   sex = col_character(),
#   earn = col_double()
# )
```

age is an integer

sex is a character

### COLUMN TYPES

Each column type has a function and corresponding string abbreviation.

- `col_logical()` - "l"
- `col_integer()` - "i"
- `col_double()` - "d"
- `col_number()` - "n"
- `col_character()` - "c"
- `col_factor(levels, ordered = FALSE)` - "f"
- `col_datetime(format = "")` - "T"
- `col_date(format = "")` - "D"
- `col_time(format = "")` - "t"
- `col_skip()` - "-" , "\_"
- `col_guess()` - "?"

### DEFINE COLUMN SPECIFICATION

#### Set a default type

```
read_csv(
  file,
  col_type = list(.default = col_double())
)
```

#### Use column type or string abbreviation

```
read_csv(
  file,
  col_type = list(x = col_double(), y = "l", z = "_")
)
```

#### Use a single string of abbreviations

```
# col types: skip, guess, integer, logical, character
read_csv(
  file,
  col_type = "_?ilc"
)
```

# Import Spreadsheets with readxl

## READ EXCEL FILES

|   | A  | B  | C  | D  | E  |
|---|----|----|----|----|----|
| 1 | x1 | x2 | x3 | x4 | x5 |
| 2 | x  |    | z  | 8  | NA |
| 3 | y  | 7  |    | 9  | 10 |
|   | s1 |    |    |    |    |

```
read_excel(path, sheet = NULL, range = NULL)
Read a .xls or .xlsx file based on the file extension.
See front page for more read arguments. Also
read_xls() and read_xlsx().
read_excel("excel_file.xlsx")
```

## READ SHEETS

| A | B  | C  | D  | E |
|---|----|----|----|---|
|   |    |    |    |   |
|   |    |    |    |   |
|   | s1 | s2 | s3 |   |

|    |    |    |
|----|----|----|
| s1 | s2 | s3 |
|----|----|----|

| A  | B  | C  | D | E |
|----|----|----|---|---|
|    |    |    |   |   |
|    |    |    |   |   |
|    |    |    |   |   |
| s  |    |    |   |   |
| s1 | s2 | s3 |   |   |

```
path <- "your_file_path.xlsx"
path %>% excel_sheets() %>%
  set_names() %>%
  map_dfr(read_excel, path = path)
```

## OTHER USEFUL EXCEL PACKAGES

For functions to write data to Excel files, see:

- **openxlsx**
- **writexl**

For working with non-tabular Excel data, see:

- **tidyxl**



# with googlesheets4

## READ SHEETS

|   | A  | B  | C  | D  | E  |
|---|----|----|----|----|----|
| 1 | x1 | x2 | x3 | x4 | x5 |
| 2 | x  |    | z  | 8  | NA |
| 3 | y  | 7  |    | 9  | 10 |
|   | s1 |    |    |    |    |

|  | x1 | x2 | x3 | x4 | x5 |
|--|----|----|----|----|----|
|  | x  | NA | z  | 8  | NA |
|  | y  | 7  | NA | 9  | 10 |
|  | s1 |    |    |    |    |

### READXL COLUMN SPECIFICATION

Column specifications define what data type each column of a file will be imported as.

Use the **col\_types** argument of **read\_excel()** to set the column specification.

#### Guess column types

To guess a column type, **read\_excel()** looks at the first 1000 rows of data. Increase with the **guess\_max** argument.

```
read_excel(path, guess_max = Inf)
```

#### Set all columns to same type, e.g. character

```
read_excel(path, col_types = "text")
```

#### Set each column individually

```
read_excel(
  path,
  col_types = c("text", "guess", "guess", "numeric"))
```

## COLUMN TYPES

| logical | numeric | text  | date       | list  |
|---------|---------|-------|------------|-------|
| TRUE    | 2       | hello | 1947-01-08 | hello |
| FALSE   | 3.45    | world | 1956-10-21 | 1     |

- skip
- logical
- date
- guess
- numeric
- list
- text

Use **list** for columns that include multiple data types. See **tidy** and **purrr** for list-column data.

Use the **range** argument of **readxl::read\_excel()** or **googlesheets4::read\_sheet()** to read a subset of cells from a sheet.

```
read_excel(path, range = "Sheet1!B1:D2")
read_sheet(ss, range = "B1:D2")
```

Also use the range argument with cell specification functions **cell\_limits()**, **cell\_rows()**, **cell\_cols()**, and **anchored()**.



### GOOGLESHEETS4 COLUMN SPECIFICATION

Column specifications define what data type each column of a file will be imported as.

```
read_sheet(ss, sheet = NULL, range = NULL)
Read a sheet from a URL, a Sheet ID, or a dribble
from the googledrive package. See front page for
more read arguments. Same as range_read().
```

## Commands

|                                 |
|---------------------------------|
| File name = <b>read.delim()</b> |
| Attach(File name)               |
| File name = <b>read_excel()</b> |
| File name = <b>read_csv()</b>   |
| Install.packages()              |
| Plot.ts()                       |
| acf()                           |
| pacf()                          |

## COLUMN TYPES

| I     | n    | c     | D          | L     |
|-------|------|-------|------------|-------|
| TRUE  | 2    | hello | 1947-01-08 | hello |
| FALSE | 3.45 | world | 1956-10-21 | 1     |

- skip - "\_" or "-"
- guess - "?"
- logical - "I"
- integer - "i"
- double - "d"
- numeric - "n"
- date - "D"
- datetime - "T"
- character - "c"
- list-column - "L"
- cell - "C" Returns list of raw cell data.

Use list for columns that include multiple data types. See **tidy** and **purrr** for list-column data.

## FILE LEVEL OPERATIONS

**googlesheets4** also offers ways to modify other aspects of Sheets (e.g. freeze rows, set column width, manage (work)sheets). Go to [googlesheets4.tidyverse.org](https://googlesheets4.tidyverse.org) to read more.

For whole-file operations (e.g. renaming, sharing, placing within a folder), see the tidyverse package **googledrive** at [googledrive.tidyverse.org](https://googledrive.tidyverse.org).

# rmarkdown :: CHEAT SHEET

## What is rmarkdown?



**.Rmd files** • Develop your code and ideas side-by-side in a single document. Run code as individual chunks or as an entire document.

**Dynamic Documents** • Knit together plots, tables, and results with narrative text. Render to a variety of formats like HTML, PDF, MS Word, or MS PowerPoint.

**Reproducible Research** • Upload, link to, or attach your report to share. Anyone can read or run your code to reproduce your work.

## Workflow

- 1 Open a **new .Rmd file** in the RStudio IDE by going to *File > New File > R Markdown*.
- 2 **Embed code** in chunks. Run code by line, by chunk, or all at once.
- 3 **Write text** and add tables, figures, images, and citations. Format with Markdown syntax or the RStudio Visual Markdown Editor.
- 4 **Set output format(s) and options** in the YAML header. Customize themes or add parameters to execute or add interactivity with Shiny.
- 5 **Save and render** the whole document. Knit periodically to preview your work as you write.
- 6 **Share your work!**

## Embed Code with knitr

### CODE CHUNKS

Surround code chunks with `{{r}}` and `{{` or use the Insert Code Chunk button. Add a chunk label and/or chunk options inside the curly braces after r.

```
```{r chunk-label, include=FALSE}
summary(mtcars)
...```

```

### SET GLOBAL OPTIONS

Set options for the entire document in the first chunk.

```
```{r include=FALSE}
knitr::opts_chunk$message = FALSE
...```

```

### INLINE CODE

Insert `r <code>` into text sections. Code is evaluated at render and results appear as text.

"Built with `r getRversion()`" --> "Built with 4.1.0"



### SOURCE EDITOR

The Source Editor shows an R Markdown file (report.Rmd) with the following content:

```
---
title: "Document Title"
author: "Author Name"
output:
  html_document:
    toc: TRUE
---
```

Code chunks are embedded in the text. The editor interface includes a toolbar with icons for file operations, code execution, and sharing.

### VISUAL EDITOR

The Visual Editor shows an R Markdown file (report.Rmd) with the following content:

```
This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents.
```

The editor interface includes a toolbar with icons for file operations, code execution, and styling.

### RENDERED OUTPUT

The Rendered Output pane shows the generated HTML document (report.html) with the following content:

file path to output document  
report.html Open in Browser Find Publish

**Document Title**  
Author Name  
• R Markdown  
• Including Plots

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

```
summary(cars)
```

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

802.9 KB Jul 9, 2021, 4:35 PM

## Write with Markdown

The syntax on the left renders as the output on the right.

Plain text.	Plain text.
lmtest(arma)	Upload, link to, or attach your report to share. Anyone can read or run your code.
forecast(pred)	readxl
readxl	vars(ur.df)
vars(ur.df)	urca
urca	aTSA(pp)
aTSA(pp)	Hmisc(Engle)
Hmisc(Engle)	rugarch
rugarch	stochvol
stochvol	

<b>Header 1</b>	# Header 1
<b>Header 2</b>	## Header 2
...	...
<b>Header 6</b>	##### Header 6
- unordered list	- unordered list
- item 2	- item 2
- item 2a (indent 1 tab)	- item 2a (indent 1 tab)
- item 2b	- item 2b
1: ordered list	1: ordered list
2: item 2	- item 2a (indent 1 tab)
- item 2b	- item 2b

<link url>  
[This is a link.] (link url)  
[This is another link][id].

At the end of the document:  
[id]: link url

![Caption](image.png)  
or ! [Caption][id2]

At the end of the document:  
[id2]: image.png

'verbatim code'  
'''

multiple lines  
of verbatim code

> block quotes

equation: \$e^{i\pi} + 1 = 0\$

equation block:  
\$E = mc^2\$

horizontal rule:  
---

Right   Left   Default   Center
----- :----- ----- :-----
12   12   12   12
123   123   123   123

horizontal rule:  
---

Right	Left	Default	Center
12	12	12	12
123	123	123	123
1	1	1	1

HTML Tabsets  
# Results {.tabset}  
## Plots text  
text  
## Tables  
more text

Results
Plots

OPTION	DEFAULT EFFECTS
echo	TRUE variable = ts(variable, start = c(year,quarter), frequency =)TRUE
error	FALSE variable = window(variable, end = c(year,quarter))
eval	TRUE
include	TRUE " nvariable = diff(log(variable)))
message	TRUE ar/ma = arima(nvariable, order = c(0,0,0), fixed = c(NA,0))
warning	TRUE coefest(ar/ma)
results	ar/ma\$aic <
fig.align	NULL n = predict(ar/ma, n.ahead = x)
fig.alt	NULL n\$pred
fig.cap	"figure/" 7 plot.ts(nvariable)
fig.path	lines(n\$pred, col="red")
fig.width & fig.height	FALSE "#"
out.width	NULL
collapse	TRUE extracting source code with knitr::purl()
comment	See more options and defaults by running str(knitr::opts_chunk\$get())

RStudio® is a trademark of RStudio, PBC • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at [rmarkdown.rstudio.com](http://rmarkdown.rstudio.com) • rmarkdown 2.9.4 • Updated: 2021-08



# Set Output Formats and their Options in YAML

Use the document's YAML header to set an **output format** and customize it with **output options**.

```
---
```

```
title: "My Document"
author: "Author Name"
output:
  html_document: toc: TRUE
---
```

Indent format 2 characters,  
indent options 4 characters

OUTPUT FORMAT	CREATES
html_document	.html
pdf_document*	.pdf
word_document	Microsoft Word (.docx)
powerpoint_presentation	Microsoft Powerpoint (.pptx)
odt_document	OpenDocument Text
rtf_document	Rich Text Format
md_document	Markdown
github_document	Markdown for Github
ioslides_presentation	ioslides HTML slides
slidy_presentation	Slidy HTML slides
beamer_presentation*	Beamer slides

\* Requires LaTeX, use `tinytex::install_tinytex()`  
Also see `flexdashboard`, `bookdown`, `distill`, and `blogdown`.

	DESCRIPTION	HTML	PDF	MS Word	MS PPT
<b>anchor_sections</b>	xvariable = ur.df(variable, type = „, selectlags = „AIC“)	X			
<b>citation_package</b>	summary(xvariable) (tau<t nonS) (phi<t trend)		X		
<b>code_download</b>	pp.test(variable) (<lag noT/noD)		X		
<b>code_folding</b>	xvariable=ts(variable, start = c(year,quarter), frequency = )		X		
<b>css</b>	xvariable = window(xvariable, end = c(year,quarter))X				
<b>dev</b>	acf(xvariable) choose lag		X X		
<b>df_print</b>	ar/ma, coeftest(ar/ma), pacf(residuals(arX)), ar/ma\$aic		X X X X		
<b>fig_caption</b>	dvariable = predict(ar/ma, n.head = x)\$pred		X X X X		
<b>highlight</b>	cvariable = tail(xvariable, x)		X X X		
<b>includes</b>	rmvariable = sqrt(sum((dvariable-cvariable)^2)/length(cvariable))		X X		
<b>keep_md</b>	rmvariable (RMSE<G) automatic		X X X X		
<b>keep_tex</b>	accuracy(dvariable, cvariable)		X		
<b>latex_engine</b>	max.pred = predict(ar/ma, n.head=x)\$pred+ same()\$se*1.96		X		
<b>reference_docx/_doc</b>	min.pred = predict(ar/ma, n.head=x)\$pred- same()\$se*1.96		X X		
<b>theme</b>	plot.ts(xvariable), lines(max.pred, col="red") lines...		X		
<b>toc</b>	forecast(ar/ma, h=x)		X X X X		
<b>toc_depth</b>	The lowest level of headings to add to table of contents (e.g. 2, 3)		X X X X		
<b>toc_float</b>	Float the table of contents to the left of the main document content (TRUE or FALSE)	X			

Use `?<output format>` to see all of a format's options, e.g. `?html_document`

## More Header Options

### PARAMETERS

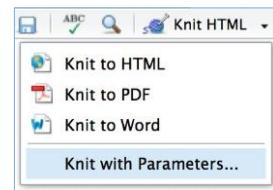
Parameterize your documents to reuse with new inputs (e.g., data, values, etc.).

1. **Add parameters** in the header as sub-values of params.

```
---
```

```
params:
  state: "hawaii"
---
```

```
```{r}
data <- df[, params$state]
summary(data)
```
```



### BOOTSWATCH THEMES

Customize HTML documents with Bootswatch themes from the **bslib** package using the theme output option.

Use `bslib::bootswatch_themes()` to list available themes.



```
---
```

```
title: "Document Title"
author: "Author Name"
output:
  html_document:
    theme: bootstrap: solar
---
```

### CUSTOM THEMES

Customize individual HTML elements using bslib variables. Use `?bs_theme` to see more variables.

```
---
```

```
output:
  html_document:
    theme:
      bg: "#121212"
      fg: "#E4E4E4"
      base_font:
        google: "Prompt"
---
```

More on **bslib** at [pkgs.rstudio.com/bslib/](https://pkgs.rstudio.com/bslib/).

### STYLING WITH CSS AND SCSS

Add CSS and SCSS to your document by adding a path to a file with the **css** option in the YAML header.

```
---
```

```
title: "My Document"
author: "Author Name"
output:
  html_document:
    css: "style.css"
---
```

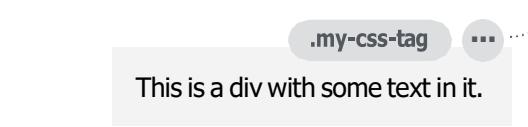
Apply CSS styling by writing HTML tags directly or:

• Use markdown to apply style attributes inline:

Bracketed Span  
A [green]{.my-color} word.

Fenced Div  
:::{.my-color}  
All of these words  
are green.  
...

- Use the Visual Editor. Go to **Format > Div/Span** and add CSS styling directly with Edit Attributes.



## Render

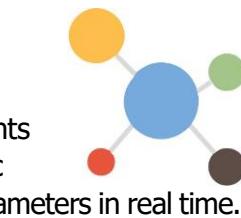
When you render a document, rmarkdown:

1. Runs the code and embeds results and text into an .md file with knitr.
2. Converts the .md file into the output format with Pandoc.



**Save**, then **Knit** to preview the document output. The resulting HTML/PDF/MS Word/etc. document will be created and saved in the same directory as the .Rmd file.

Use `rmarkdown::render()` to render/knit in the R console. See `?render` for available options.



## Share

### Publish on RStudio Connect

to share R Markdown documents securely, schedule automatic updates, and interact with parameters in real time.

[rstudio.com/products/connect/](https://rstudio.com/products/connect/)



### INTERACTIVITY

Turn your report into an interactive Shiny document in 4 steps:

1. Add **runtime: shiny** to the YAML header.
2. Call Shiny input functions to embed input objects.
3. Call Shiny render functions to embed reactive output.
4. Render with `rmarkdown::run()` or click **Run Document** in RStudio IDE.

```
output: html_document
runtime: shiny
---
```

```
```{r, echo = FALSE}
numericInput("n",
  "How many cars?", 5)

renderTable({
  head(cars, input$n)
})```

```

How many cars?	
5	
speed	dist
1	4.00
2	10.00
3	4.00
4	22.00
5	16.00

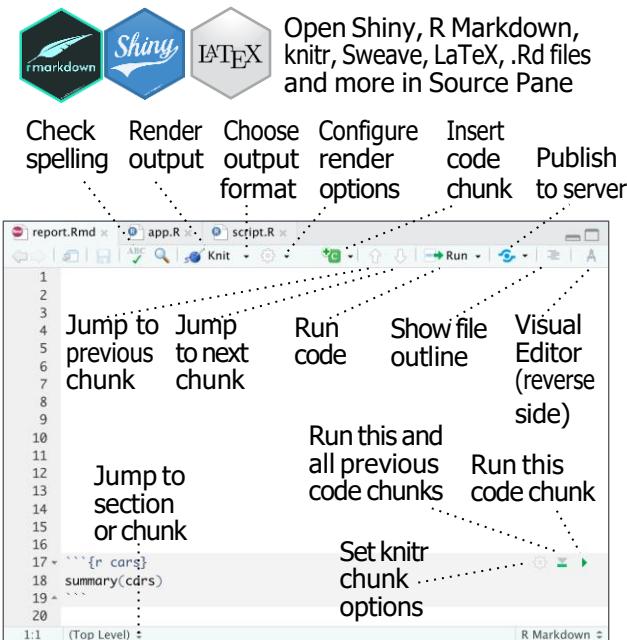
Also see Shiny Prerendered for better performance.

[rstudio.com/rmarkdown/shiny-prerendered/](https://rstudio.com/rmarkdown/shiny-prerendered/)

Embed a complete app into your document with `shiny::shinyAppDir()`. More at [bookdown.org/yihui/rmarkdown/shiny-embedded.html](https://bookdown.org/yihui/rmarkdown/shiny-embedded.html).

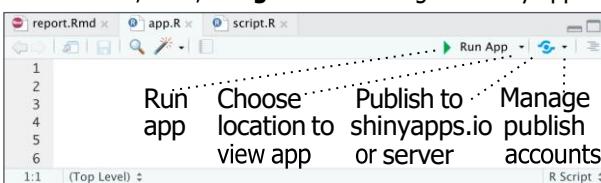
# RStudio IDE : : CHEAT SHEET

## Documents and Apps



Access markdown guide at [Help > Markdown Quick Reference](#)  
See reverse side for more on [Visual Editor](#)

RStudio recognizes that files named **app.R**, **server.R**, **ui.R**, and **global.R** belong to a shiny app

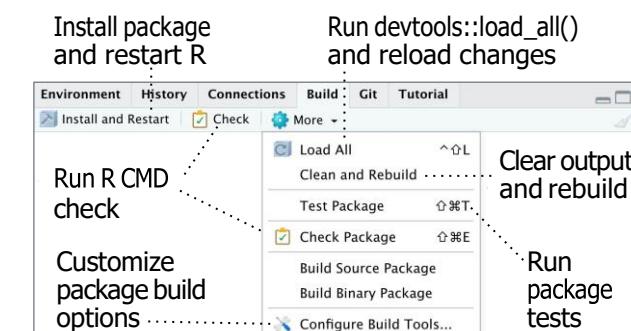


## Package Development

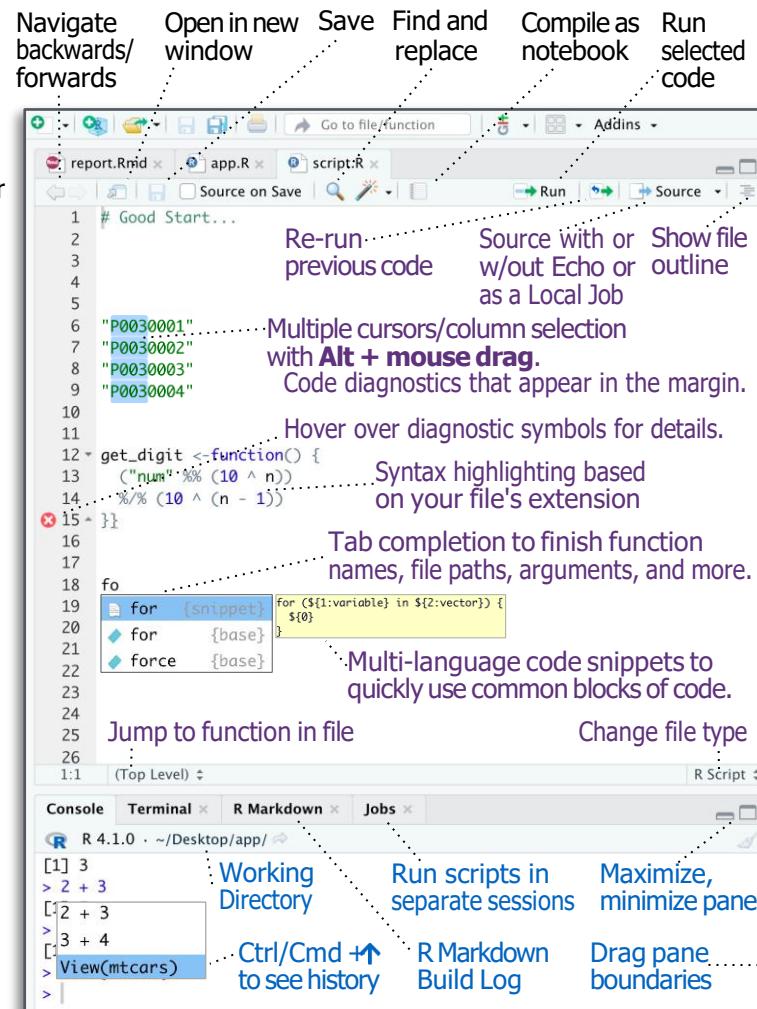
Create a new package with [File > New Project > New Directory > R Package](#)  
Enable roxygen documentation with [Tools > Project Options > Build Tools](#)

Roxygen guide at [Help > Roxygen Quick Reference](#)

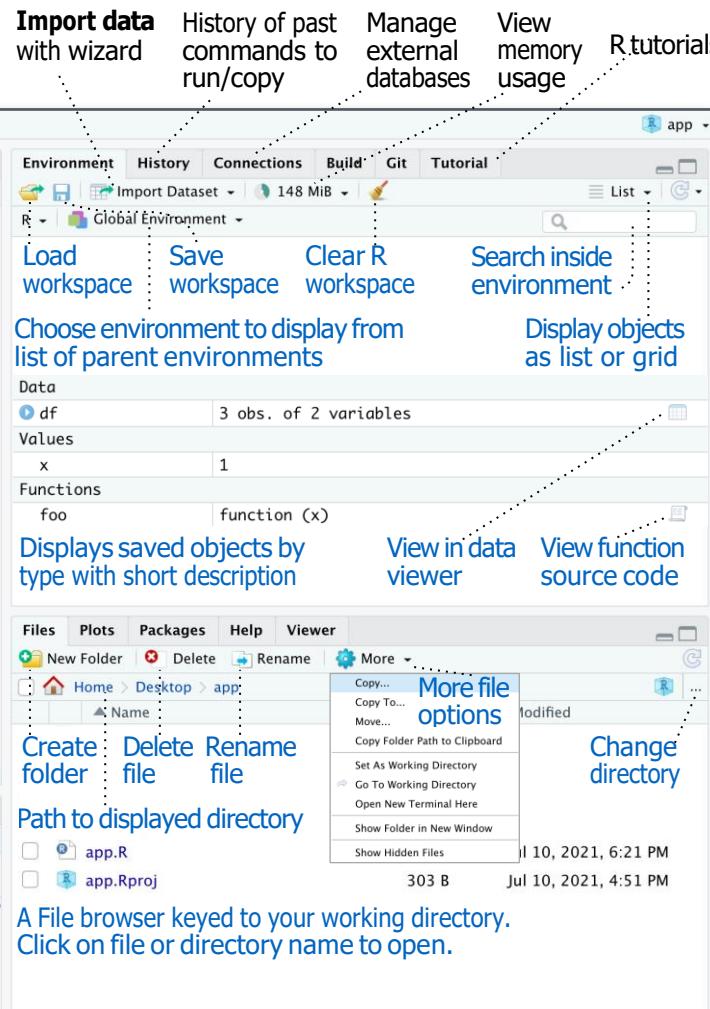
See package information in the **Build Tab**



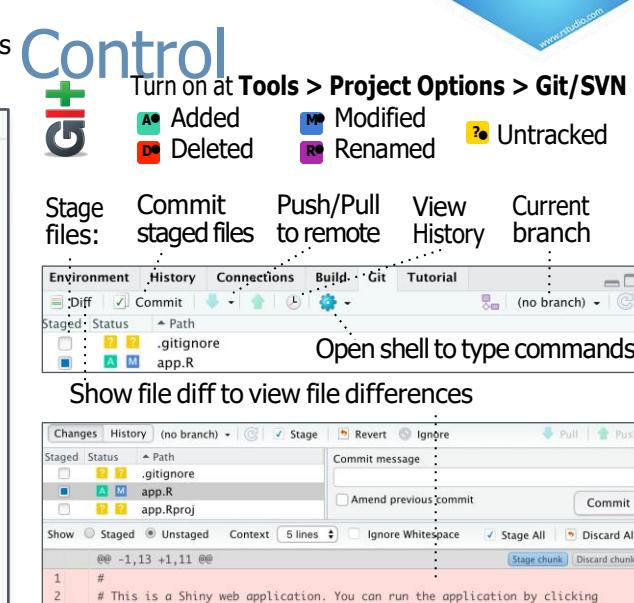
## Source Editor



## Tab Panes



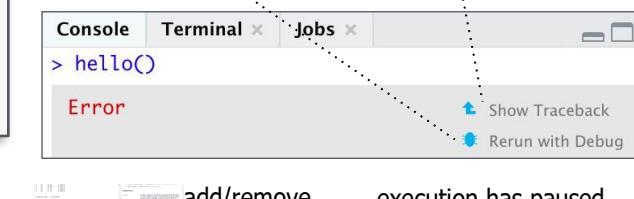
## Version Control



## Debug Mode

Use **debug()**, **browser()**, or a breakpoint and execute your code to open the debugger mode.

Open traceback to examine the functions that R called before the error occurred

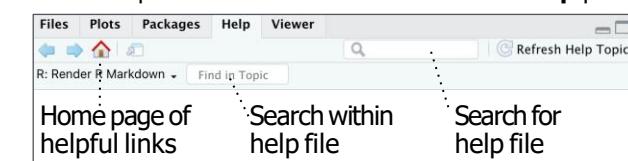


execution has paused a breakpoint.

RStudio opens plots in a dedicated **Plots** pane



RStudio opens documentation in a dedicated **Help** pane



**Viewer** pane displays HTML content, such as Shiny apps, RMarkdown reports, and interactive visualizations



**View(<data>)** opens spreadsheet like view of data set

Filter	mpg	cyl	disp	hp	drat	wt	qsec	vs	am
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1

Filter rows by value or value range

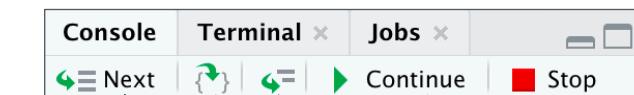
Sort by values

Search for value

Run commands in environment where execution has paused

Examine variables in executing environment

Select function in traceback to debug



Step through code one line at time a Step into and out of functions to run





# Keyboard Shortcuts

## RUN CODE

Search command history  
Interrupt current command  
Clear console

## Windows/Linux Mac

Ctrl+↑	Cmd+↑
Esc	Esc
Ctrl+L	Ctrl+L

## NAVIGATE CODE

Go to File/Function

## WRITE CODE

Attempt completion

Insert <- (assignment operator)  
Insert %>% (pipe operator)  
(Un)Comment selection

## Windows/Linux Mac

Alt+-	Option+-
Ctrl+Shift+M	Cmd+Shift+M
Ctrl+Shift+C	Cmd+Shift+C

## Windows/Linux Mac

Ctrl+Shift+L	Cmd+Shift+L
Ctrl+Shift+T	Cmd+Shift+T
Ctrl+Shift+D	Cmd+Shift+D

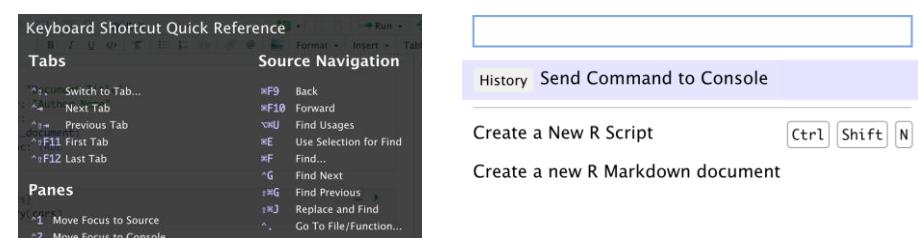
## DOCUMENTS AND APPS

Knit Document (knitr)	Ctrl+Shift+K	Cmd+Shift+K
Insert chunk (Sweave & Knitr)	Ctrl+Alt+I	Cmd+Option+I
Run from start to current line	Ctrl+Alt+B	Cmd+Option+B

## MORE KEYBOARD SHORTCUTS

Keyboard Shortcuts Help	Alt+Shift+K	Option+Shift+K
Show Command Palette	Ctrl+Shift+P	Cmd+Shift+P

View the Keyboard Shortcut Quick Reference with **Tools > Keyboard Shortcuts** or Alt/Option + Shift + K



# RStudio Workbench

## WHY RSTUDIO WORKBENCH?

Extend the open source server with a commercial license, support, and more:

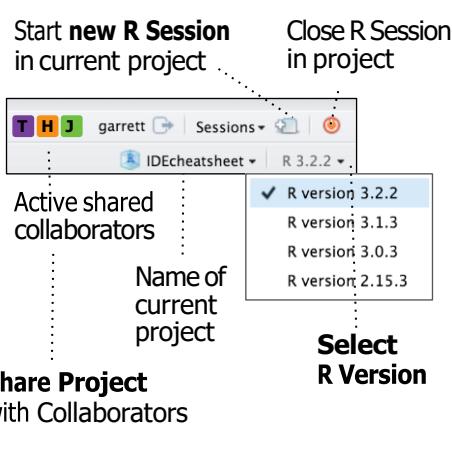
- open and run multiple R sessions at once
- tune your resources to improve performance
- administrative tools for managing user sessions
- collaborate real-time with others in shared projects
- switch easily from one version of R to a different version
- integrate with your authentication, authorization, and audit practices
- work in the RStudio IDE, JupyterLab, Jupyter Notebooks, or VS Code

Download a free 45 day evaluation at [www.rstudio.com/products/workbench/evaluation/](http://www.rstudio.com/products/workbench/evaluation/)

# Share Projects

## File > New Project

RStudio saves the call history, workspace, and working directory associated with a project. It reloads each when you re-open a project.



# Visual Editor

The screenshot shows the RStudio Visual Editor interface with several keyboard shortcut overlays. The shortcuts are:

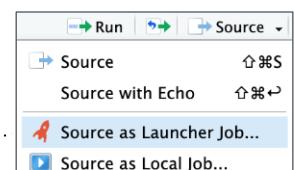
- Check spelling
- Render output
- Choose output format
- Choose output location
- Insert code chunk
- Jump to previous chunk
- Jump to next chunk
- Run selected lines
- Publish to server
- Show file outline
- Back to Source Editor (front page)
- File outline
- Add/Edit attributes
- Set knitr chunk options
- Run this and all previous code chunks
- Run this code chunk

Below the editor, there's a preview pane showing R Markdown code and a preview of the output. A status bar at the bottom indicates the current mode is 'R Markdown'.



# Run Remote Jobs

Run R on remote clusters (Kubernetes/Slurm) via the Job Launcher



The screenshot shows the RStudio Job Launcher interface with a list of jobs:

Console	Terminal	Jobs	Launcher
fast.R	Running	Local	0:09
sleepy.R	Succeeded 11:22 AM	Local	0:41
sleepy.R	Idle	KubernetesX	Waiting

Buttons for 'Monitor launcher jobs' and 'Launch a job' are visible.

Run launcher jobs remotely

# Data transformation with dplyr : : CHEAT SHEET



dplyr functions work with pipes and expect **tidy data**. In tidy data:



Each **variable** is in its own **column**

Each **observation**, or **case**, is in its own **row**



$x \%>% f(y)$  becomes  $f(x, y)$

## Summarise Cases

Apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).



 **summarise(.data, ...)**  
Compute table of summaries.  
`summarise(mtcars, avg = mean(mpg))`

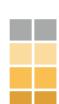
 **count(.data, ..., wt = NULL, sort = FALSE, name = NULL)** Count number of rows in each group defined by the variables in ... Also **tally()**.  
`count(mtcars, cyl)`

## Group Cases

Use **group\_by(.data, ..., .add = FALSE, .drop = TRUE)** to create a "grouped" copy of a table grouped by columns in ... dplyr functions will manipulate each "group" separately and combine the results.

 **mtcars %>% group\_by(cyl) %>% summarise(avg = mean(mpg))**

Use **rowwise(.data, ...)** to group data into individual rows. dplyr functions will compute results for each row. Also apply functions to list-columns. See tidyverse cheat sheet for list-column workflow.

 **starwars %>% rowwise() %>% mutate(film\_count = length(films))**

**ungroup(x, ...)** Returns ungrouped copy of table.  
`ungroup(g_mtcars)`

## Manipulate Cases

### EXTRACT CASES

Row functions return a subset of rows as a new table.



**filter(.data, ..., .preserve = FALSE)** Extract rows that meet logical criteria.  
`filter(mtcars, mpg > 20)`



**distinct(.data, ..., .keep\_all = FALSE)** Remove rows with duplicate values.  
`distinct(mtcars, gear)`



**slice(.data, ..., .preserve = FALSE)** Select rows by position.  
`slice(mtcars, 10:15)`



**slice\_sample(.data, ..., n, prop, weight\_by = NULL, replace = FALSE)** Randomly select rows. Use n to select a number of rows and prop to select a fraction of rows.  
`slice_sample(mtcars, n = 5, replace = TRUE)`



**slice\_min(.data, order\_by, ..., n, prop, with\_ties = TRUE)** and **slice\_max()** Select rows with the lowest and highest values.  
`slice_min(mtcars, mpg, prop = 0.25)`



**slice\_head(.data, ..., n, prop)** and **slice\_tail()** Select the first or last rows.  
`slice_head(mtcars, n = 5)`

### Logical and boolean operators to use with filter()

<code>==</code>	<code>&lt;</code>	<code>&lt;=</code>	<code>is.na()</code>	<code>%in%</code>	<code> </code>	<code>xor()</code>
<code>!=</code>	<code>&gt;</code>	<code>&gt;=</code>	<code>!is.na()</code>	<code>!</code>	<code>&amp;</code>	

See [?base::Logic](#) and [?Comparison](#) for help.

### ARRANGE CASES



**arrange(.data, ..., .by\_group = FALSE)** Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.  
`arrange(mtcars, mpg)`  
`arrange(mtcars, desc(mpg))`



**add\_row(.data, ..., .before = NULL, .after = NULL)** Add one or more rows to a table.  
`add_row(cars, speed = 1, dist = 1)`

## Manipulate Variables

### EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.



**pull(.data, var = -1, name = NULL, ...)** Extract column values as a vector, by name or index.  
`pull(mtcars, wt)`



**select(.data, ...)** Extract columns as a table.  
`select(mtcars, mpg, wt)`



**relocate(.data, ..., .before = NULL, .after = NULL)** Move columns to new position.  
`relocate(mtcars, mpg, cyl, .after = last_col())`

### Use these helpers with select() and across()

e.g. `select(mtcars, mpg:cyl)`

**contains(match)** **num\_range(prefix, range)** ;, e.g. `mpg:cyl`  
**ends\_with(match)** **all\_of(x)/any\_of(x, ..., vars)** -, e.g., `-gear`  
**starts\_with(match)** **matches(match)** **everything()**

### MANIPULATE MULTIPLE VARIABLES AT ONCE



**across(.cols, .funs, ..., .names = NULL)** Summarise or mutate multiple columns in the same way.  
`summarise(mtcars, across(everything(), mean))`



**c\_across(.cols)** Compute across columns in row-wise data.  
`transmute(rowwise(UKgas), total = sum(c_across(1:2)))`

### MAKE NEW VARIABLES

Apply **vectorized functions** to columns. Vectorized functions take vectors as input and return vectors of the same length as output (see back).



**mutate(.data, ..., .keep = "all", .before = NULL, .after = NULL)** Compute new column(s). Also **add\_column()**, **add\_count()**, and **add\_tally()**.  
`mutate(mtcars, gpm = 1 / mpg)`



**transmute(.data, ...)** Compute new column(s), drop others.  
`transmute(mtcars, gpm = 1 / mpg)`



**rename(.data, ...)** Rename columns. Use **rename\_with()** to rename with a function.  
`rename(cars, distance = dist)`



# Vectorized Functions

## TO USE WITH MUTATE ()

**mutate()** and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

### vectorized function

## OFFSET

dplyr::lag() - offset elements by 1  
dplyr::lead() - offset elements by -1

## CUMULATIVE AGGREGATE

dplyr::cumall() - cumulative all()  
dplyr::cumany() - cumulative any()  
dplyr::cummax() - cumulative max()  
dplyr::cummean() - cumulative mean()  
dplyr::cummin() - cumulative min()  
dplyr::cumprod() - cumulative prod()  
dplyr::cumsum() - cumulative sum()

## RANKING

dplyr::cume\_dist() - proportion of all values <= dplyr::dense\_rank() - rank w ties = min, no gaps dplyr::min\_rank() - rank with ties = min dplyr::ntile() - bins into n bins dplyr::percent\_rank() - min\_rank scaled to [0,1] dplyr::row\_number() - rank with ties = "first"

## MATH

+, -, \*, /, ^, %/%, %% - arithmetic ops  
log(), log2(), log10() - logs  
<, <=, >, >=, !=, == - logical comparisons  
dplyr::between() - x >= left & x <= right  
dplyr::near() - safe == for floating point numbers

## MISCELLANEOUS

dplyr::case\_when() - multi-case if\_else()  
starwars %>%  
mutate(type = case\_when(  
height > 200 | mass > 200 ~ "large",  
species == "Droid" ~ "robot",  
TRUE ~ "other"))  
  
dplyr::coalesce() - first non-NA values by element across a set of vectors  
dplyr::if\_else() - element-wise if() + else()  
dplyr::na\_if() - replace specific values with NA  
pmax() - element-wise max()  
pmin() - element-wise min()

# Summary Functions

## TO USE WITH SUMMARISE ()

**summarise()** applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

### summary function

## COUNT

dplyr::n() - number of values/rows  
dplyr::n\_distinct() - # of uniques  
sum(is.na()) - # of non-NA's

## POSITION

mean() - mean, also mean(is.na())  
median() - median

## LOGICAL

mean() - proportion of TRUE's  
sum() - # of TRUE's

## ORDER

dplyr::first() - first value  
dplyr::last() - last value  
dplyr::nth() - value in nth location of vector

## RANK

quantile() - nth quantile  
min() - minimum value  
max() - maximum value

## SPREAD

IQR() - Inter-Quartile Range  
mad() - median absolute deviation  
sd() - standard deviation  
var() - variance

# Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

**tibble::rownames\_to\_column()**  
Move row names into col.  
a <- rownames\_to\_column(mtcars, var = "C")

**tibble::column\_to\_rownames()**  
Move col into row names.  
column\_to\_rownames(a, var = "C")

Also tibble::has\_rownames() and tibble::remove\_rownames().

# Combine Tables

## COMBINE VARIABLES

X	Y	=
A B C a t 1 b u 2 c v 3	E F G a t 3 b u 2 d w 1	A B C E F G a t 1 a t 3 b u 2 b u 2 c v 3 d w 1

**bind\_cols(..., .name\_repair)** Returns tables placed side by side as a single table. Column lengths must be equal. Columns will NOT be matched by id (to do that look at Relational Data below), so be sure to check that both tables are ordered the way you want before binding.

## RELATIONAL DATA

Use a "Mutating Join" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

X	Y	=
A B C D a t 1 3 b u 2 2 c v 3 NA		

**left\_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ..., keep = FALSE, na\_matches = "na")** Join matching values from y to x.

X	Y	=
A B C D a t 1 3 b u 2 2 d w NA 1		

**right\_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ..., keep = FALSE, na\_matches = "na")** Join matching values from x to y.

X	Y	=
A B C D a t 1 3 b u 2 2		

**inner\_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ..., keep = FALSE, na\_matches = "na")** Join data. Retain only rows with matches.

X	Y	=
A B C D a t 1 3 b u 2 2 c v 3 NA		

**full\_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ..., keep = FALSE, na\_matches = "na")** Join data. Retain all values, all rows.

## COMBINE CASES

X	Y	=
A B C a t 1 b u 2	A B C c v 3 d w 4	

**bind\_rows(..., .id = NULL)** Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured).

Use a "Filtering Join" to filter one table against the rows of another.

X	Y	=
A B C a t 1 b u 2 c v 3	A B D a t 3 b u 2 d w 1	

**semi\_join(x, y, by = NULL, copy = FALSE, ..., na\_matches = "na")** Return rows of x that have a match in y. Use to see what will be included in a join.

**anti\_join(x, y, by = NULL, copy = FALSE, ..., na\_matches = "na")** Return rows of x that do not have a match in y. Use to see what will not be included in a join.

Use a "Nest Join" to inner join one table to another into a nested data frame.

X	Y	=
A B C a t 1 b u 2 c v 3		

**nest\_join(x, y, by = NULL, copy = FALSE, keep = FALSE, name = NULL, ...)** Join data, nesting matches from y in a single new data frame column.

## SET OPERATIONS

**intersect(x, y, ...)**  
Rows that appear in both x and y.



**setdiff(x, y, ...)**  
Rows that appear in x but not y.



**union(x, y, ...)**  
Rows that appear in x or y.  
(Duplicates removed). **union\_all()** retains duplicates.



Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

# Data tidying with `tidyr` :: CHEAT SHEET

Tidy data is a way to organize tabular data in a

consistent data structure across packages.

A table is tidy if:



Each **variable** is in its own column

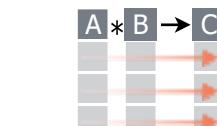
&



Each **observation**, or **case**, is in its own row



Access variables as vectors



Preserve cases in vectorized operations

## Tibbles

AN ENHANCED DATA FRAME

Tibbles are a table format provided by the `tibble` package. They inherit the data frame class, but have improved behaviors:

- Subset a new tibble with `[ ]`, a vector with `[[ ]]` and `$`.
- No partial matching when subsetting columns.
- Display concise views of the data on one screen.

`options(tibble.print_max = n, tibble.print_min = m, tibble.width = Inf)` Control default display settings.

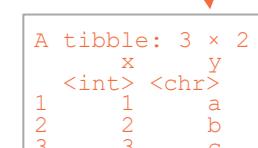
`View()` or `glimpse()` View the entire data set.

## CONSTRUCT A TIBBLE

`tibble(...)` Construct by columns.

`tibble(x = 1:3, y = c("a", "b", "c"))`

Both make this tibble



`as_tibble(x, ...)` Convert a data frame to a tibble.

`enframe(x, name = "name", value = "value")`

Convert a named vector to a tibble. Also `deframe()`.

`is_tibble(x)` Test whether x is a tibble.



## Reshape Data

- Pivot data to reorganize values into a new layout.

table4a

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K



country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

table2

country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T
C	2000	cases	213K
C	2000	pop	1T



country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	213K	1T

## Split Cells

- Use these functions to split or combine cells into individual, isolated values.

table5

country	century	year
A	19	99
A	20	00
B	19	99
B	20	00

country	year
A	1999
A	2000
B	1999
B	2000

table3

country	year	rate
A	1999	0.7K 19M
A	2000	2K 20M
B	1999	37K 172M
B	2000	80K 174M

table3

country	year	rate
A	1999	0.7K 19M
A	2000	2K 20M
B	1999	37K 172M
B	2000	80K 174M

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172
B	2000	80K	174

`pivot_longer(data, cols, names_to = "name", values_to = "value", values_drop_na = FALSE)`

"Lengthen" data by collapsing several columns into two. Column names move to a new names\_to column and values to a new values\_to column.

`pivot_longer(table4a, cols = 2:3, names_to = "year", values_to = "cases")`

`pivot_wider(data, names_from = "name", values_from = "value")`

The inverse of `pivot_longer()`. "Widen" data by expanding two columns into several. One column provides the new column names, the other the values.

`pivot_wider(table2, names_from = type, values_from = count)`

## Expand Tables

Create new combinations of variables or identify implicit missing values (combinations of variables not present in the data).

x1	x2	x3
A	1	3
B	1	4
B	2	3

x1	x2
A	1
A	2
B	1
B	2

`expand(data, ...)` Create a new tibble with all possible combinations of the values listed in ... Drop other variables.

`expand(mtcars, cyl, gear, carb)`

x1	x2	x3
A	1	3
B	1	4
B	2	3

x1	x2	x3
A	1	3
A	2	NA
B	1	4
B	2	3

`complete(data, ..., fill = list())` Add missing possible combinations of values of variables listed in ... Fill remaining variables with NA.

`complete(mtcars, cyl, gear, carb)`

x1	x2




<tbl\_r cells="2" ix="4" maxcspan="1"



# Nested Data

A nested data frame stores individual tables as a list-column of data frames within a larger organizing data frame. List-columns can also be lists of vectors or lists of varying data types.

Use a nested data frame to:

- Preserve relationships between observations and subsets of data. Preserve the type of the variables being nested (factors and datetimes aren't coerced to character).
- Manipulate many sub-tables at once with **purrr** functions like `map()`, `map2()`, or `pmap()` or with **dplyr** `rowwise()` grouping.

## CREATE NESTED DATA

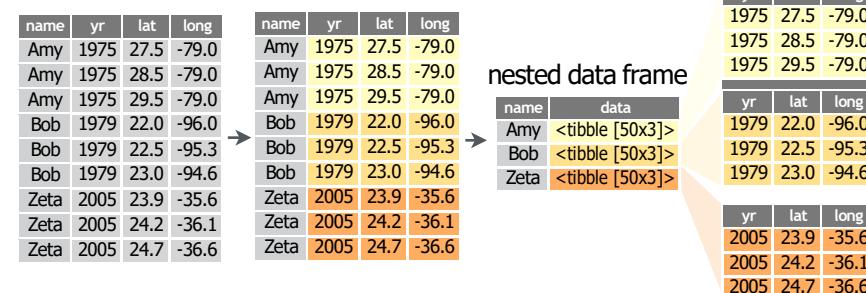
`nest(data, ...)` Moves groups of cells into a list-column of a data frame. Use alone or with `dplyr::group_by()`:

1. Group the data frame with `group_by()` and use `nest()` to move the groups into a list-column.

```
n_storms <- storms %>%
  group_by(name) %>%
  nest()
```

2. Use `nest(new_col = c(x, y))` to specify the columns to group using `dplyr::select()` syntax.

```
n_storms <- storms %>%
  nest(data = c(year:long))
```



Index list-columns with `[[[]]]`. `n_storms$data[[1]]`

## CREATE TIBBLES WITH LIST-COLUMNS

`tibble::tribble(...)` Makes list-columns when needed.

```
tribble(~max, ~seq,
       3, 1:3, max = 3, seq = <int[3]>
       4, 1:4, max = 4, seq = <int[4]>
       5, 1:5, max = 5, seq = <int[5]>)
```

`tibble::tibble(...)` Saves list input as list-columns.

```
tibble(max = c(3, 4, 5), seq = list(1:3, 1:4, 1:5))
```

`tibble::enframe(x, name="name", value="value")`

Converts multi-level list to a tibble with list-cols.  
`enframe(list('3'=1:3, '4'=1:4, '5'=1:5), 'max', 'seq')`

## OUTPUT LIST-COLUMNS FROM OTHER FUNCTIONS

`dplyr::mutate()`, `transmute()`, and `summarise()` will output list-columns if they return a list.

```
mtcars %>%
  group_by(cyl) %>%
  summarise(q = list(quantile(mpg)))
```

## RESHAPE NESTED DATA

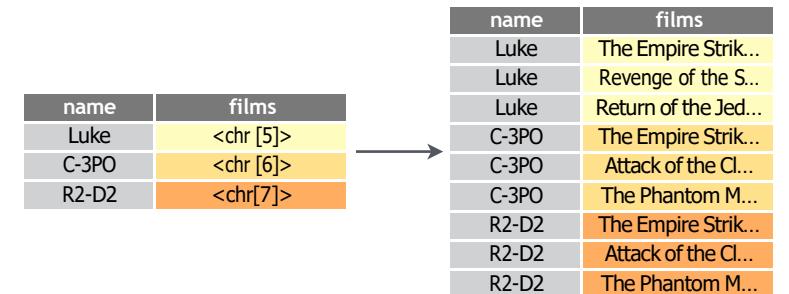
`unnest(data, cols, ..., keep_empty = FALSE)` Flatten nested columns back to regular columns. The inverse of `nest()`.

```
n_storms %>% unnest(data)
```

`unnest_longer(data, col, values_to = NULL, indices_to = NULL)`

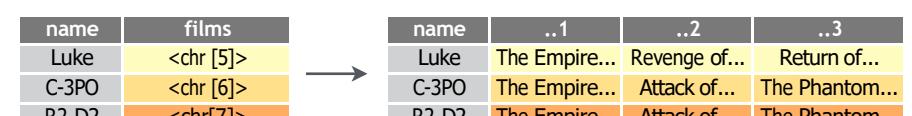
Turn each element of a list-column into a row.

```
starwars %>%
  select(name, films) %>%
  unnest_longer(films)
```



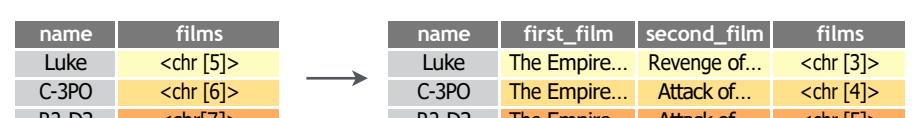
`unnest_wider(data, col)` Turn each element of a list-column into a regular column.

```
starwars %>%
  select(name, films) %>%
  unnest_wider(films)
```



`hoist(.data, .col, ..., .remove = TRUE)` Selectively pull list components out into their own top-level columns. Uses `purrr::pluck()` syntax for selecting from lists.

```
starwars %>%
  select(name, films) %>%
  hoist(films, first_film = 1, second_film = 2)
```



## TRANSFORM NESTED DATA

A vectorized function takes a vector, transforms each element in parallel, and returns a vector of the same length. By themselves vectorized functions cannot work with lists, such as list-columns.

`dplyr::rowwise(.data, ...)` Group data so that each row is one group, and within the groups, elements of list-columns appear directly (accessed with `[[ ]]`, not as lists of length one. When you use `rowwise()`, `dplyr` functions will seem to apply functions to list-columns in a vectorized fashion.



Apply a function to a list-column and create a new list-column.

```
n_storms %>%
  rowwise() %>%
  mutate(n = list(dim(data)))
```

`dim()` returns two values per row  
wrap with `list` to tell `mutate` to create a list-column

Apply a function to a list-column and create a regular column.

```
n_storms %>%
  rowwise() %>%
  mutate(n = nrow(data))
```

`nrow()` returns one integer per row

Collapse multiple list-columns into a single list-column.

```
starwars %>%
  rowwise() %>%
  mutate(transport = list(append(vehicles, starships)))
```

`append()` returns a list for each row, so col type must be list

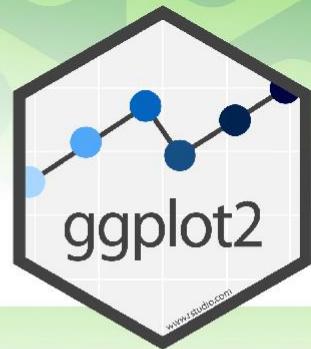
Apply a function to multiple list-columns.

```
starwars %>%
  rowwise() %>%
  mutate(n_transports = length(c(vehicles, starships)))
```

`length()` returns one integer per row

See `purrr` package for more list functions.

# Data visualization with ggplot2 : : CHEAT SHEET



## Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and **geoms**—visual marks that represent data points.



To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.



Complete the template below to build a graph.

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION> (mapping = aes(<MAPPINGS>),
  stat = <STAT>, position = <POSITION>) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION> +
  <SCALE_FUNCTION> +
  <THEME_FUNCTION>
```

The code template is annotated with notes:  
 - A bracket on the right side is labeled "required".  
 - A bracket on the left side is labeled "Not required, sensible defaults supplied".

`ggplot(data = mpg, aes(x = cty, y = hwy))` Begins a plot that you finish by adding layers to. Add one geom function per layer.

`last_plot()` Returns the last plot.

`ggsave("plot.png", width = 5, height = 5)` Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

## Aes

Common aesthetic values.

**color** and **fill** - string ("red", "#RRGGBB")

**linetype** - integer or string (0 = "blank", 1 = "solid", 2 = "dashed", 3 = "dotted", 4 = "dotdash", 5 = "longdash", 6 = "twodash")

**lineend** - string ("round", "butt", or "square")

**linejoin** - string ("round", "mitre", or "bevel")

**size** - integer (line width in mm)

**shape** - integer/shape name or a single character ("a")



## Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables.  
Each function returns a layer.

### GRAPHICAL PRIMITIVES

```
a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))
```

- a + geom\_blank()** and **a + expand\_limits()**  
Ensure limits include values across all plots.
- b + geom\_curve(aes(yend = lat + 1, xend = long + 1, curvature = 1))** - x, y, alpha, angle, color, curvature, linetype, size
- a + geom\_path(lineend = "butt", linejoin = "round", linemitre = 1)** - x, y, alpha, color, group, linetype, size
- a + geom\_polygon(aes(alpha = 50))** - x, y, alpha, color, fill, group, subgroup, linetype, size
- b + geom\_rect(aes(xmin = long, ymin = lat, xmax = long + 1, ymax = lat + 1))** - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size
- a + geom\_ribbon(aes(ymin = unemploy - 900, ymax = unemploy + 900))** - x, ymax, ymin, alpha, color, fill, group, linetype, size

### LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

- b + geom\_abline(aes(intercept = 0, slope = 1))**
- b + geom\_hline(aes(yintercept = lat))**
- b + geom\_vline(aes(xintercept = long))**
- b + geom\_segment(aes(yend = lat + 1, xend = long + 1))**
- b + geom\_spoke(aes(angle = 1:1155, radius = 1))**

### ONE VARIABLE continuous

- ```
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)
```
- c + geom\_area(stat = "bin")** - x, y, alpha, color, fill, linetype, size
  - c + geom\_density(kernel = "gaussian")** - x, y, alpha, color, fill, group, linetype, size, weight
  - c + geom\_dotplot()** - x, y, alpha, color, fill
  - c + geom\_freqpoly()** - x, y, alpha, color, group, linetype, size
  - c + geom\_histogram(binwidth = 5)** - x, y, alpha, color, fill, linetype, size, weight
  - c2 + geom\_qq(aes(sample = hwy))** - x, y, alpha, color, fill, linetype, size, weight

### discrete

```
d <- ggplot(mpg, aes(fl))
```

- d + geom\_bar()** - x, alpha, color, fill, linetype, size, weight

### TWO VARIABLES

#### both continuous

```
e <- ggplot(mpg, aes(cty, hwy))
```

- e + geom\_label(aes(label = cty), nudge\_x = 1, nudge\_y = 1)** - x, y, alpha, color, angle, family, fontface, hjust, lineheight, size, vjust
- e + geom\_point()** - x, y, alpha, color, fill, shape, size, stroke
- e + geom\_quantile()** - x, y, alpha, color, group, linetype, size, weight
- e + geom\_rug(sides = "bl")** - x, y, alpha, color, linetype, size
- e + geom\_smooth(method = lm)** - x, y, alpha, color, fill, group, linetype, size, weight
- e + geom\_text(aes(label = cty), nudge\_x = 1, nudge\_y = 1)** - x, y, alpha, color, family, fontface, hjust, lineheight, size, vjust

### one discrete, one continuous

```
f <- ggplot(mpg, aes(class, hwy))
```

- f + geom\_col()** - x, y, alpha, color, fill, group, linetype, size
- f + geom\_boxplot()** - x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight
- f + geom\_dotplot(binaxis = "y", stackdir = "center")** - x, y, alpha, color, fill, group
- f + geom\_violin(scale = "area")** - x, y, alpha, color, fill, group, linetype, size, weight

### both discrete

```
g <- ggplot(diamonds, aes(cut, color))
```

- g + geom\_count()** - x, y, alpha, color, fill, shape, size, stroke
- e + geom\_jitter(height = 2, width = 2)** - x, y, alpha, color, fill, shape, size

### THREE VARIABLES

```
seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2)); I <- ggplot(seals, aes(long, lat))
```

- I + geom\_contour(aes(z = z))** - x, y, z, alpha, color, group, linetype, size, weight
- I + geom\_contour\_filled(aes(fill = z))** - x, y, alpha, color, fill, group, linetype, size, subgroup
- I + geom\_raster(aes(fill = z), hjust = 0.5, vjust = 0.5, interpolate = FALSE)** - x, y, alpha, fill
- I + geom\_tile(aes(fill = z))** - x, y, alpha, color, fill, linetype, size, width

### continuous bivariate distribution

```
h <- ggplot(diamonds, aes(carat, price))
```

- h + geom\_bin2d(binwidth = c(0.25, 500))** - x, y, alpha, color, fill, linetype, size, weight
- h + geom\_hex()** - x, y, alpha, color, fill, size
- h + geom\_density\_2d()** - x, y, alpha, color, group, linetype, size

### continuous function

```
i <- ggplot(economics, aes(date, unemploy))
```

- i + geom\_area()** - x, y, alpha, color, fill, linetype, size
- i + geom\_line()** - x, y, alpha, color, group, linetype, size
- i + geom\_step(direction = "hv")** - x, y, alpha, color, group, linetype, size

### visualizing error

```
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se))
```

- j + geom\_crossbar(fatten = 2)** - x, y, ymax, ymin, alpha, color, fill, group, linetype, size
- j + geom\_errorbar()** - x, y, ymax, ymin, alpha, color, group, linetype, size, width  
Also **geom\_errorbarh()**.
- j + geom\_linerange()** - x, ymin, ymax, alpha, color, group, linetype, size
- j + geom\_pointrange()** - x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

### maps

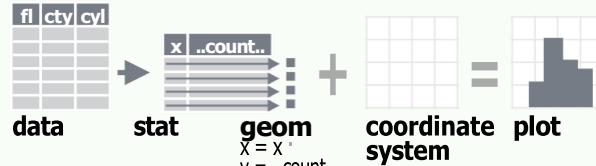
```
data <- data.frame(murder = USArrests$Murder,
  state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))
```

- k + geom\_map(aes(map\_id = state), map = map) + expand\_limits(x = map\$long, y = map\$lat)**  
map\_id, alpha, color, fill, linetype, size

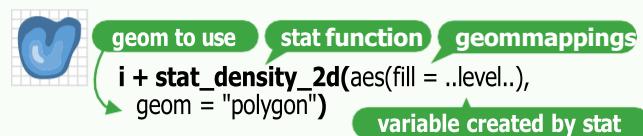
# Stats

An alternative way to build a layer.

A stat builds new variables to plot (e.g., count, prop).



Visualize a stat by changing the default stat of a geom function, `geom_bar(stat = "count")` or by using a stat function, `stat_count(geom = "bar")`, which calls a default geom to make a layer (equivalent to a geom function). Use `..name..` syntax to map stat variables to aesthetics.

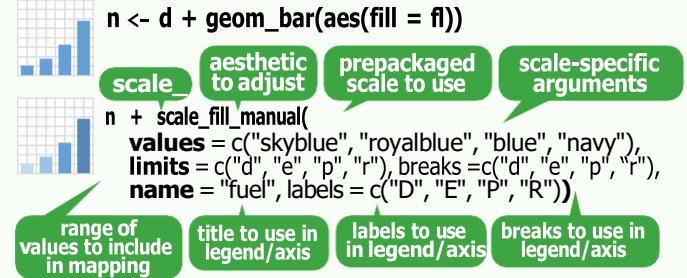


```
c + stat_bin(binwidth = 1, boundary = 10)
x, y | ..count.., ..ncount.., ..density.., ..ndensity..
c + stat_count(width = 1) x, y | ..count.., ..prop..
c + stat_density(adjust = 1, kernel = "gaussian")
x, y | ..count.., ..density.., ..scaled..
e + stat_bin_2d(bins = 30, drop = T)
x, y, fill | ..count.., ..density..
e + stat_bin_hex(bins = 30) x, y, fill | ..count.., ..density..
e + stat_density_2d(contour = TRUE, n = 100)
x, y, color, size | ..level..
e + stat_ellipse(level = 0.95, segments = 51, type = "t")
l + stat_contour(aes(z = z)) x, y, z, order | ..level..
l + stat_summary_hex(aes(z = z), bins = 30, fun = max)
x, y, z, fill | ..value..
l + stat_summary_2d(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..
f + stat_boxplot(coef = 1.5)
x, y | ..lower.., ..middle.., ..upper.., ..width.., ..ymin.., ..ymax..
f + stat_ydensity(kernel = "gaussian", scale = "area") x, y
| ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..
e + stat_ecdf(n = 40) x, y | ..x.., ..y..
e + stat_quantile(quantiles = c(0.1, 0.9),
formula = y ~ log(x), method = "rq") x, y | ..quantile..
e + stat_smooth(method = "lm", formula = y ~ x, se = T,
level = 0.95) x, y | ..se.., ..x.., ..y.., ..ymin.., ..ymax..
ggplot() + xlim(-5, 5) + stat_function(fun = dnorm,
n = 20, geom = "point") x | ..x.., ..y..
ggplot() + stat_qq(aes(sample = 1:100))
x, y, sample | ..sample.., ..theoretical..
e + stat_sum() x, y, size | ..n.., ..prop..
e + stat_summary(fun.data = "mean_cl_boot")
h + stat_summary_bin(fun = "mean", geom = "bar")
e + stat_identity()
e + stat_unique()
```

# Scales

Override defaults with `scales` package.

**Scales** map data values to the visual values of an aesthetic. To change a mapping, add a new scale.



## GENERAL PURPOSE SCALES

Use with most aesthetics

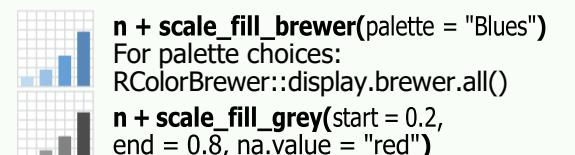
`scale_*_continuous()` - Map cont' values to visual ones.  
`scale_*_discrete()` - Map discrete values to visual ones.  
`scale_*_binned()` - Map continuous values to discrete bins.  
`scale_*_identity()` - Use data values as visual ones.  
`scale_*_manual(values = c())` - Map discrete values to manually chosen visual ones.  
`scale_*_date(date_labels = "%m/%d")`,  
`date_breaks = "2 weeks"` - Treat data values as dates.  
`scale_*_datetime()` - Treat data values as date times. Same as `scale_*_date()`. See `?strptime` for label formats.

## X & Y LOCATION SCALES

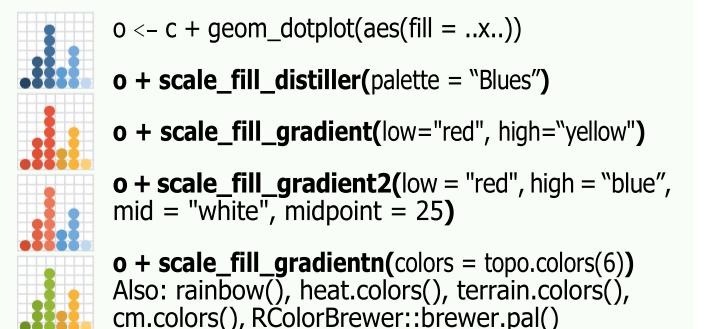
Use with x or y aesthetics (x shown here)

`scale_x_log10()` - Plot x on log10 scale.  
`scale_x_reverse()` - Reverse the direction of the x axis.  
`scale_x_sqrt()` - Plot x on square root scale.

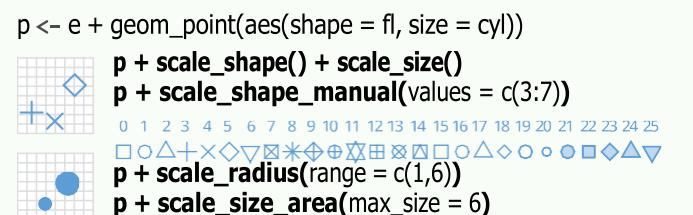
## COLOR AND FILL SCALES (DISCRETE)



## COLOR AND FILL SCALES (CONTINUOUS)



## SHAPE AND SIZE SCALES



# Coordinate Systems

`r <- d + geom_bar()`

`r + coord_cartesian(xlim = c(0, 5))` - xlim, ylim  
The default cartesian coordinate system.

`r + coord_fixed(ratio = 1/2)`  
ratio, xlim, ylim - Cartesian coordinates with fixed aspect ratio between x and y units.

`ggplot(mpg, aes(y = fl)) + geom_bar()`  
Flip cartesian coordinates by switching x and y aesthetic mappings.

`r + coord_polar(theta = "x", direction=1)`  
theta, start, direction - Polar coordinates.

`r + coord_trans(y = "sqrt")` - x, y, xlim, ylim  
Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.

`n + coord_quickmap()`  
`n + coord_map(projection = "ortho", orientation = c(41, -74, 0))` - projection, xlim, ylim  
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.).

## Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

`s <- ggplot(mpg, aes(fl, fill = drv))`

`s + geom_bar(position = "dodge")`  
Arrange elements side by side.

`s + geom_bar(position = "fill")`  
Stack elements on top of one another, normalize height.

`e + geom_point(position = "jitter")`  
Add random noise to X and Y position of each element to avoid overplotting.

`e + geom_label(position = "nudge")`  
Nudge labels away from points.

`s + geom_bar(position = "stack")`  
Stack elements on top of one another.

Each position adjustment can be recast as a function with manual `width` and `height` arguments:  
`s + geom_bar(position = position_dodge(width = 1))`

## Themes

`r + theme_bw()`  
White background with grid lines.

`r + theme_gray()`  
Grey background (default theme).

`r + theme_dark()`  
Dark for contrast.

`r + theme_classic()`

`r + theme_light()`

`r + theme_linedraw()`

`r + theme_minimal()`

Minimal theme.

`r + theme_void()`

Empty theme.

`r + theme()` Customize aspects of the theme such as axis, legend, panel, and facet properties.  
`r + ggtitle("Title") + theme(plot.title.position = "plot")`  
`r + theme(panel.background = element_rect(fill = "blue"))`

# Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

`t <- ggplot(mpg, aes(cty, hwy)) + geom_point()`

`t + facet_grid(cols = vars(fl))`  
Facet into columns based on fl.

`t + facet_grid(rows = vars(year))`  
Facet into rows based on year.

`t + facet_grid(rows = vars(year), cols = vars(fl))`  
Facet into both rows and columns.

`t + facet_wrap(vars(fl))`  
Wrap facets into a rectangular layout.

Set `scales` to let axis limits vary across facets.

`t + facet_grid(rows = vars(drv), cols = vars(fl), scales = "free")`  
x and y axis limits adjust to individual facets:

"`free_x`" - x axis limits adjust  
"`free_y`" - y axis limits adjust

Set `labeler` to adjust facet label:

`t + facet_grid(cols = vars(fl), labeler = label_both)`

`fl: c fl: d fl: e fl: p fl: r`

`t + facet_grid(rows = vars(fl), labeler = label_bquote(alpha ^ .(fl)))`

`l^c l^d l^e l^p l^r`

## Labels and Legends

Use `labs()` to label the elements of your plot.

`t + labs(x = "New x axis label", y = "New y axis label", title = "Add a title above the plot", subtitle = "Add a subtitle below title", caption = "Add a caption below plot", alt = "Add alt text to the plot",  
<code> = "New </code> legend title")`

`t + annotate(geom = "text", x = 8, y = 9, label = "A")`  
Places a geom with manually selected aesthetics.

`p + guides(x = guide_axis(n.dodge = 2))` Avoid crowded or overlapping labels with `guide_axis(n.dodge` or `angle`).

`n + guides(fill = "none")` Set legend type for each aesthetic: colorbar, legend, or none (no legend).

`n + theme(legend.position = "bottom")` Place legend at "bottom", "top", "left", or "right".

`n + scale_fill_discrete(name = "Title", labels = c("A", "B", "C", "D", "E"))`

Set legend title and labels with a scale function.

## Zooming

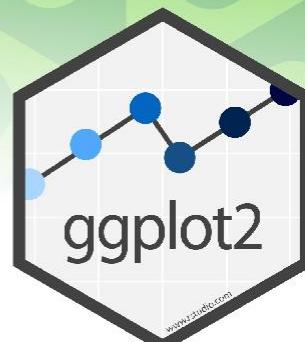
**Without clipping** (preferred):

`t + coord_cartesian(xlim = c(0, 100), ylim = c(10, 20))`

**With clipping** (removes unseen data points):

`t + xlim(0, 100) + ylim(10, 20)`

`t + scale_x_continuous(limits = c(0, 100)) + scale_y_continuous(limits = c(0, 100))`

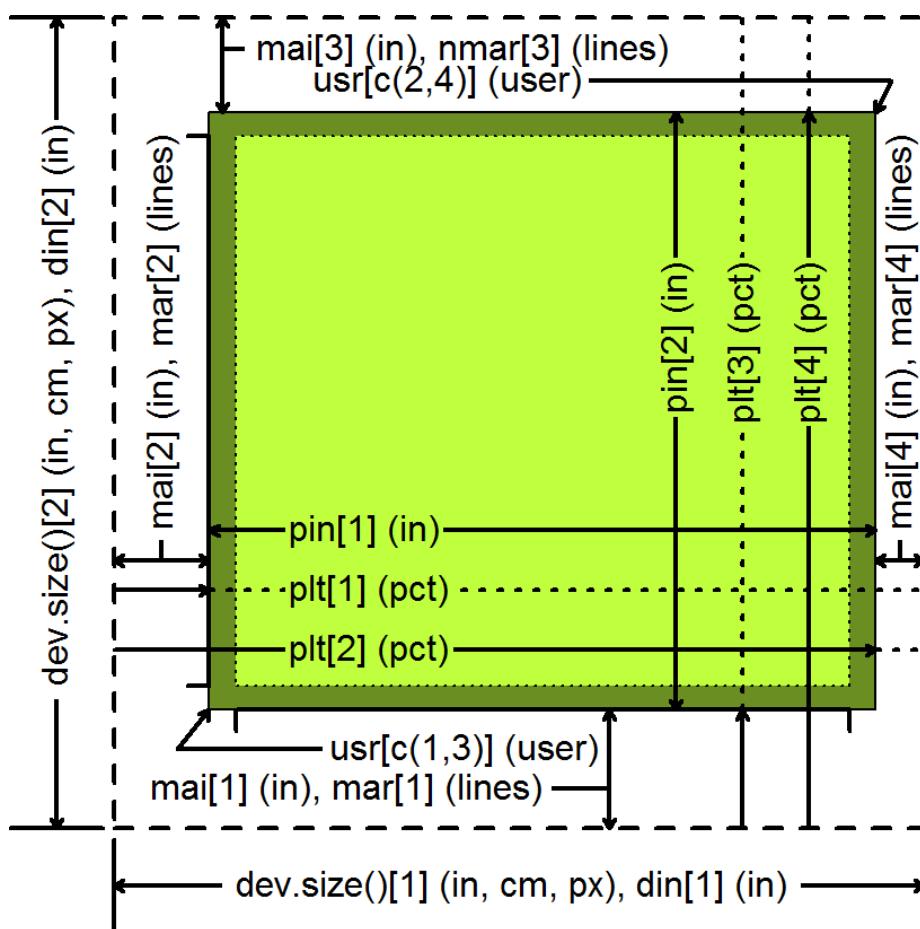


# How Big is Your Graph?

## An R Cheat Sheet

### Introduction

All functions that open a device for graphics will have **height** and **width** arguments to control the size of the graph and a **pointsize** argument to control the relative font size. In **knitr**, you control the size of the graph with the chunk options, **fig.width** and **fig.height**. This sheet will help you with calculating the size of the graph and various parts of the graph within R.



### Your graphics device

`dev.size()` (width, height)  
`par("din")` (r.o.) (width, height) in inches

Both the `dev.size` function and the `din` argument of `par` will tell you the size of the graphics device. The `dev.size` function will report the size in

1. inches (`units="in"`), the default
2. centimeters (`units="cm"`)
3. pixels (`units="px"`)

Like several other `par` arguments, `din` is read only (r.o.) meaning that you can ask its current value (`par("din")`) but you cannot change it (`par(din=c(5,7))` will fail).

### Your plot margins

`par("mai")` (bottom, left, top, right) in inches  
`par("mar")` (bottom, left, top, right) in lines

Margins provide you space for your axes, axis labels, and titles.

A "line" is the amount of vertical space needed for a line of text.

If your graph has no axes or titles, you can remove the margins (and maximize the plotting region) with

`par(mar=rep(0,4))`

### Your plotting region

`par("pin")` (width, height) in inches  
`par("plt")` (left, right, bottom, top) in pct

The `pin` argument `par` gives you the size of the plotting region (the size of the device minus the size of the margins) in inches.

The `plt` argument `par` gives you the percentage of the device from the left/bottom edge up to the left edge of the plotting region, the right edge, the bottom edge, and the top edge. The first and third values are equivalent to the percentage of space devoted to the left and bottom margins. Subtract the second and fourth values from 1 to get the percentage of space devoted to the right and top margins.

### Your x-y coordinates

`par("usr")` (xmin, ymin, xmax, ymax)

Your x-y coordinates are the values you use when plotting your data. This normally is not the same as the values you specified with the `xlim` and `ylim` arguments in `plot`. By default, R adds an extra 4% to the plotting range (see the dark green region on the figure) so that points right up on the edges of your plot do not get partially clipped. You can override this by setting `xaxs="i"` and/or the `yaxs="i"` in `par`.

Run `par("usr")` to find the minimum X value, the maximum X value, the minimum Y value, and the maximum Y value. If you assign new values to `usr`, you will update the x-y coordinates to the new values.

### Getting a square graph

`par("pty")`

You can produce a square graph manually by setting the width and height to the same value and setting the margins so that the sum of the top and bottom margins equal the sum of the left and right margins. But a much easier way is to specify `pty="s"`, which adjusts the margins so that the size of the plotting region is always square, even if you resize the graphics window.

### Converting units

For many applications, you need to be able to translate user coordinates to pixels or inches. There are some cryptic shortcuts, but the simplest way is to get the range in user coordinates and measure the proportion of the graphics device devoted to the plotting region.

`user.range <- par("usr")[c(2,4)] - par("usr")[c(1,3)]`

`region.pct <- par("plt")[c(2,4)] - par("plt")[c(1,3)]`

`region.px <- dev.size(units="px") * region.pct`

`px.per.xy <- region.px / user.range`

To convert a horizontal or distance from the x-coordinate value to pixels, multiply by `px.per.xy[1]`. To convert a vertical distance, multiply by `region.px.per.xy[2]`. To convert a diagonal distance, you need to invoke Pythagoras.

`a.px <- x.dist*px.per.xy[1]`  
`b.px <- y.dist*px.per.xy[2]`  
`c.px <- sqrt(a.px^2+b.px^2)`

To rotate a string to match the slope of a line segment, you need to convert the distances to pixels, calculate the arctangent, and convert from radians to degrees.

```
segments(x0, y0, x1, y1)
delta.x <- (x1 - x0) * px.per.xy[1]
delta.y <- (y1 - y0) * px.per.xy[2]
angle.radians <- atan2(delta.y, delta.x)
angle.degrees <- angle.radians * 180 / pi
text(x1, y1, "TEXT", srt=angle.degrees)
```

## Panels

`par("fig")` (width, height) in pct  
`par("fin")` (width, height) in inches

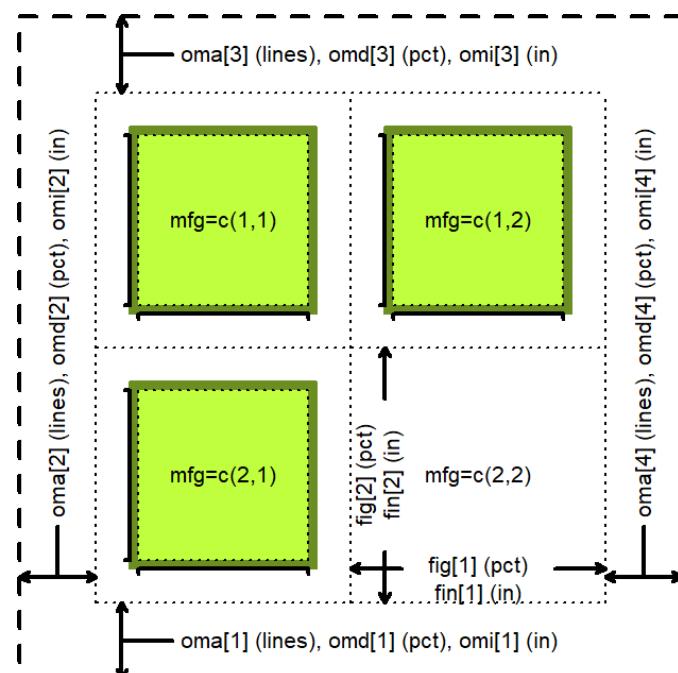
If you display multiple plots within a single graphics window (e.g., with the `mrow` or `mfcol` arguments of `par` or with the `layout` function), then the `fig` and `fin` arguments will tell you the size of the current subplot window in percent or inches, respectively.

`par("oma")` (bottom, left, top, right) in lines  
`par("omd")` (bottom, left, top, right) in pct  
`par("omi")` (bottom, left, top, right) in inches

Each subplot will have margins specified by `mai` or `mar`, but no outer margin around the entire set of plots, unless you specify them using `oma`, `omd`, or `omi`. You can place text in the outer margins using the `mtext` function with the argument `outer=TRUE`.

`par("mfg")` (r, c) or (r, c, maxr, maxc)

The `mfg` argument of `par` will allow you to jump to a subplot in a particular row and column. If you query with `par("mfg")`, you will get the current row and column followed by the maximum row and column.



## Character and string sizes

### `strheight()`

The `strheight` functions will tell you the height of a specified string in inches (`units="inches"`), x-y user coordinates (`units="user"`) or as a percentage of the graphics device (`units="figure"`).

For a single line of text, `strheight` will give you the height of the letter "M". If you have a string with one or more linebreaks ("\\n"), the `strheight` function will measure the height of the letter "M" plus the height of one or more additional lines. The height of a line is dependent on the line spacing, set by the `lheight` argument of `par`. The default line height (`lheight=1`), corresponding to single spaced lines, produces a line height roughly 1.5 times the height of "M".

### `strwidth()`

The `strwidth` function will produce different widths to individual characters, representing the proportional spacing used by most fonts (a "W" using much more space than an "i"). For the width of a string, the `strwidth` function will sum up the lengths of the individual characters in the string.

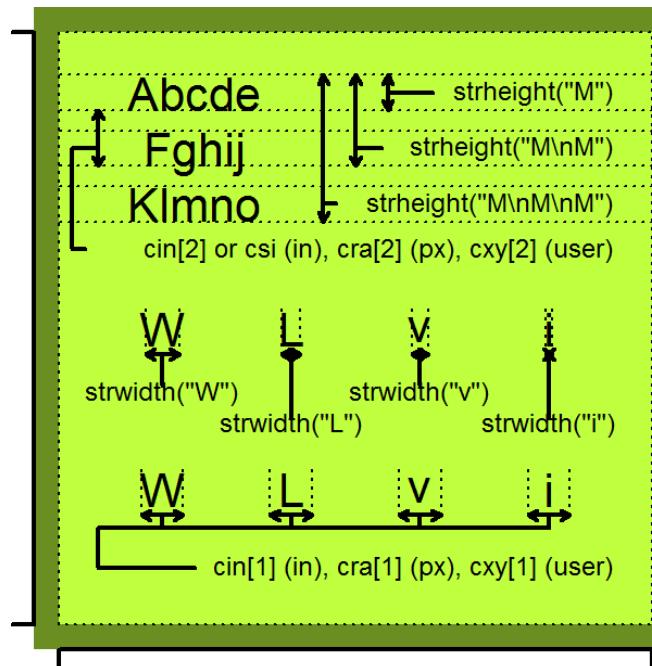
`par("cin")` (r.o.) (width, height) in inches  
`par("csi")` (r.o.) height in inches  
`par("cra")` (r.o.) (width, height) in pixels  
`par("cxy")` (r.o.) (width, height) in xy coordinates

The single value returned by the `csi` argument of `par` gives you the height of a line of text in inches. The second of the two values returned by `cin`, `cra`, and `cxy` gives you the height of a line, in inches, pixels, or xy (user) coordinates.

The first of the two values returned by the `cin`, `cra`, and `cxy` arguments to `par` gives you the approximate width of a single character, in inches, pixels, or xy (user) coordinates. The width, very slightly smaller than the actual width of the letter "W", is a rough estimate at best and ignores the variable width of individual letters.

These values are useful, however, in providing fast ratios of the relative sizes of the differing units of measure

```
px.per.in <- par("cra") / par("cin")
px.per.xy <- par("cra") / par("cxy")
xy.per.in <- par("cxy") / par("cin")
```



## If your fonts are too big or too small

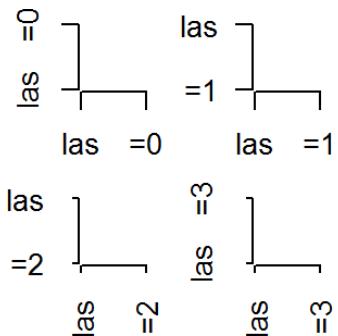
Fixing this takes a bit of trial and error.

- Specify a larger/smaller value for the `pointsize` argument when you open your graphics device.
- Try opening your graphics device with different values for `height` and `width`. Fonts that look too big might be better proportioned in a larger graphics window.
- Use the `cex` argument to increase or decrease the relative size of your fonts.

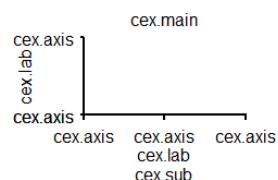
## If your axes don't fit

There are several possible solutions.

- You can assign wider margins using the `mar` or `mai` argument in `par`.
- You can change the orientation of the axis labels with `las`. Choose among
  - `las=0` both axis labels parallel
  - `las=1` both axis labels horizontal
  - `las=2` both axis labels perpendicular
  - `las=3` both axis labels vertical



- change the relative size of the font
  - `cex.axis` for the tick mark labels.
  - `cex.lab` for `xlab` and `ylab`.
  - `cex.main` for the main title
  - `cex.sub` for the subtitle.



# R Syntax Comparison :: CHEAT SHEET

## Dollar sign syntax

```
goal(data$x, data$y)
```

### SUMMARY STATISTICS:

one continuous variable:

```
mean(mtcars$mpg)
```

one categorical variable:

```
table(mtcars$cyl)
```

two categorical variables:

```
table(mtcars$cyl, mtcars$am)
```

one continuous, one categorical:

```
mean(mtcars$mpg[mtcars$cyl==4])
```

```
mean(mtcars$mpg[mtcars$cyl==6])
```

```
mean(mtcars$mpg[mtcars$cyl==8])
```

### PLOTTING:

one continuous variable:

```
hist(mtcars$disp)
```

```
boxplot(mtcars$disp)
```

one categorical variable:

```
barplot(table(mtcars$cyl))
```

two continuous variables:

```
plot(mtcars$disp, mtcars$mpg)
```

two categorical variables:

```
mosaicplot(table(mtcars$am, mtcars$cyl))
```

one continuous, one categorical:

```
histogram(mtcars$disp[mtcars$cyl==4])
```

```
histogram(mtcars$disp[mtcars$cyl==6])
```

```
histogram(mtcars$disp[mtcars$cyl==8])
```

```
boxplot(mtcars$disp[mtcars$cyl==4])
```

```
boxplot(mtcars$disp[mtcars$cyl==6])
```

```
boxplot(mtcars$disp[mtcars$cyl==8])
```

### WRANGLING:

subsetting:

```
mtcars[mtcars$mpg>30, ]
```

making a new variable:

```
mtcars$efficient[mtcars$mpg>30] <- TRUE
```

```
mtcars$efficient[mtcars$mpg<30] <- FALSE
```

## Formula syntax

```
goal(y~x|z, data=data, group=w)
```

### SUMMARY STATISTICS:

one continuous variable:

```
mosaic::mean(~mpg, data=mtcars)
```

one categorical variable:

```
mosaic::tally(~cyl, data=mtcars)
```

two categorical variables:

```
mosaic::tally(cyl~am, data=mtcars)
```

one continuous, one categorical:

```
mosaic::mean(mpg~cyl, data=mtcars)
```

tilde

### PLOTTING:

one continuous variable:

```
lattice::histogram(~disp, data=mtcars)
```

```
lattice::bwplot(~disp, data=mtcars)
```

one categorical variable:

```
mosaic::bargraph(~cyl, data=mtcars)
```

two continuous variables:

```
lattice::xyplot(mpg~disp, data=mtcars)
```

two categorical variables:

```
mosaic::bargraph(~am, data=mtcars, group=cyl)
```

one continuous, one categorical:

```
lattice::histogram(~disp|cyl, data=mtcars)
```

```
lattice::bwplot(cyl~disp, data=mtcars)
```

The variety of R syntaxes give you many ways to “say” the same thing

read across the cheatsheet to see how different syntaxes approach the same problem

## Tidyverse syntax

```
data %>% goal(x)
```

### SUMMARY STATISTICS:

one continuous variable:

```
mtcars %>% dplyr::summarize(mean(mpg))
```

one categorical variable:

```
mtcars %>% dplyr::group_by(cyl) %>%  
dplyr::summarize(n())
```

two categorical variables:

```
mtcars %>% dplyr::group_by(cyl, am) %>%  
dplyr::summarize(n())
```

one continuous, one categorical:

```
mtcars %>% dplyr::group_by(cyl) %>%  
dplyr::summarize(mean(mpg))
```

### PLOTTING:

one continuous variable:

```
ggplot2::qplot(x=mpg, data=mtcars, geom = "histogram")
```

```
ggplot2::qplot(y=disp, x=1, data=mtcars, geom="boxplot")
```

one categorical variable:

```
ggplot2::qplot(x=cyl, data=mtcars, geom="bar")
```

two continuous variables:

```
ggplot2::qplot(x=disp, y=mpg, data=mtcars, geom="point")
```

two categorical variables:

```
ggplot2::qplot(x=factor(cyl), data=mtcars, geom="bar") +  
facet_grid(.~am)
```

one continuous, one categorical:

```
ggplot2::qplot(x=disp, data=mtcars, geom = "histogram") +  
facet_grid(.~cyl)
```

```
ggplot2::qplot(y=disp, x=factor(cyl), data=mtcars,  
geom="boxplot")
```

### WRANGLING:

subsetting:

```
mtcars %>% dplyr::filter(mpg>30)
```

making a new variable:

```
mtcars <- mtcars %>%  
dplyr::mutate(efficient = if_else(mpg>30, TRUE, FALSE))
```

the pipe

# R Syntax Comparison :: CHEAT SHEET

**Syntax** is the set of rules that govern what code works and doesn't work in a programming language. Most programming languages offer one standardized syntax, but R allows package developers to specify their own syntax. As a result, there is a large variety of (equally valid) R syntaxes.

The three most prevalent R syntaxes are:

1. The **dollar sign syntax**, sometimes called **base R syntax**, expected by most base R functions. It is characterized by the use of `dataset$variablename`, and is also associated with square bracket subsetting, as in `dataset[1, 2]`. Almost all R functions will accept things passed to them in dollar sign syntax.
2. The **formula syntax**, used by modeling functions like `lm()`, lattice graphics, and mosaic summary statistics. It uses the tilde (~) to connect a response variable and one (or many) predictors. Many base R functions will accept formula syntax.
3. The **tidyverse syntax** used by `dplyr`, `tidyr`, and more. These functions expect data to be the first argument, which allows them to work with the "pipe" (`%>%`) from the `magrittr` package. Typically, `ggplot2` is thought of as part of the tidyverse, although it has its own flavor of the syntax using plus signs (+) to string pieces together. `ggplot2` author Hadley Wickham has said the package would have had different syntax if he had written it after learning about the pipe.

Educators often try to teach within one unified syntax, but most R programmers use some combination of all the syntaxes.

## Internet research tip:

If you are searching on google, StackOverflow, or another favorite online source and see code in a syntax you don't recognize:

- Check to see if the code is using one of the three common syntaxes listed on this cheatsheet
- Try your search again, using a keyword from the syntax name ("tidyverse") or a relevant package ("mosaic")



Sometimes particular syntaxes work, but are considered dangerous to use, because they are so easy to get wrong. For example, passing variable names without assigning them to a named argument.

## Even more ways to say the same thing

Even within one syntax, there are often variations that are equally valid. As a case study, let's look at the `ggplot2` syntax. `ggplot2` is the plotting package that lives within the tidyverse. If you read down this column, all the code here produces the same graphic.

### quickplot

`qplot()` stands for quickplot, and allows you to make quick plots. It doesn't have the full power of `ggplot2`, and it uses a slightly different syntax than the rest of the package.

```
ggplot2::qplot(x=disp, y=mpg, data=mtcars, geom="point")
```

```
ggplot2::qplot(x=disp, y=mpg, data=mtcars) !
```

```
ggplot2::qplot(disp, mpg, data=mtcars) ! !
```

read down this column for many pieces of code in one syntax that look different but produce the same graphic

### ggplot

To unlock the power of `ggplot2`, you need to use the `ggplot()` function (which sets up a plotting region) and add `geoms` to the plot.

```
ggplot2::ggplot(mtcars) +  
  geom_point(aes(x=disp, y=mpg))
```

```
ggplot2::ggplot(data=mtcars) +  
  geom_point(mapping=aes(x=disp, y=mpg))
```

plus adds layers

```
ggplot2::ggplot(mtcars, aes(x=disp, y=mpg)) +  
  geom_point()
```

```
ggplot2::ggplot(mtcars, aes(x=disp)) +  
  geom_point(aes(y=mpg))
```

### ggformula

The "third and a half way" to use the formula syntax, but get `ggplot2`-style graphics

```
ggformula::gf_point(mpg~disp, data= mtcars)
```

### formulas in base plots

Base R plots will also take the formula syntax, although it's not as commonly used

```
plot(mpg~disp, data=mtcars)
```

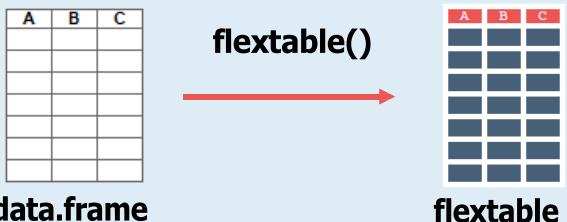
# Tabular reporting with *flextable* : : CHEAT SHEET



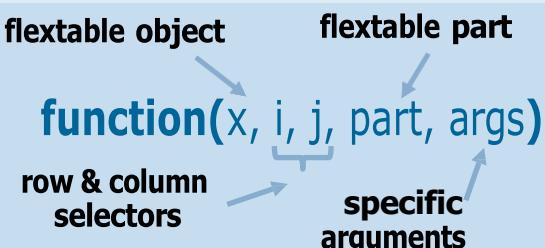
## Basics

The **flextable** package provides a framework for **easily create tables for reporting and publications**.

Functions are provided to let users create tables, modify, format and define their content.



## GENERAL FUNCTION'S STRUCTURE



## TABLE PARTS AND THEIRS DEFAULT VALUES

|  |                   |     |
|--|-------------------|-----|
|  | header : colnames | all |
|  | body : data       |     |
|  | footer : empty    |     |

## Selectors

i: row selector

j: column selector

### FORMULA

i = ~ col %in% "xxx"  
col : column name  
xxx : value  
j = ~ col1 + col2  
col\* : column name

### CHARACTER VECTOR

j = c("col1", "col2")  
col\* : column name

### INTEGER VECTOR

i = 1:3, j = 1:3

### LOGICAL VECTOR

i = c(TRUE, FALSE), j = c(TRUE, FALSE)

## Format

### GENERAL

```
Abc ft <- flextable(data)

get_flextable_defaults() : get flextable defaults
formatting properties
set_flextable_defaults() : modify flextable
defaults formatting properties
init_flextable_defaults() : re-init all values
with the package defaults
style(pr_t, pr_p, pr_c) : modify flextable text,
paragraphs and cells formatting properties
(needs officer package)
pr_t: object of class fp_text
pr_p object of class fp_par
pr_c: object of class fp_cell
```

### TEXT

```
Abc font(ft, fontname = "Brush Script MT")
Abc fontsize(ft, size = 7)
Abc italic(ft, italic = TRUE)
Abc bold(ft, bold = TRUE)
Abc color(ft, color = "#eb5555")
Abc highlight(ft, color = "yellow")
Abc rotate(ft, rotation = "tblr")
```

### CELL

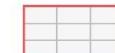
```
Abc align(ft, align = "center")
Abc valign(ft, valign = "top")
Abc padding(ft, padding = 10)
Abc bg(ft, bg = "#475f77")
Abc line_spacing(ft, space = 1.6)
```

### THEME ft <- theme\_\*(ft)

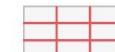
| alafoli(ft)     | booktabs(ft) | box(ft)     | tron(ft)  |
|-----------------|--------------|-------------|-----------|
|                 |              |             |           |
| tron_legacy(ft) | vader(ft)    | vanilla(ft) | zebra(ft) |
|                 |              |             |           |

## BORDER

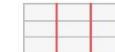
brdr <- fp\_border(color = "#eb5555", width = 1.5)



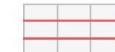
border\_outer(ft, border = brdr)



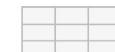
border\_inner(ft, border = brdr)



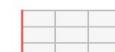
border\_inner\_v(ft, border = brdr)



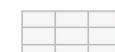
border\_inner\_h(ft, border = brdr)



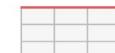
border\_remove(ft)



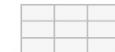
vline\_left(ft, border = brdr)



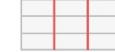
vline\_right(ft, border = brdr)



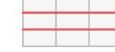
hline\_top(ft, border = brdr)



hline\_bottom(ft, border = brdr)



vline(ft, j=1:2, border = brdr)



hline(ft, i = 1:2, border = brdr)

## Officer

**fp\_text()** : Text formatting properties  
color, font.size, bold, italic, underlined, font.family, vertical.align, shading.color

**fp\_par()** : Paragraph formatting properties  
text.align, padding, line\_spacing, border, shading.color, padding.bottom, padding.top, padding.left, padding.right, border.bottom, border.left, border.top, border.right

**fp\_cell()** : Cell formatting properties  
border, border.bottom, border.left, border.top, border.right, vertical.align, margin, margin.bottom, margin.top, margin.left, margin.right, background.color, text.direction

**fp\_border()**: border properties object  
color, style, width

**update(x, args)**: update an object of class fp\_\*



## Layout

### HEADER AND FOOTER

#### COLWIDTHS

```
Abc add_header_row(ft, values = c("a",
"b", "c"), colwidths = c(1, 1, 1), top =
FALSE)
Abc add_footer_row(ft, values = c("", "", ""),
colwidths = c(1, 1, 1))
```

#### IN LINE

```
Abc add_header_lines(ft, values = "line",
top = FALSE)
Abc add_footer_lines(ft, values = "line")
```

#### COLNAME

```
Abc add_header(ft, A = "a", B = "b", top =
FALSE)
Abc add_body(ft, A = "a", B = "b", C = "")
Abc add_footer(ft, A = "", B = "")
```

#### GENERAL

```
Aaa Bbb Ccc set_header_labels(ft, A = "Aaa", B =
"Bbb", C = "Ccc")
Abc delete_part(ft, part = "body")
```

### 1, 2, 3, 4

|  |  |  |  |
|--|--|--|--|
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### merge\_at(i = 1:2, j = 1:2)

### merge\_h(ft)

### merge\_v(ft)

### merge\_h\_range(ft, i = ~ C %in% "0", j1 = "A", j2 = "B")

**fix\_border\_issues(ft)**: fix border issues when cell are merged

## CAPTIONS & FOOTNOTES

my caption

```
A B C set_caption(ft, caption = "my
caption")
```

```
A B C footnote(ft, j = 1, value =
as_paragraph(c("footnote 1")),
ref_symbols = c("1"), part =
"header")
```

# Tabular reporting with *flextable* : : CHEAT SHEET



## Table size

```
ft <- flextable(data)
flextable_dim(ft):
$widths
[1] 2.25
$heights
[1] 1.75
$aspect_ratio
[1] 0.78

dim(ft):
$widths
A   B   C
0.75 0.75 0.75
$heights
[1] 0.25 0.25 0.25
0.25 0.25 0.25

dim_pretty(ft):
$widths
[1] 0.22 0.22 0.22
$heights
[1] 0.22 0.22 0.22 0.22 0.22 0.22
autofit(ft, add_w = w, add_h = h)
w = 0, h = 0
width: 0.22
width: 0.22
width: 0.22
width: 0.22
width: 0.22
width: 0.22
width(ft, i = 1, width = 0.5)
width: 0.5 width: 0.75
ft <- height(ft, i = 1, height = 0.40, part = "body")
ft <- height(ft, i = 4, height = 0.30, part = "body")
A   B   C
ft <- hrule(ft, rule = "auto", part = "header")
ft <- hrule(ft, i = 1, rule = "exact", part =
"body"): size exactly at 0.4
ft <- hrule(ft, i = 4, rule = "atleast", part =
"body"): size atleast at 0.3
```

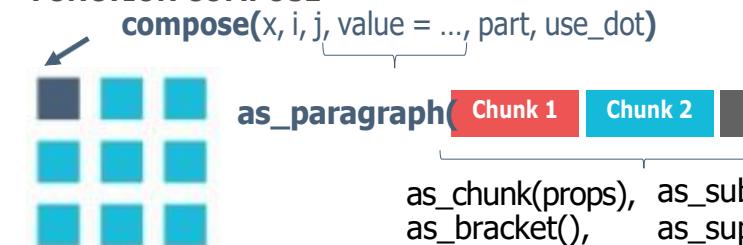
## Cell content

### SIMPLE FORMATTING

| args         | colformat_char | colformat_date | colformat_datetime | colformat_image | colformat_double | colformat_int | colformat_lgl | colformat_num |
|--------------|----------------|----------------|--------------------|-----------------|------------------|---------------|---------------|---------------|
| x            | ✓              | ✓              | ✓                  | ✓               | ✓                | ✓             | ✓             | ✓             |
| i            | ✓              | ✓              | ✓                  | ✓               | ✓                | ✓             | ✓             | ✓             |
| j            | ✓              | ✓              | ✓                  | ✓               | ✓                | ✓             | ✓             | ✓             |
| na_str       | ✓              | ✓              | ✓                  | ✓               | ✓                | ✓             | ✓             | ✓             |
| prefix       | ✓              | ✓              | ✓                  | ✓               | ✓                | ✓             | ✓             | ✓             |
| suffix       | ✓              | ✓              | ✓                  | ✓               | ✓                | ✓             | ✓             | ✓             |
| big.mark     |                |                |                    |                 |                  |               |               |               |
| decimal.mark |                |                |                    |                 |                  |               |               |               |
| fmt_date     |                |                | ✓                  |                 |                  |               |               |               |
| fmt_datetime |                |                |                    | ✓               |                  |               |               |               |
| width        |                |                |                    |                 | ✓                |               |               |               |
| true         |                |                |                    |                 |                  | ✓             |               |               |
| digits       |                |                |                    |                 |                  | ✓             |               |               |
| height       |                |                |                    |                 |                  | ✓             |               |               |
| false        |                |                |                    |                 |                  |               | ✓             |               |

### MULTI CONTENT

#### FUNCTION COMPOSE



**use\_dot()**: by default use\_dot=FALSE; if use\_dot=TRUE, value is evaluated within a data.frame augmented of a column named . containing the jth column

## Rendering

**flextable** default format is **HTML output** printed in the rstudio viewer pane.  
**flextable** objects can be rendered in **HTML** format, **Microsoft Word**, **Microsoft PowerPoint** and **PDF**.



### SIMPLE EXPORT

```
save_as_html(ft, "ft.html")
save_as_docx(ft, "ft.docx")
save_as_pptx(ft, "ft.pptx")
save_as_image(ft, "ft.png")
```

### INTERACTIVE SESSION

```
print(ft, preview = "docx")
print(ft, preview = "pptx")
```

### RMARKDOWN DOCUMENTS

```
```{r}
library(flextable)
ft <- flextable(ft)
ft
```
```

### LOOPING IN RMARKDOWN WITH FOR

**flextable\_to\_rmd(ft)**

### WITH OFFICER

```
ph_with(ppt, value = ft) (PowerPoint)
ppt : an rpptx object
body_add_flextable(value = ft) (Word)
```

### IN SHINY

```
library(shiny)
library(flextable)
ft <- flextable(data)
# In UI
uiOutput("ft")

# In server
output$ft <- renderUI({
  htmltools_value(ft)
})
```