

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

Creating Vectors

c(2, 4, 6)	2 4 6	Join elements into a vector
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
rep(1:2, each=3)	1 1 1 2 2 2	Repeat elements of a vector

Vector Functions

sort(x)

Return x sorted.

rev(x)

Return x reversed.

table(x)

See counts of values.

unique(x)

See unique values.

Selecting Vector Elements

By Position

x[4]

The fourth element.

x[-4]

All but the fourth.

x[2:4]

Elements two to four.

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.

x[x < 0]

All elements less than zero.

x[x %in% c(1, 2, 5)]

Elements in the set 1, 2, 5.

Named Vectors

x['apple']

Element with name 'apple'.

Programming

For Loop

```
for (variable in sequence){  
  Do something  
}
```

Example

```
for (i in 1:4){  
  j <- i + 10  
  print(j)  
}
```

While Loop

```
while (condition){  
  Do something  
}
```

Example

```
while (i < 5){  
  print(i)  
  i <- i + 1  
}
```

Functions

```
function_name <- function(var){  
  Do something  
  return(new_variable)  
}
```

Example

```
square <- function(x){  
  squared <- x*x  
  return(squared)  
}
```

Reading and Writing Data

Also see the **readr** package.

Input	Output	Description
df <- read.table('file.txt')	write.table(df, 'file.txt')	Read and write a delimited text file.
df <- read.csv('file.csv')	write.csv(df, 'file.csv')	Read and write a comma separated value file. This is a special case of read.table/write.table.
load('file.RData')	save(df, file = 'file.Rdata')	Read and write an R data file, a file type special for R.

Conditions	a == b	Are equal	a > b	Greater than	a >= b	Greater than or equal to	is.na(a)	Is missing
	a != b	Not equal	a < b	Less than	a <= b	Less than or equal to	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



`m[, 1]` - Select a column



`m[2, 3]` - Select an element

`t(m)`

Transpose

`m %*% n`

Matrix Multiplication

`solve(m, n)`

Find x in: $m \cdot x = n$

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]]`

Second element of l.

`l[1]`

New list with only the first element.

`l$x`

Element named x.

`l['y']`

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

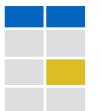
`df[, 2]`



`df[2,]`



`df[2, 2]`



List subsetting

<code>df\$x</code>	<code>df[[2]]</code>
--------------------	----------------------

Understanding a data frame

`View(df)` See the full data frame.

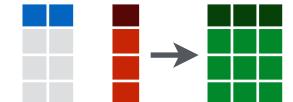
`head(df)` See the first 6 rows.

`nrow(df)` Number of rows.

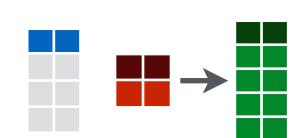
`ncol(df)` Number of columns.

`dim(df)` Number of columns and rows.

`cbind` - Bind columns.



`rbind` - Bind rows.



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.

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.

`glm(y ~ x, data=df)`

Generalised linear model.

`summary`

Get more detailed information out a model.

`t.test(x, y)`

Perform a t-test for difference between means.

`pairwise.t.test`

Perform a t-test for paired data.

`prop.test`

Test for a difference between proportions.

`aov`

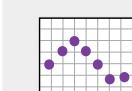
Analysis of variance.

Distributions

	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	<code>rnorm</code>	<code>dnorm</code>	<code>pnorm</code>	<code>qnorm</code>
Poisson	<code>rpois</code>	<code>dpois</code>	<code>ppois</code>	<code>qpois</code>
Binomial	<code>rbinom</code>	<code>dbinom</code>	<code>pbinom</code>	<code>qbinom</code>
Uniform	<code>runif</code>	<code>dunif</code>	<code>unif</code>	<code>qunif</code>

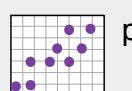
Plotting

Also see the `ggplot2` package.



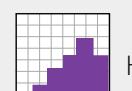
`plot(x)`

Values of x in order.



`plot(x, y)`

Values of x against y.



`hist(x)`

Histogram of x.

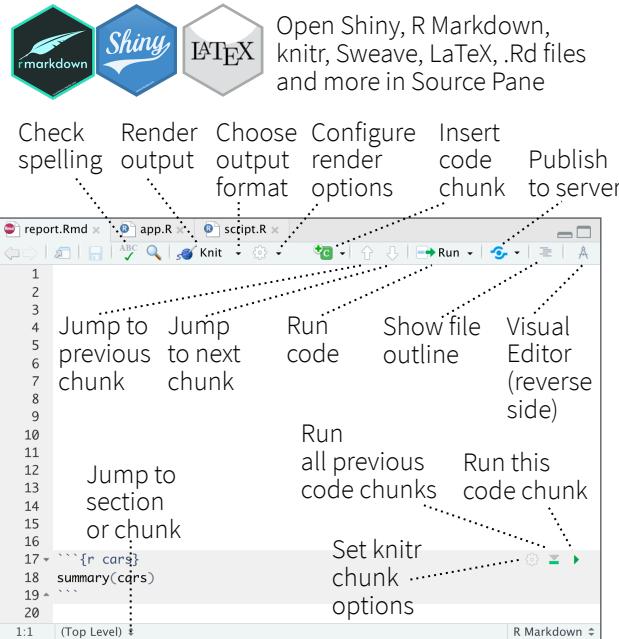
Dates

See the `lubridate` package.

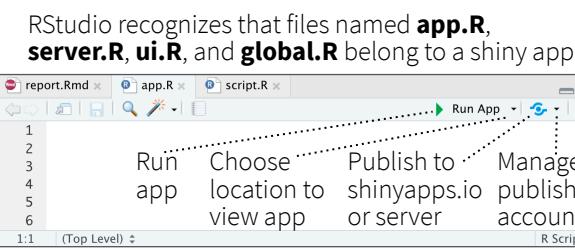
RStudio IDE :: CHEATSHEET



Documents and Apps



Access markdown guide at **Help > Markdown Quick Reference**
See reverse side for more on **Visual Editor**



Package Development

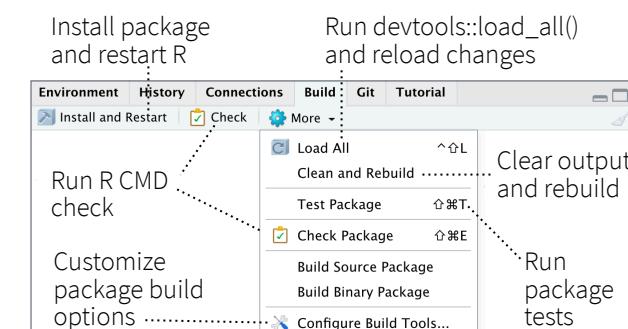


Create a new package with **File > New Project > New Directory > R Package**

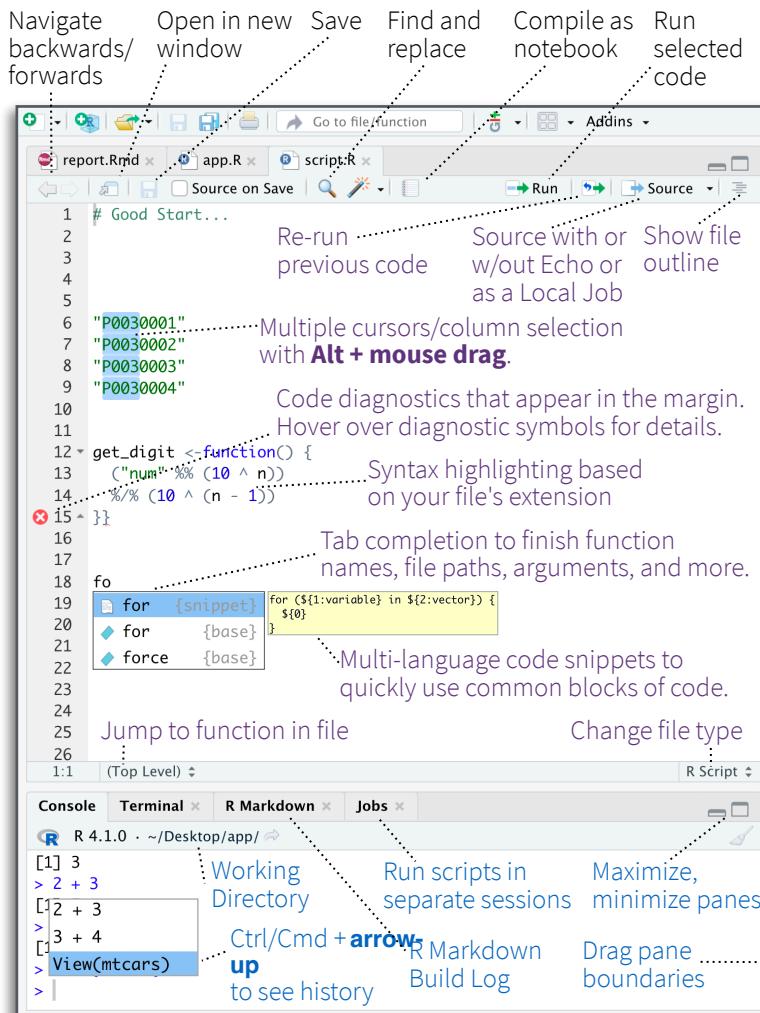
Enable roxygen documentation with **Tools > Project Options > Build Tools**

Roxygen guide at **Help > Roxygen Quick Reference**

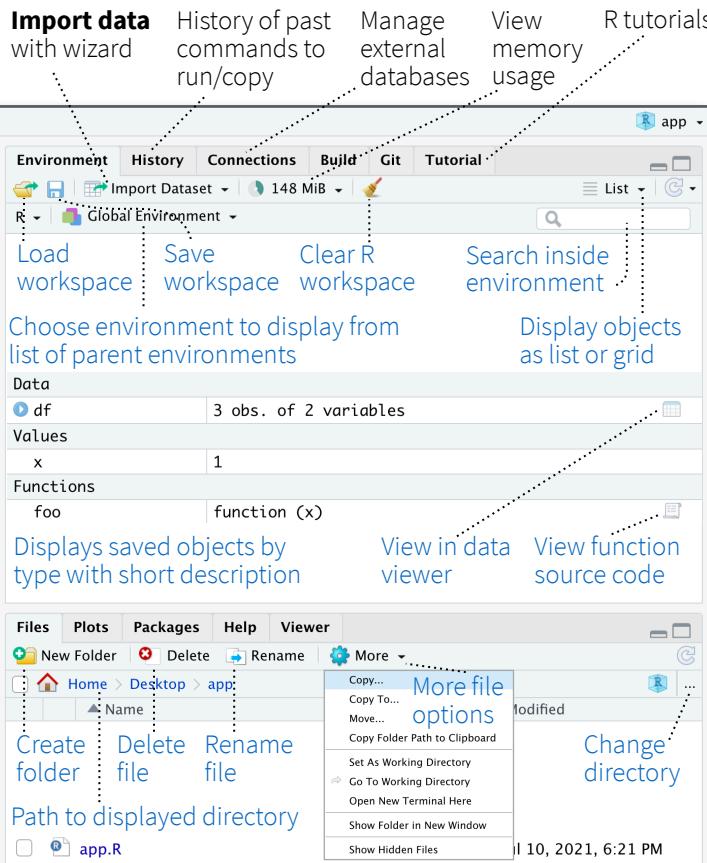
See package information in the **Build Tab**



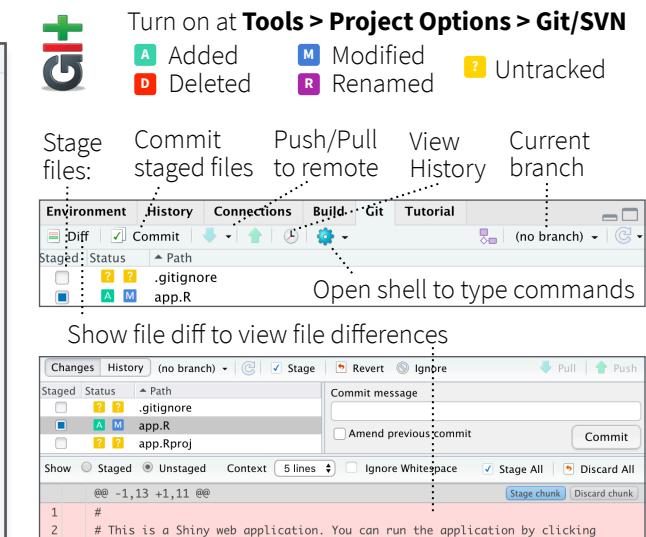
Source Editor



Tab Panes

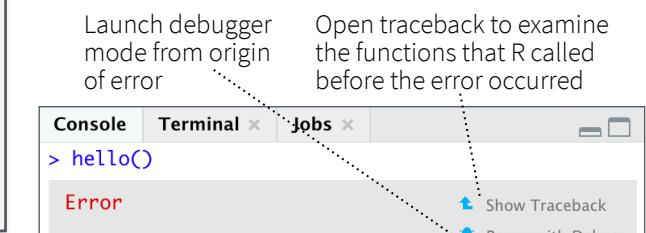


Version Control



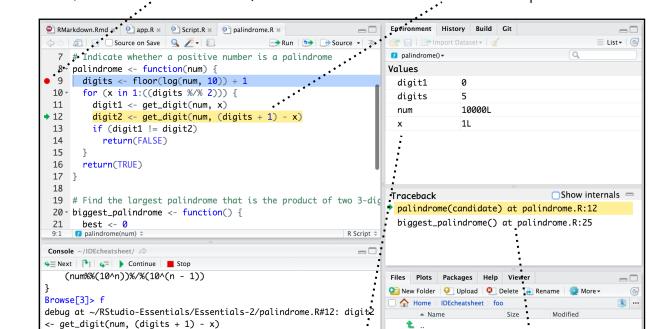
Debug Mode

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



Click next to line number to add/remove a breakpoint.

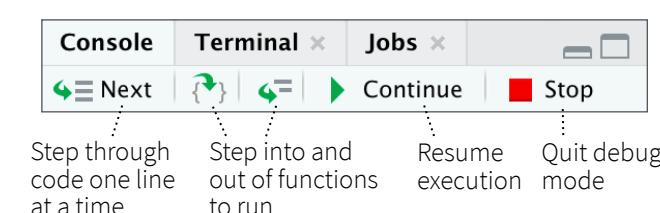
Highlighted line shows where execution has paused



Run commands in environment where execution has paused

Examine variables in executing environment

Select function in traceback to debug





Keyboard Shortcuts

RUN CODE

Search command history
Interrupt current command
Clear console

	Windows/Linux	Mac
Ctrl+arrow-up	Ctrl+arrow-up	Cmd+arrow-up
Esc	Esc	Esc
Ctrl+L	Ctrl+L	Ctrl+L

NAVIGATE CODE

Go to File/Function

Ctrl+.	Ctrl+.
--------	--------

WRITE CODE

Attempt completion

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

	Windows/Linux	Mac
Alt+-	Option+-	Cmd+Shift+M
Ctrl+Shift+M	Ctrl+Shift+M	Cmd+Shift+C
Ctrl+Shift+C	Ctrl+Shift+C	Cmd+Shift+C

MAKE PACKAGES

Load All (devtools)
Test Package (Desktop)
Document Package

	Windows/Linux	Mac
Ctrl+Shift+L	Ctrl+Shift+L	Cmd+Shift+L
Ctrl+Shift+T	Ctrl+Shift+T	Cmd+Shift+T
Ctrl+Shift+D	Ctrl+Shift+D	Cmd+Shift+D

DOCUMENTS AND APPS

Knit/Render Document (knitr)
Insert chunk (Sweave & Knitr)
Run from start to current line

Windows/Linux Mac

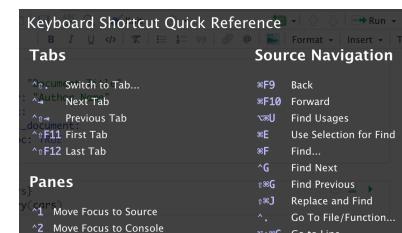
Ctrl+Shift+K	Cmd+Shift+K
Ctrl+Alt+I	Cmd+Option+I
Ctrl+Alt+B	Cmd+Option+B

MORE KEYBOARD SHORTCUTS

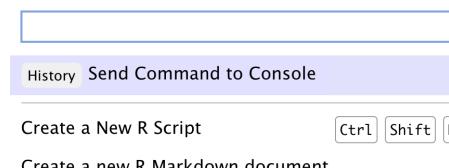
Keyboard Shortcuts Help
Show Command Palette

Alt+Shift+K	Option+Shift+K
Ctrl+Shift+P	Cmd+Shift+P

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



Search for keyboard shortcuts with **Tools > Show Command Palette** or **Ctrl/Cmd + Shift + P**.



Posit Workbench

WHY POSIT WORKBENCH?

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

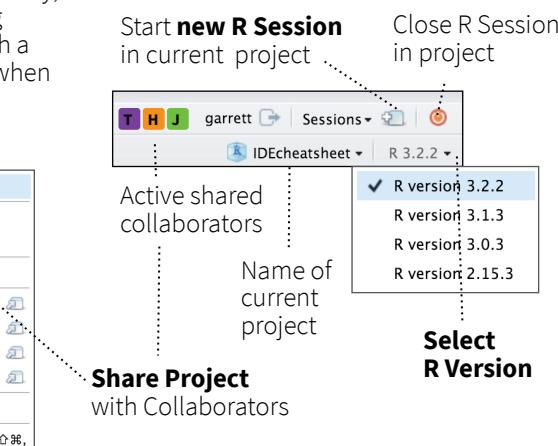
- open and run multiple R sessions at once
- tune your resources to improve performance
- administrative tools for managing user sessions
- collaborate real-time with others in shared projects
- switch easily from one version of R to a different version
- integrate with your authentication, authorization,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

Download a free 45 day evaluation at posit.co/products/enterprise/workbench/

Share Projects

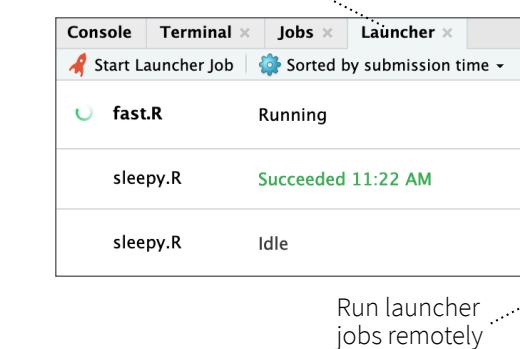
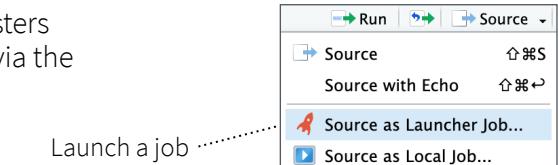
File > New Project

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



Run Remote Jobs

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



Visual Editor

The R Markdown editor interface includes the following features:

- 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

Other visible elements include the R Markdown toolbar, code editor with R code, and the R Markdown pane.

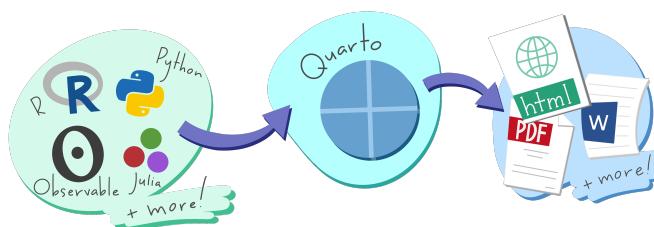
R Markdown

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

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

Jump to chunk or header

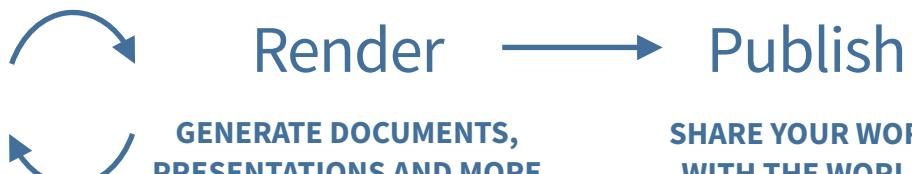
Publish and Share with Quarto :: CHEATSHEET



Author

WRITE AND CODE IN PLAIN TEXT

Author documents as .qmd files or Jupyter notebooks. Write in a rich Markdown syntax.



Publish

SHARE YOUR WORK WITH THE WORLD

Produce HTML, PDF, MS Word reveal.js, MS Powerpoint, Beamer Websites, blogs, books...

Quickly deploy to GitHub Pages, Netlify, Quarto Pub, Posit Cloud, or Posit Connect

Author

SOURCE FILE: hello.qmd

```
---  
title: "Hello, Penguins"  
format: html  
execute:  
  echo: false  
---  
  
## Meet the penguins  
  
The `penguins` data contains information about penguins from three islands in the Palmer Archipelago. The three species of penguins have quite distinct distributions of physical dimensions (@fig-penguins).  
  
```{r}  
#| label: fig-penguins
#| fig-cap: "Dimensions of penguins across three islands."
#| warning: false
library(tidyverse, quietly = TRUE)
library(palmerpenguins)
penguins >
 ggplot(aes(x = flipper_length_mm, y = bill_length_mm)) +
 geom_point(aes(color = species)) +
 scale_color_manual(
 values = c("darkorange", "purple", "cyan4")) +
```

**Set format(s) and options**  
Use YAML Syntax

**## Write with \*\*Markdown\*\***  
RStudio: Help > Markdown Quick Reference  
 Use Visual Editor

**Include code**  
R, Python, Julia, Observable, or any language with a Jupyter kernel

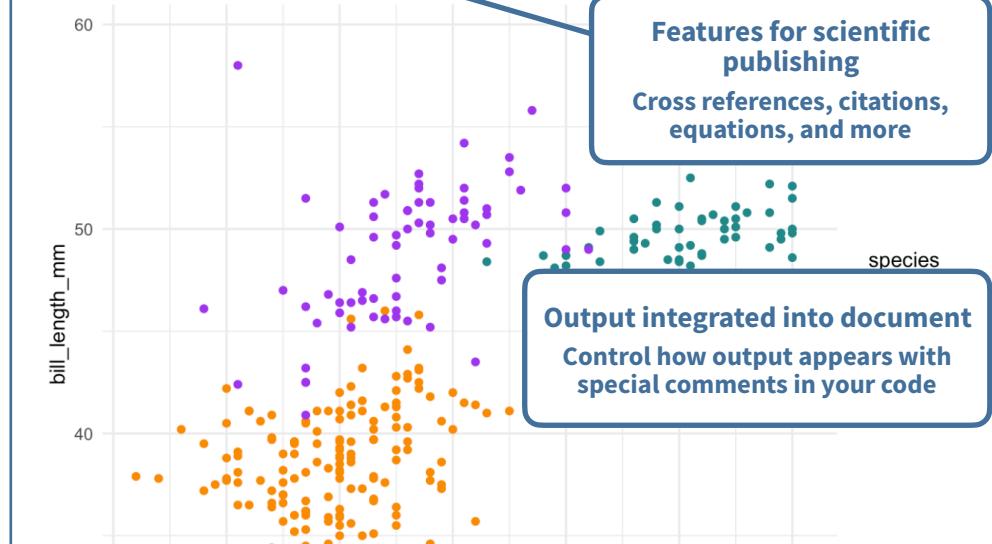
## Render

### RENDERED OUTPUT: hello.html

#### Hello, Penguins

#### Meet the penguins

The three species of penguins have quite distinct distributions of physical dimensions (Figure 1).



## USE A TOOL WITH A RICH EDITING EXPERIENCE

RStudio Positron VS Code + Quarto extension

Run code cells as you write

Render with a button or keyboard shortcut

Edit Quarto documents with a **Visual Editor**

## OR ANY TEXT EDITOR

Quarto documents (.qmd) can be edited in any tool that edits text.

Apply formatting in Visual Editor. Saved as Markdown in source.  
Insert elements like code cells, cross references, and more.

Save, then render to preview the document output.

Terminal

quarto preview hello.qmd

Use Render button

Use Preview button

The resulting HTML/PDF/MS Word/etc. document will be created and saved in the same directory as the source .qmd file.

## BEHIND THE SCENES

When you render a document, Quarto:

- Runs the code and embeds results and text into an .md file with: **Knitr**, if any {r} cells or, **Jupyter**, if any other cells.
- Converts the .md file into the output format with Pandoc.

## GET QUARTO

<https://quarto.org/docs/download/>

Or use version **bundled with Positron or RStudio**

## GET STARTED

<https://quarto.org/docs/get-started/>

## Publish

### Terminal

quarto publish {venue} hello.qmd

{venue}: quarto-pub, connect, gh-pages, netlify, confluence

Quarto Pub

Free publishing service for Quarto content.

posit Connect

Org-hosted, control access, schedule updates.

Use Publish button

Use Posit Publisher extension

## Quarto Projects

### CREATE WEBSITES, BOOKS, AND MORE

A directory of Quarto documents + a configuration file (\_quarto.yml)

See examples at <https://quarto.org/docs/gallery/>

Get started from the command line:

### Terminal

quarto create project {type}

{type}: default, website, blog, book, confluence, manuscript

Use File > New Project

Use Quarto: Create Project command

Artwork from "Hello, Quarto" keynote by Julia Lowndes and Mine Çetinkaya-Rundel, presented at RStudio Conference 2022. Illustrated by Allison Horst.

# Include Code

## CODE CELLS

Code cells start with `{{language}}` and end with `}`.

Use **Insert Code Chunk/Cell**

```
```{r}
#| label: chunk-id
library(tidyverse)
```
```{python}
#| label: chunk-id
import pandas as pd
```
```

Other languages: {julia}, {ojs}

Add code cell options with #| comments.

Cell options control **execution**, figures, tables, layout and more. See them all at: <https://quarto.org/docs/reference/cells>

## EXECUTION OPTIONS

### OPTION DEFAULT EFFECTS

|                |       |                                                                     |
|----------------|-------|---------------------------------------------------------------------|
| <b>echo</b>    | true  | false: hide code<br>fenced: include code cell syntax                |
| <b>eval</b>    | true  | false: don't run code                                               |
| <b>include</b> | true  | false: don't include code or results                                |
| <b>output</b>  | true  | false: don't include results<br>asis: treat results as raw markdown |
| <b>warning</b> | true  | false: don't include warnings in output                             |
| <b>error</b>   | false | true: include error in output and continue with render              |

Set execution options at the **cell level**:

```
```{r}
#| echo: false
```
```{python}
#| echo: false
```
```
```

Or, **globally** in the YAML header with the **execute** option:

```
---
```

execute:
 echo: false

```
---
```

Set options in code cells with #| comments and YAML syntax:
key: value

INLINE CODE

Use computed values directly in text sections.
Code is evaluated at render and results appear as text.

KNITR JUPYTER OUTPUT

Value is `r 2 + 2`. Value is `{python} 2 + 2`. Value is 4.

Set Format and Options

SET FORMAT OPTIONS

```
---
```

title: "My Document"
format:
 html:
 code-fold: true
 toc: true

Indent options 4 spaces

Indent format 2 spaces

MULTIPLE FORMATS

```
---
```

title: "My Document"
toc: true
format:
 html:
 code-fold: true
 pdf: default

Top-level options apply to all formats

Common formats: **html, pdf, docx, odt, rtf, gfm, pptx, revealjs, beamer**

Render **all** formats:

Terminal
quarto render hello.qmd

Render a **specific** format:

Terminal
quarto render hello.qmd --to pdf

OPTION

	html/revealjs pdf/beamer docx/pptx	DESCRIPTION
Nav	toc	X X X Add a table of contents (true or false)
	toc-depth	X X X Lowest level of headings to add to table of contents (e.g. 2, 3)
	anchor-sections	X Show section anchors on mouse hover (true or false)
Style	highlight-style	X X X Syntax highlighting theme (e.g. arrow, pygments, kate, zenburn)
	mainfont, monofont	X X Font name. HTML: sets CSS font-family; LaTeX: via fonts package
	theme	X Bootswatch theme name (e.g. cosmo, darkly, solar etc.)
	css	X CSS or SCSS file to use to style the document (e.g. "style.css")
	reference-doc	X docx/pptx file containing template styles (e.g. file.docx, file.pptx)

include-in-header	X X	Files of content to include in header of output document, also include-before-body, include-after-body
keep-md	X X X	Keep intermediate markdown (true or false), also keep-ipynb, keep-tex

LaTeX	documentclass	X LaTeX document class, set document class options with classoption
	pdf-engine	X LaTeX engine to produce PDF output (xelatex, pdflatex, lualatex)
	cite-method	X Method used to format citations (citeproc, natbib, biblatex)

Code	code-fold	X Let readers toggle the display of R code (false, true, or show)
	code-tools	X Add menu for hiding, showing, and downloading code (true or false)
	code-overflow	X Display of wide code (scroll, or wrap)

Figures	fig-align	X X / Alignment of figures (default, left, right, or center)
	fig-width, fig-height	X X X Default width and height for figures in inches
	fig-format	X X X Format for Matplotlib or R figures (retina, png, jpeg, svg, or pdf)

Visit <https://quarto.org/docs/reference/> to see **all options** by format

Also use in code cells

✓
✓
✓
✓ Knitr

Add Content

FIGURES



MARKDOWN

```
![CAP](image.png){#fig-LABEL fig-alt="ALT"}
```

COMPUTATION

```
```{python}
#| label: fig-LABEL
#| fig-cap: CAP
#| fig-alt: ALT
{ plot code here }
```
```

Or {r}

CROSS REFERENCES

1. Add labels

Code cell: add option **label: prefix-LABEL**
Markdown: add attribute **#prefix-LABEL**

2. Add references @prefix-LABEL, e.g.

You can see in @fig-scatterplot,
that...

| Prefix | Renders | Prefix | Renders |
|--------|----------|--------|------------|
| fig- | Figure 1 | eq- | Equation 1 |
| tbl- | Table 1 | sec- | Section 1 |

TABLES

MARKDOWN

| object | radius |
|--------|----------------------|
| --- | --- |
| Sun | 696000 |
| Earth | 6371 |
| : | CAPTION {#tbl-LABEL} |

COMPUTATION

```
object	radius
Sun	696000
Earth	6371
: CAPTION {#tbl-LABEL}
```

Knitr

Use knitr::kable() to produce Markdown:

```
```{r}
#| label: tbl-LABEL
#| tbl-cap: CAPTION
import pandas as pd, tabulate
from IPython.display import Markdown
df = pd.DataFrame({ "A": [1, 2],
 "B": [1, 2] })
Markdown(df.to_markdown(index=False))
```
```

Also see the R packages: gt, flextable, kableExtra.

CITATIONS

1. Add a bibliography file to the YAML header:

```
---
bibliography: references.bib
---
```

2. Add citations: **@citation**, or **@citation**

Visual Editor

Build your bibliography file from your Zotero library, DOI, Crossref, DataCite, or PubMed

CALLOUTS

Instead of **tip** use one of:

note, caution, warning, or important.

note

warning

caution

important

SHORTCODES

```
{< include _file.qmd >}
{{< embed file.ipynb#id >}}
{{< video video.mp4 >}}
```

rmarkdown :: CHEATSHEET

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"



The screenshot shows the RStudio IDE with the Source Editor open. A numbered callout path highlights the workflow: 1. New File (File menu), 2. Embed Code (Knit on Save button), 3. Write Text (Visual tab), 4. Set Output Format(s) and Options (YAML header), 5. Save and Render (Knit button), and 6. Share (Publish dropdown).

The screenshot shows the RStudio IDE with the Visual Editor tab selected. A numbered callout path highlights the workflow: 1. New File (File menu), 2. Embed Code (Knit on Save button), 3. Write Text (Visual tab), 4. Set Output Format(s) and Options (YAML header), 5. Save and Render (Knit button), and 6. Share (Publish dropdown).

The screenshot shows the RStudio IDE with the Render tab selected. A numbered callout path highlights the workflow: 1. New File (File menu), 2. Embed Code (Knit on Save button), 3. Write Text (Visual tab), 4. Set Output Format(s) and Options (YAML header), 5. Save and Render (Knit button), and 6. Share (Publish dropdown).

Insert Citations

Create citations from a bibliography file, a Zotero library, or from DOI references.

BUILD YOUR BIBLIOGRAPHY

- Add BibTeX or CSL bibliographies to the YAML header.

```
---
title: "My Document"
bibliography: references.bib
link-citations: TRUE
---
```

- If Zotero is installed locally, your main library will automatically be available.
- Add citations by DOI by searching "from DOI" in the **Insert Citation** dialog.

INSERT CITATIONS

- Access the **Insert Citations** dialog in the Visual Editor by clicking the @ symbol in the toolbar or by clicking **Insert > Citation**.
- Add citations with markdown syntax by typing **[@cite]** or **@cite**.

Insert Tables

Output data frames as tables using **kable**(data, caption).

```
```{r}
data <- faithful[1:4,]
knitr::kable(data,
 caption = "Table with kable")
```
```

Other table packages include **flextable**, **gt**, and **kableExtra**.



Write with Markdown

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

Plain text.

Plain text.

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

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

Also end with a backslash\ to make a new line.

Also end with a backslash\ to make a new line.

italics* and ****bold****

italics and **bold**

superscript²/**subscript**₂

superscript²/**subscript**₂

~~strikethrough~~

strikethrough

escaped: `* _ _`

escaped: `* _ _`

endash: `--`, **emdash:** `---`

endash: `--`, **emdash:** `---`

Header 1

Header 2

...

Header 6

```
# Header 1
# Header 2
...
##### Header 6
```

- unordered list

• item 2
 - item 2a (indent 1 tab)
 - item 2b

1. ordered list

1. item 2
 - item 2a (indent 1 tab)
 - item 2b

2. item 2
 - item 2a (indent 1 tab)

2. item 2
 - item 2a (indent 1 tab)
 - item 2b

<link url>

<http://www.posit.co/>

[This is a link.](link url)

This is a link.

[This is another link][id].

This is another link.

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

block quotes

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

equation block:

`$\$E = mc^2\$`

$E = mc^2$

horizontal rule:
 `---`

horizontal rule:

| Right | Left | Default | Center |

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

| 12 | 12 | 12 | 12 |

| 123 | 123 | 123 | 123 |

| 1 | 1 | 1 | 1 |

| 1 | 1 | 1 | 1 |

multiple lines of verbatim code

block quotes

### HTML Tabsets

`## Results {tabset}`

`## Plots`

`text`

`## Tables`

`more text`



# 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`.

IMPORTANT OPTIONS	DESCRIPTION	HTML	PDF	MS Word	MS PPT
anchor_sections	Show section anchors on mouse hover (TRUE or FALSE)	X			
citation_package	The LaTeX package to process citations ("default", "natbib", "biblatex")	X			
code_download	Give readers an option to download the .Rmd source code (TRUE or FALSE)	X			
code_folding	Let readers to toggle the display of R code ("none", "hide", or "show")	X			
css	CSS or SCSS file to use to style document (e.g. "style.css")	X			
dev	Graphics device to use for figure output (e.g. "png", "pdf")	X X			
df_print	Method for printing data frames ("default", "kable", "tibble", "paged")	X X X X			
fig_caption	Should figures be rendered with captions (TRUE or FALSE)	X X X X			
highlight	Syntax highlighting ("tango", "pygments", "kate", "zenburn", "textmate")	X X X			
includes	File of content to place in doc ("in_header", "before_body", "after_body")	X X			
keep_md	Keep the Markdown .md file generated by knitting (TRUE or FALSE)	X X X X			
keep_tex	Keep the intermediate TEX file used to convert to PDF (TRUE or FALSE)	X			
latex_engine	LaTeX engine for producing PDF output ("pdflatex", "xelatex", or "lualatex")	X			
reference_docx/_doc	docx/pptx file containing styles to copy in the output (e.g. "file.docx", "file.pptx")	X X			
theme	Theme options (see Bootswatch and Custom Themes below)	X			
toc	Add a table of contents at start of document (TRUE or FALSE)	X X X X			
toc_depth	The lowest level of headings to add to table of contents (e.g. 2, 3)	X X X X			
toc_float	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

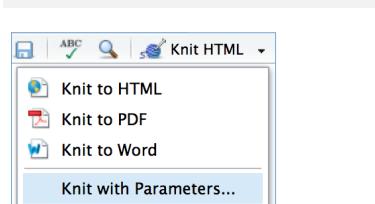
### PARAMETERS

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

1. **Add parameters** in the header as sub-values of `params`.
2. **Call parameters** in code using `params$<name>`.
3. **Set parameters** with Knit with Parameters or the `params` argument of `render()`.

### REUSABLE TEMPLATES

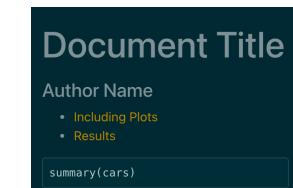
1. **Create a new package** with a `inst/rmarkdown/templates` directory.
2. **Add a folder** containing `template.yaml` (below) and `skeleton.Rmd` (template contents).
3. **Install** the package to access template by going to **File > New R Markdown > From Template**.



### 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:
 bootswatch: 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.

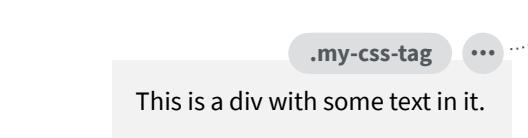
Bracketed Span  
A [green]{.my-color} word.

A green word.

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

All of these words  
are green.

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



This is a div with some text in it.

## 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 Posit Connect

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

[posit.co/products/enterprise/connect](https://posit.co/products/enterprise/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?

speed	dist
1	4.00
2	4.00
3	7.00
4	7.00
5	8.00

Also see Shiny Prerendered for better performance. rmarkdown.rstudio.com/authoring_shiny_prerendered.

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

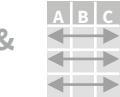
Data transformation with dplyr :: CHEATSHEET



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**

pipes

$x |> f(y)$ becomes $f(x, y)$

Summarize 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).

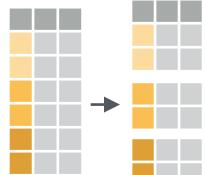


summarize(.data, ...)
Compute table of summaries.
`mtcars |> summarize(avg = mean(mpg))`

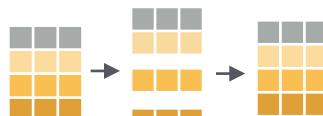
count(.data, ..., wt = NULL, sort = FALSE, name = NULL) Count number of rows in each group defined by the variables in ... Also **tally()**, **add_count()**, **add_tally()**.
`mtcars |> count(cyl)`

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) |> summarize(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.
`g_mtcars <- mtcars |> group_by(cyl)`
`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.
`mtcars |> filter(mpg > 20)`



distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values.
`mtcars |> distinct(gear)`



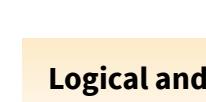
slice(.data, ..., .preserve = FALSE) Select rows by position.
`mtcars |> slice(10:15)`



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.
`mtcars |> slice_sample(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.
`mtcars |> slice_min(mpg, prop = 0.25)`



slice_head(.data, ..., n, prop) and **slice_tail()** Select the first or last rows.
`mtcars |> slice_head(n = 5)`

Logical and boolean operators to use with filter()

<code>==</code>	<code><</code>	<code><=</code>	<code>is.na()</code>	<code>%in%</code>	<code> </code>	<code>xor()</code>
<code>!=</code>	<code>></code>	<code>>=</code>	<code>!is.na()</code>	<code>!</code>	<code>&</code>	

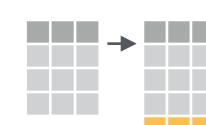
See [?base::Logic](#) and [?Comparison](#) for help.

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.
`mtcars |> arrange(mpg)`
`mtcars |> arrange(desc(mpg))`

ADD CASES



add_row(.data, ..., .before = NULL, .after = NULL) Add one or more rows to a table.
`cars |> add_row(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.
`mtcars |> pull(wt)`



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



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

Use these helpers with select() and across()

e.g. `mtcars |> select(mpg:cyl)`

contains(match)

ends_with(match)

starts_with(match)

num_range(prefix, range)

all_of(x)/any_of(x, ..., vars)

matches(match)

; e.g., `mpg:cyl`

!, e.g., `!gear`

everything()

MANIPULATE MULTIPLE VARIABLES AT ONCE

`df <- tibble(x_1 = c(1, 2), x_2 = c(3, 4), y = c(4, 5))`



across(.cols, .funs, ..., .names = NULL) Summarize or mutate multiple columns in the same way.
`df |> summarize(across(everything()), mean))`

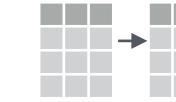


c_across(.cols) Compute across columns in row-wise data.
`df |> rowwise() |> mutate(x_total = sum(c_across(1:2)))`

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).

vectorized function



mutate(.data, ..., .keep = "all", .before = NULL, .after = NULL) Compute new column(s). Also **add_column()**.
`mtcars |> mutate(gpm = 1 / mpg)`
`mtcars |> mutate(gpm = 1 / mpg, .keep = "none")`



rename(.data, ...) Rename columns. Use **rename_with()** to rename with a function.
`mtcars |> rename(miles_per_gallon = mpg)`



Vectorized Functions

TO USE WITH MUTATE ()

mutate() applies 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()
cummax() - cumulative max()
dplyr::cummean() - cumulative mean()
cummin() - cumulative min()
cumprod() - cumulative prod()
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 SUMMARIZE ()

summarize() 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-NAs

POSITION

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

LOGICAL

mean() - proportion of TRUEs
sum() - # of TRUEs

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.

A B → C A B tibble::rownames_to_column()
1 a t → 1 a Move row names into col.
2 b u → 2 b a <- mtcars |>
3 c v → 3 c rownames_to_column(var = "C")

A B C → A B tibble::column_to_rownames()
1 a t → 1 a Move col into row names.
2 b u → 2 b a |> column_to_rownames(var = "C")
3 c v → 3 c

Also tibble::has_rownames() and tibble::remove_rownames().

Combine Tables

COMBINE VARIABLES

X	y
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.

A B C D	left_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ..., keep = FALSE, na_matches = "na")
a t 1 3	Join matching values from y to x.
b u 2 2	
c v 3 NA	

A B C D	right_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ..., keep = FALSE, na_matches = "na")
a t 1 3	Join matching values from x to y.
b u 2 2	
d w NA 1	

A B C D	inner_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ..., keep = FALSE, na_matches = "na")
a t 1 3	Join data. Retain only rows with matches.
b u 2 2	

A B C D	full_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ..., keep = FALSE, na_matches = "na")
a t 1 3	Join data. Retain all values, all rows.
b u 2 2	
c v 3 NA 1	

COLUMN MATCHING FOR JOINS

A B x C B y D	Use by = c("col1", "col2", ...) to specify one or more common columns to match on.
a t 1 t 3	left_join(x, y, by = "A")
b u 2 u 2	
c v 3 NA NA	

A x B . x C A y B y	Use a named vector, by = c("col1" = "col2") , to match on columns that have different names in each table.
a t 1 d w	left_join(x, y, by = c("C" = "D"))
b u 2 b u	
c v 3 a t	

A1 B1 C A2 B2	Use suffix to specify the suffix to give to unmatched columns that have the same name in both tables.
a t 1 d w	left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))
b u 2 b u	
c v 3 a t	

COMBINE CASES

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

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).

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

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.

A B C	semi_join(x, y, by = NULL, copy = FALSE, ..., na_matches = "na")
a t 1	
b u 2	

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.

A B C	y
a t 1	<tibble [1x2]>
b u 2	<tibble [1x2]>
c v 3	<tibble [1x2]>

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

A B C
c v 3

intersect(x, y, ...)

Rows that appear in both x and y.



A B C
a t 1
b u 2

setdiff(x, y, ...)

Rows that appear in x but not y.



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



Data import with the tidyverse :: CHEATSHEET

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
A	B	C
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
A	B	C
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
A	B	C
1.5	2	3
4.5	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
A	B	C
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)`

x	y	z
A	B	C
1	2	3
4	5	NA

Provide header

`read_csv("file.csv", col_names = c("x", "y", "z"))`

A	B	C
NA	2	3
4	5	NA

Read multiple files into a single table

`read_csv(c("f1.csv", "f2.csv", "f3.csv"), id = "origin_file")`

A;B;C	1,5;2;3;0	
A	B	C
1	5	NA
2	3	0

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

A	B	C
NA	2	3
4	5	NA

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	
A	B	C
1	5	NA
2	3	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

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(),
#   edu = col_character(),
#   earn = col_double()
# )
```

age is an integer

edu is a character

earn is a double (numeric)

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	
3	y	7		9	10

```
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
A	B	C	D	E
A	B	C	D	E

- To **read multiple sheets**:
1. Get a vector of sheet names from the file path.
 2. Set the vector names to be the sheet names.
 3. Use purrr::map() and purrr::list_rbind() to read multiple files into one data frame.

```
path <- "your_file_path.xlsx"  
path >  
  excel_sheets() |>  
  set_names() |>  
  map(read_excel, path = path) |>  
  list_rbind()
```

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	
3	y	7		9	10

```
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().
```

SHEETS METADATA

URLs are in the form:
<https://docs.google.com/spreadsheets/d/>
SPREADSHEET_ID/edit#gid=**SHEET_ID**

gs4_get(ss) Get spreadsheet meta data.

gs4_find(...) Get data on all spreadsheet files.

sheet_properties(ss) Get a tibble of properties
for each worksheet. Also **sheet_names()**.

WRITE SHEETS

1	x	4
2	y	5
3	z	6

→

A	B	C
1	1	x
2	2	y
3	3	z

s1

write_sheet(data, ss = NULL, sheet = NULL)
Write a data frame into a new or existing Sheet.

gs4_create(name, ..., sheets = NULL) Create a new Sheet with a vector of names, a data frame, or a (named) list of data frames.

sheet_append(ss, data, sheet = 1) Add rows to the end of a worksheet.

1	A	B	C
2			

→

A	B	C
1	x1	x2
2	y	5
3	z	6

s1

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

GOOGLESHEETS4 COLUMN SPECIFICATION

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

Use the **col_types** argument of **read_sheet()**/**range_read()** 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_sheet(path, col_types = "c")

Set each column individually

col types: skip, guess, integer, logical, character
read_sheets(ss, col_types = "?ilc")

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 - "l"
- 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 **tidyxl** 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 to read more.

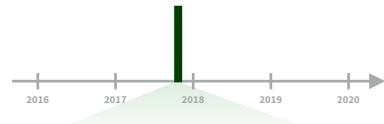
For whole-file operations (e.g. renaming, sharing, placing within a folder), see the tidyverse package **googledrive** at googledrive.tidyverse.org.



Dates and times with lubridate :: CHEATSHEET



Date-times



2017-11-28 12:00:00

2017-11-28 12:00:00

A **date-time** is a point on the timeline, stored as the number of seconds since 1970-01-01 00:00:00 UTC

```
dt <- as_datetime(1511870400)
## "2017-11-28 12:00:00 UTC"
```

PARSE DATE-TIMES (Convert strings or numbers to date-times)

1. Identify the order of the year (**y**), month (**m**), day (**d**), hour (**h**), minute (**m**) and second (**s**) elements in your data.
2. Use the function below whose name replicates the order. Each accepts a tz argument to set the time zone, e.g. ymd(x, tz = "UTC").

2017-11-28T14:02:00

ymd_hms(), ymd_hm(), ymd_h().
ymd_hms("2017-11-28T14:02:00")

2017-22-12 10:00:00

ydm_hms(), ydm_hm(), ydm_h().
ydm_hms("2017-22-12 10:00:00")

11/28/2017 1:02:03

mdy_hms(), mdy_hm(), mdy_h().
mdy_hms("11/28/2017 1:02:03")

1 Jan 2017 23:59:59

dmy_hms(), dmy_hm(), dmy_h().
dmy_hms("1 Jan 2017 23:59:59")

20170131

ymd(), ydm(). ymd(20170131)

July 4th, 2000

mdy(), myd(). mdy("July 4th, 2000")

4th of July '99

dmy(), dym(). dmy("4th of July '99")

2001: Q3

yq() Q for quarter. yq("2001: Q3")

07-2020

my(), ym(). my("07-2020")

2:01

hms::hms() Also lubridate::hms(), hm() and ms(), which return periods.* hms::hms(seconds = 0, minutes = 1, hours = 2)

2017.5

date_decimal(decimal, tz = "UTC")
date_decimal(2017.5)

now(zone = "") Current time in tz (defaults to system tz). now()

today(zone = "") Current date in a tz (defaults to system tz). today()

fast.strptime() Faster strftime.

fast.strptime("9/1/01", "%y/%m/%d")

parse_date_time() Easier strftime.

parse_date_time("09-01-01", "ymd")



2017-11-28

A **date** is a day stored as the number of days since 1970-01-01

```
d <- as_date(17498)
## "2017-11-28"
```

GET AND SET COMPONENTS

Use an accessor function to get a component. Assign into an accessor function to change a component in place.

2018-01-31 11:59:59

date(x) Date component. date(dt)

year(x) Year. year(dt)
isoyear(x) The ISO 8601 year.
epiyear(x) Epidemiological year.

2018-01-31 11:59:59

month(x, label, abbr) Month. month(dt)

2018-01-31 11:59:59

day(x) Day of month. day(dt)
wday(x, label, abbr) Day of week.
qday(x) Day of quarter.

2018-01-31 11:59:59

hour(x) Hour. hour(dt)

2018-01-31 11:59:59

minute(x) Minutes. minute(dt)

2018-01-31 11:59:59

second(x) Seconds. second(dt)

2018-01-31 11:59:59 UTC

tz(x) Time zone. tz(dt)

2018-01-31 11:59:59

week(x) Week of the year. week(dt)
isoweek() ISO 8601 week.
epiweek() Epidemiological week.

2018-01-31 11:59:59

quarter(x) Quarter. quarter(dt)

2018-01-31 11:59:59

semester(x, with_year = FALSE) Semester. semester(dt)

2018-01-31 11:59:59

am(x) Is it in the am? am(dt)
pm(x) Is it in the pm? pm(dt)

2018-01-31 11:59:59

dst(x) Is it daylight savings? dst(dt)

2018-01-31 11:59:59

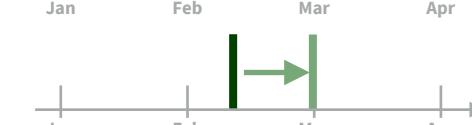
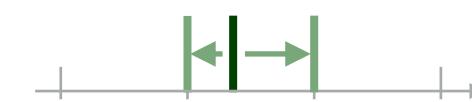
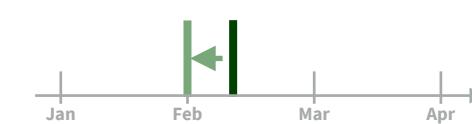
leap_year(x) Is it a leap year?
leap_year(dt)

2018-01-31 11:59:59

update(object, ..., simple = FALSE)
update(dt, mday = 2, hour = 1)



Round Date-times



Valid units are second, minute, hour, day, week, month, bimonth, quarter, season, halfyear and year.

rollback(dates, roll_to_first = FALSE, preserve_hms = TRUE) Roll back to last day of previous month. Also **rollforward()**. rollback(dt)

Stamp Date-times

stamp() Derive a template from an example string and return a new function that will apply the template to date-times. Also **stamp_date()** and **stamp_time()**.

1. Derive a template, create a function
sf <- stamp("Created Sunday, Jan 17, 1999 3:34")

2. Apply the template to dates
sf(ymd("2010-04-05"))
[1] "Created Monday, Apr 05, 2010 00:00"

Tip: use a date with day > 12

Time Zones

R recognizes ~600 time zones. Each encodes the time zone, Daylight Savings Time, and historical calendar variations for an area. R assigns one time zone per vector.

Use the **UTC** time zone to avoid Daylight Savings.

OlsonNames() Returns a list of valid time zone names. OlsonNames()

Sys.timezone() Gets current time zone.

5:00 Mountain 6:00 Central
4:00 Pacific 7:00 Eastern

PT MT CT ET

7:00 Pacific 7:00 Mountain
7:00 Central

with_tz(time, tzzone = "") Get the same date-time in a new time zone (a new clock time). Also **local_time(dt, tz, units)**. with_tz(dt, "US/Pacific")

force_tz(time, tzzone = "") Get the same clock time in a new time zone (a new date-time). Also **force_tzs()**. force_tz(dt, "US/Pacific")



Math with Date-times

Math with date-times relies on the **timeline**, which behaves inconsistently. Consider how the timeline behaves during:

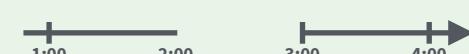
A normal day

```
nor <- ymd_hms("2018-01-01 01:30:00", tz="US/Eastern")
```



The start of daylight savings (spring forward)

```
gap <- ymd_hms("2018-03-11 01:30:00", tz="US/Eastern")
```



The end of daylight savings (fall back)

```
lap <- ymd_hms("2018-11-04 00:30:00", tz="US/Eastern")
```



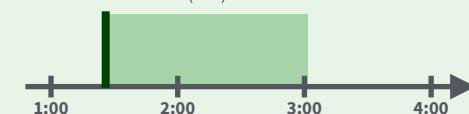
Leap years and leap seconds

```
leap <- ymd("2019-03-01")
```

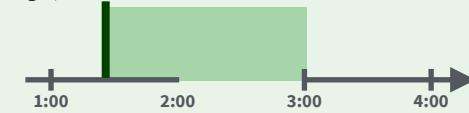


Periods track changes in clock times, which ignore time line irregularities.

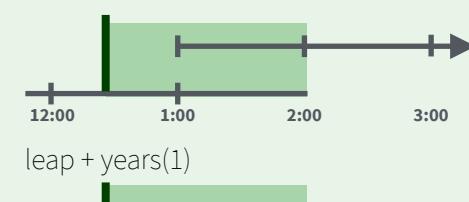
nor + minutes(90)



gap + minutes(90)



lap + minutes(90)

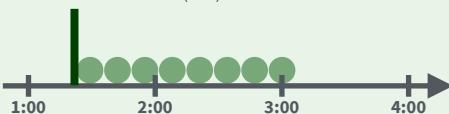


leap + years(1)



Durations track the passage of physical time, which deviates from clock time when irregularities occur.

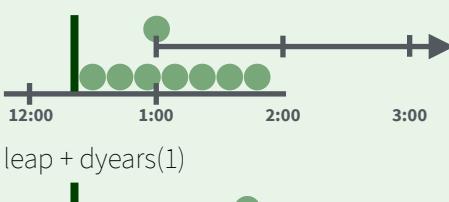
nor + dminutes(90)



gap + dminutes(90)



lap + dminutes(90)



leap + dyears(1)



Intervals represent specific intervals of the timeline, bounded by start and end date-times.

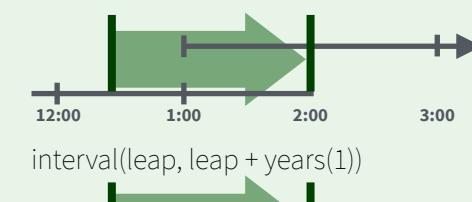
interval(nor, nor + minutes(90))



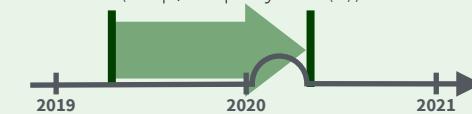
interval(gap, gap + minutes(90))



interval(lap, lap + minutes(90))



interval(leap, leap + years(1))



Not all years are 365 days due to **leap days**.

Not all minutes are 60 seconds due to **leap seconds**.

It is possible to create an imaginary date by adding **months**, e.g. February 31st

```
jan31 <- ymd(20180131)
```

```
jan31 + months(1)
```

```
## NA
```

%m+% and %m-% will roll imaginary dates to the last day of the previous month.

```
jan31 %m+% months(1)
```

```
## "2018-02-28"
```

add_with_rollback(e1, e2, roll_to_first = TRUE) will roll imaginary dates to the first day of the new month.

```
add_with_rollback(jan31, months(1), roll_to_first = TRUE)
```

```
## "2018-03-01"
```

PERIODS

Add or subtract periods to model events that happen at specific clock times, like the NYSE opening bell.

Make a period with the name of a time unit **pluralized**, e.g.

```
p <- months(3) + days(12)
```

```
p
```

```
"3m 12d 0H 0M 0S"
```

Number of months Number of days etc.

years(x = 1) x years.

months(x) x months.

weeks(x = 1) x weeks.

days(x = 1) x days.

hours(x = 1) x hours.

minutes(x = 1) x minutes.

seconds(x = 1) x seconds.

milliseconds(x = 1) x milliseconds.

microseconds(x = 1) x microseconds.

nanoseconds(x = 1) x nanoseconds.

picoseconds(x = 1) x picoseconds.

period(num = NULL, units = "second", ...)

An automation friendly period constructor.
period(5, unit = "years")

as.period(x, unit) Coerce a timespan to a period, optionally in the specified units. Also **is.period()**. as.period(p)

period_to_seconds(x) Convert a period to the "standard" number of seconds implied by the period. Also **seconds_to_period()**. period_to_seconds(p)

DURATIONS

Add or subtract durations to model physical processes, like battery life. Durations are stored as seconds, the only time unit with a consistent length.

Diftimes are a class of durations found in base R.

Make a duration with the name of a period prefixed with a **d**, e.g.

```
dd <- ddays(14)
```

```
dd
```

```
"1209600s (~2 weeks)"
```

Exact length in seconds Equivalent in common units

dyears(x = 1) 31536000x seconds.

dmonths(x = 1) 2629800x seconds.

dweeks(x = 1) 604800x seconds.

ddays(x = 1) 86400x seconds.

dhours(x = 1) 3600x seconds.

dminutes(x = 1) 60x seconds.

dseconds(x = 1) x seconds.

dmilliseconds(x = 1) x × 10⁻³ seconds.

dmicroseconds(x = 1) x × 10⁻⁶ seconds.

dnanoseconds(x = 1) x × 10⁻⁹ seconds.

dpicoseconds(x = 1) x × 10⁻¹² seconds.

duration(num = NULL, units = "second", ...)

An automation friendly duration constructor. duration(5, unit = "years")

as.duration(x, ...) Coerce a timespan to a duration. Also **is.duration()**, **is.difftime()**. as.duration(i)

make_difftime(x) Make difftime with the specified number of units. make_difftime(99999)

INTERVALS

Divide an interval by a duration to determine its physical length, divide an interval by a period to determine its implied length in clock time.

Make an interval with **interval()** or %--%, e.g.

```
i <- interval(ymd("2017-01-01"), d)
```

```
## 2017-01-01 UTC--2017-11-28 UTC
```

```
j <- d %--% ymd("2017-12-31")
```

```
## 2017-11-28 UTC--2017-12-31 UTC
```

- a %within% b Does interval or date-time a fall within interval b? now() %within% i
- int_start(int)** Access/set the start date-time of an interval. Also **int_end()**. int_start(i) <- now(); int_start(i)
- int_aligns(int1, int2)** Do two intervals share a boundary? Also **int_overlaps()**. int_aligns(i, j)
- int_diff(times)** Make the intervals that occur between the date-times in a vector. v <- c(dt, dt + 100, dt + 1000); int_diff(v)
- int_flip(int)** Reverse the direction of an interval. Also **int_standardize()**. int_flip(i)
- int_length(int)** Length in seconds. int_length(i)
- int_shift(int, by)** Shifts an interval up or down the timeline by a timespan. int_shift(i, days(-1))
- as.interval(x, start, ...)** Coerce a timespan to an interval with the start date-time. Also **is.interval()**. as.interval(days(1), start = now())

String manipulation with stringr :: CHEATSHEET



The **stringr** package provides a set of internally consistent tools for working with character strings, i.e. sequences of characters surrounded by quotation marks.

Detect Matches

	str_detect(string, pattern, negate = FALSE) Detect the presence of a pattern match in a string. Also str_like() . str_detect(fruit, "a")
	str_starts(string, pattern, negate = FALSE) Detect the presence of a pattern match at the beginning of a string. Also str_ends() . str_starts(fruit, "a")
	str_which(string, pattern, negate = FALSE) Find the indexes of strings that contain a pattern match. str_which(fruit, "a")
	str_locate(string, pattern) Locate the positions of pattern matches in a string. Also str_locate_all() . str_locate(fruit, "a")
	str_count(string, pattern) Count the number of matches in a string. str_count(fruit, "a")

Subset Strings

	str_sub(string, start = 1L, end = -1L) Extract substrings from a character vector. str_sub(fruit, 1, 3); str_sub(fruit, -2)
	str_subset(string, pattern, negate = FALSE) Return only the strings that contain a pattern match. str_subset(fruit, "p")
	str_extract(string, pattern) Return the first pattern match found in each string, as a vector. Also str_extract_all() to return every pattern match. str_extract(fruit, "[aeiou]")
	str_match(string, pattern) Return the first pattern match found in each string, as a matrix with a column for each () group in pattern. Also str_match_all() . str_match(sentences, "(a the) ([^ +])")

Manage Lengths

	str_length(string) The width of strings (i.e. number of code points, which generally equals the number of characters). str_length(fruit)
	str_pad(string, width, side = c("left", "right", "both"), pad = " ") Pad strings to constant width. str_pad(fruit, 17)
	str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "...") Truncate the width of strings, replacing content with ellipsis. str_trunc(sentences, 6)
	str_trim(string, side = c("both", "left", "right")) Trim whitespace from the start and/or end of a string. str_trim(str_pad(fruit, 17))
	str_squish(string) Trim whitespace from each end and collapse multiple spaces into single spaces. str_squish(str_pad(fruit, 17, "both"))

Mutate Strings

	str_sub() <- value. Replace substrings by identifying the substrings with str_sub() and assigning into the results. str_sub(fruit, 1, 3) <- "str"
	str_replace(string, pattern, replacement) Replace the first matched pattern in each string. Also str_remove() . str_replace(fruit, "p", "-")
	str_replace_all(string, pattern, replacement) Replace all matched patterns in each string. Also str_remove_all() . str_replace_all(fruit, "p", "-")
	str_to_lower(string, locale = "en")¹ Convert strings to lower case. str_to_lower(sentences)
	str_to_upper(string, locale = "en")¹ Convert strings to upper case. str_to_upper(sentences)
	str_to_title(string, locale = "en")¹ Convert strings to title case. Also str_to_sentence() . str_to_title(sentences)

Join and Split

	str_c(..., sep = "", collapse = NULL) Join multiple strings into a single string. str_c(letters, LETTERS)
	str_flatten(string, collapse = "") Combines into a single string, separated by collapse. str_flatten(fruit, ",")
	str_dup(string, times) Repeat strings times times. Also str_unique() to remove duplicates. str_dup(fruit, times = 2)
	str_split_fixed(string, pattern, n) Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match). Also str_split() to return a list of substrings and str_split_i() to return the ith substring. str_split_fixed(sentences, " ", n=3)
	str_glue(..., .sep = "", .envir = parent.frame()) Create a string from strings and {expressions} to evaluate. str_glue("Pi is {pi}")
	str_glue_data(.x, ..., .sep = "", .envir = parent.frame(), .na = "NA") Use a data frame, list, or environment to create a string from strings and {expressions} to evaluate. str_glue_data(mtcars, "[rownames(mtcars)] has {hp} hp")

Order Strings

	str_order(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)¹ Return the vector of indexes that sorts a character vector. fruit[str_order(fruit)]
	str_sort(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)¹ Sort a character vector. str_sort(fruit)

Helpers

	str_conv(string, encoding) Override the encoding of a string. str_conv(fruit, "ISO-8859-1")
	str_view(string, pattern, match = NA) View HTML rendering of all regex matches. str_view(sentences, "[aeiou])")
	str_equal(x, y, locale = "en", ignore_case = FALSE, ...)¹ Determine if two strings are equivalent. str_equal(c("a", "b"), c("a", "c"))
	str_wrap(string, width = 80, indent = 0, exdent = 0) Wrap strings into nicely formatted paragraphs. str_wrap(sentences, 20)

¹ See bit.ly/ISO639-1 for a complete list of locales.

Need to Know

Pattern arguments in string are interpreted as regular expressions *after any special characters have been parsed*.

In R, you write regular expressions as *strings*, sequences of characters surrounded by quotes ("") or single quotes('').

Some characters cannot be represented directly in an R string. These must be represented as **special characters**, sequences of characters that have a specific meaning., e.g.

Special Character	Represents
\\"	\
'"	"
\n	new line

Run `?""` to see a complete list

Because of this, whenever a \ appears in a regular expression, you must write it as \\ in the string that represents the regular expression.

Use `writeLines()` to see how R views your string after all special characters have been parsed.

```
writeLines("|\.")  
# \.
```

```
writeLines("\\| is a backslash")  
# \| is a backslash
```

INTERPRETATION

Patterns in stringr are interpreted as regexs. To change this default, wrap the pattern in one of:

`regex(pattern, ignore_case = FALSE, multiline = FALSE, comments = FALSE, dotall = FALSE, ...)`
Modifies a regex to ignore cases, match end of lines as well of end of strings, allow R comments within regex's , and/or to have . match everything including \n.
`str_detect("i", regex("i", TRUE))`

`fixed()` Matches raw bytes but will miss some characters that can be represented in multiple ways (fast). `str_detect("\u0130", fixed("i"))`

`coll()` Matches raw bytes and will use locale specific collation rules to recognize characters that can be represented in multiple ways (slow). `str_detect("\u0130", coll("i", TRUE, locale = "tr"))`

`boundary()` Matches boundaries between characters, line_breaks, sentences, or words. `str_split(sentences, boundary("word"))`

Regular Expressions -

Regular expressions, or *regexp*s, are a concise language for describing patterns in strings.

MATCH CHARACTERS

string (type this)	regexp (to mean this)	matches (which matches this)	example
a (etc.)	a (etc.)	a (etc.)	see("a")
\.	\.	.	see("\.")
\!	\!	!	see("\!")
\?	\?	?	see("\?")
\\\	\\\	\	see("\\\\")
\(\((see("\()")
\)	\))	see("\)")
\{	\{	{	see("\{")
\}	\}	}	see("\}")
\n	\n	new line (return)	see("\n")
\t	\t	tab	see("\t")
\s	\s	any whitespace (\\$ for non-whitespaces)	see("\s")
\d	\d	any digit (\D for non-digits)	see("\d")
\w	\w	any word character (\W for non-word chars)	see("\w")
\b	\b	word boundaries	see("\b")
[:digit:] ¹	[:digit:] ¹	digits	see("[:digit:]")
[:alpha:] ¹	[:alpha:] ¹	letters	see("[:alpha:]")
[:lower:] ¹	[:lower:] ¹	lowercase letters	see("[:lower:]")
[:upper:] ¹	[:upper:] ¹	uppercase letters	see("[:upper:]")
[:alnum:] ¹	[:alnum:] ¹	letters and numbers	see("[:alnum:]")
[:punct:] ¹	[:punct:] ¹	punctuation	see("[:punct:]")
[:graph:] ¹	[:graph:] ¹	letters, numbers, and punctuation	see("[:graph:]")
[:space:] ¹	[:space:] ¹	space characters (i.e. \s)	see("[:space:]")
[:blank:] ¹	[:blank:] ¹	space and tab (but not new line)	see("[:blank:]")
.	.	every character except a new line	see(".")

¹ Many base R functions require classes to be wrapped in a second set of [], e.g. [[:digit:]]

ALTERNATES

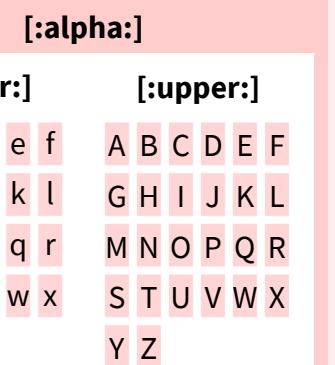
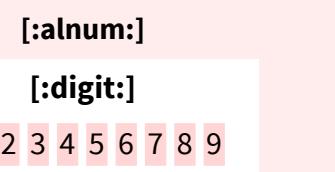
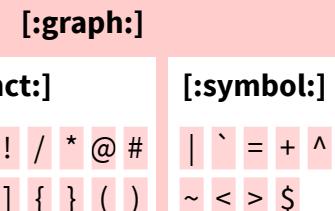
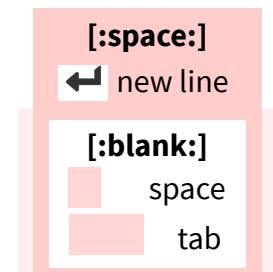
regexp	matches	example
ab d	or	alt("ab d")
[abe]	one of	alt("[abe]")
[^abe]	anything but	alt("[^abe]")
[a-c]	range	alt("[a-c]")

ANCHORS

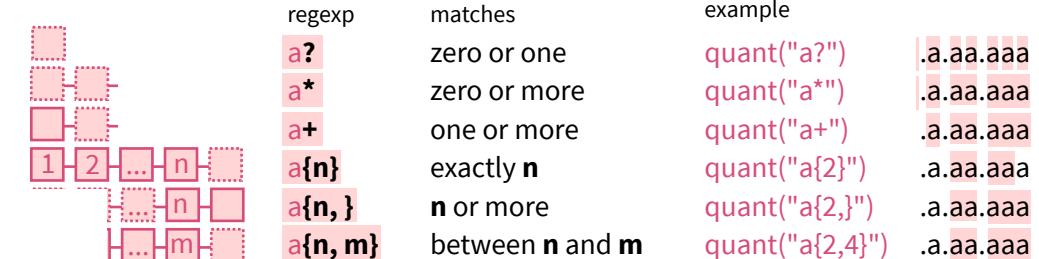
regexp	matches	example
^a	start of string	anchor("^a")
a\$	end of string	anchor("a\$")

LOOK AROUNDS

regexp	matches	example
a(?=c)	followed by	look("a(?=c)")
a(?!c)	not followed by	look("a(?!c)")
(?<=b)a	preceded by	look("(?<=b)a")
(?<!b)a	not preceded by	look("(?<!b)a")



QUANTIFIERS



GROUPS

Use parentheses to set precedent (order of evaluation) and create groups

regexp	matches	example
(ab d)e	sets precedence	alt("(ab d)e")

Use an escaped number to refer to and duplicate parentheses groups that occur earlier in a pattern. Refer to each group by its order of appearance

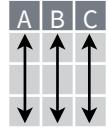
string (type this)	regexp (to mean this)	matches (which matches this)	example
\1	\1 (etc.)	first () group, etc.	ref("(a)(b)\1\2\1")

Data tidying with `tidyr` :: CHEATSHEET



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**

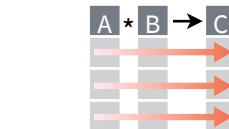
&



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

A tibble: 3 × 2
 <int> <chr>
 1 1 a
 2 2 b
 3 3 c

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



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

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).

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

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

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

Drop other variables.

</div



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))
```

name	yr	lat	long
Amy	1975	27.5	-79.0
Amy	1975	28.5	-79.0
Amy	1975	29.5	-79.0
Bob	1979	22.0	-96.0
Bob	1979	22.5	-95.3
Bob	1979	23.0	-94.6
Zeta	2005	23.9	-35.6
Zeta	2005	24.2	-36.1
Zeta	2005	24.7	-36.6

name	yr	lat	long
Amy	1975	27.5	-79.0
Amy	1975	28.5	-79.0
Amy	1975	29.5	-79.0
Bob	1979	22.0	-96.0
Bob	1979	22.5	-95.3
Bob	1979	23.0	-94.6
Zeta	2005	23.9	-35.6
Zeta	2005	24.2	-36.1
Zeta	2005	24.7	-36.6

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,
       4, 1:4,
       5, 1:5)
```

max	seq
3	<int [3]>
4	<int [4]>
5	<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)
```

name	films
Luke	The Empire Strik...
Luke	Revenge of the S...
Luke	Return of the Jed...
C-3PO	The Empire Strik...
C-3PO	Attack of the Cl...
C-3PO	The Phantom M...
R2-D2	The Empire Strik...
R2-D2	Attack of the Cl...
R2-D2	The Phantom M...

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

```
starwars |>
  select(name, films) |>
  unnest_wider(films, names_sep = "_")
```

name	films
Luke	<chr [5]>
C-3PO	<chr [6]>
R2-D2	<chr[7]>

name	films_1	films_2	films_3
Luke	The Empire...	Revenge of...	Return of...
C-3PO	The Empire...	Attack of...	The Phantom...
R2-D2	The Empire...	Attack of...	The Phantom...

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)
```

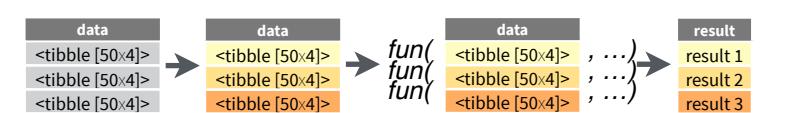
name	films
Luke	<chr [5]>
C-3PO	<chr [6]>
R2-D2	<chr[7]>

name	first_film	second_film	films
Luke	The Empire...	Revenge of...	<chr [3]>
C-3PO	The Empire...	Attack of...	<chr [4]>
R2-D2	The Empire...	Attack of...	<chr [5]>

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.