

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

<code>c(2, 4, 6)</code>	<code>2 4 6</code>	Join elements into a vector
<code>2:6</code>	<code>2 3 4 5 6</code>	An integer sequence
<code>seq(2, 3, by=0.5)</code>	<code>2.0 2.5 3.0</code>	A complex sequence
<code>rep(1:2, times=3)</code>	<code>1 2 1 2 1 2</code>	Repeat a vector
<code>rep(1:2, each=3)</code>	<code>1 1 1 2 2 2</code>	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

<code>x[4]</code>	The fourth element.
<code>x[-4]</code>	All but the fourth.
<code>x[2:4]</code>	Elements two to four.
<code>x[-(2:4)]</code>	All elements except two to four.
<code>x[c(1, 5)]</code>	Elements one and five.

By Value

<code>x[x == 10]</code>	Elements which are equal to 10.
<code>x[x < 0]</code>	All elements less than zero.
<code>x[x %in% c(1, 2, 5)]</code>	Elements in the set 1, 2, 5.

Named Vectors

<code>x['apple']</code>	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
}
```

If Statements

```
if (condition){
  Do something
} else {
  Do something different
}
```

Example

```
if (i > 3){
  print('Yes')
} else {
  print('No')
}
```

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

Conditions

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

Types

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

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

Maths Functions

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

Variable Assignment

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

The Environment

<code>ls()</code>	List all variables in the environment.
<code>rm(x)</code>	Remove x from the environment.
<code>rm(list = ls())</code>	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.
```

 <code>m[2,]</code>	- Select a row	<code>t(m)</code>	Transpose
 <code>m[, 1]</code>	- Select a column	<code>m %*% n</code>	Matrix Multiplication
 <code>m[2, 3]</code>	- Select an element	<code>solve(m, n)</code>	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.
```

<code>l[[2]]</code>	<code>l[1]</code>	<code>l\$x</code>	<code>l['y']</code>
Second element of l.	New list with only the first element.	Element named x.	New list with only element named y.

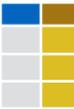
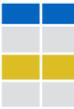
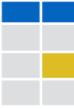
Also see the **dplyr** package.

Data Frames

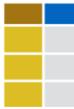
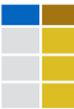
```
df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))
A special case of a list where all elements are the same length.
```

x	y
1	a
2	b
3	c

Matrix subsetting

<code>df[, 2]</code>	
<code>df[2,]</code>	
<code>df[2, 2]</code>	

List subsetting

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

Understanding a data frame

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

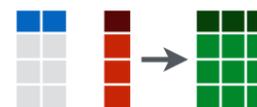
`head(df)` See the first 6 rows.

`nrow(df)`
Number of rows.

`ncol(df)`
Number of columns.

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

`cbind` - Bind columns.



`rbind` - Bind rows.



Strings

Also see the **stringr** package.

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

Factors

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

Statistics

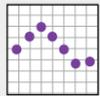
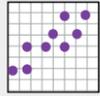
<code>lm(y ~ x, data=df)</code>	Linear model.	<code>t.test(x, y)</code>	Perform a t-test for difference between means.	<code>prop.test</code>	Test for a difference between proportions.
<code>glm(y ~ x, data=df)</code>	Generalised linear model.	<code>summary</code>	Get more detailed information out a model.	<code>pairwise.t.test</code>	Perform a t-test for paired data.
				<code>aov</code>	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>punif</code>	<code>qunif</code>

Plotting

Also see the **ggplot2** package.

	<code>plot(x)</code> Values of x in order.		<code>plot(x, y)</code> Values of x against y.		<code>hist(x)</code> Histogram of x.
---	---	---	---	---	---

Dates

See the **lubridate** package.

RStudio IDE : : CHEATSHEET



Documents and Apps

Open Shiny, R Markdown, knitr, Sweave, LaTeX, .Rd files and more in Source Pane
 Check spelling, Render output, Choose output format, Configure render options, Insert code chunk, Publish to server
 Jump to previous chunk, Jump to next chunk, Run code, Show file outline, Visual Editor (reverse side)
 Jump to section or chunk, Run all previous code chunks, Run this code chunk, Set knitr chunk options

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

Run app, Choose location to view app, Publish to shinyapps.io or server, Manage publish accounts

Source Editor

Navigate backwards/forwards, Open in new window, Save, Find and replace, Compile as notebook, Run selected code
 Re-run previous code, Source with or w/out Echo or as a Local Job, Show file outline
 Multiple cursors/column selection with **Alt + mouse drag**.
 Code diagnostics that appear in the margin. Hover over diagnostic symbols for details.
 Syntax highlighting based on your file's extension
 Tab completion to finish function names, file paths, arguments, and more.
 Multi-language code snippets to quickly use common blocks of code.
 Jump to function in file, Change file type
 Working Directory, Run scripts in separate sessions, Maximize, minimize panes
 Ctrl/Cmd + arrow up to see history, R Markdown Build Log, Drag pane boundaries

Tab Panes

Import data with wizard, History of past commands to run/copy, Manage external databases, View memory usage, R tutorials
 Load workspace, Save workspace, Clear R workspace, Search inside environment
 Choose environment to display from list of parent environments, Display objects as list or grid
 Displays saved objects by type with short description, View in data viewer, View function source code
 Create folder, Delete file, Rename file, Change directory
 Path to displayed directory, A File browser keyed to your working directory. Click on file or directory name to open.

Version Control

Turn on at **Tools > Project Options > Git/SVN**
 Added, Deleted, Modified, Renamed, Untracked
 Stage files, Commit staged files, Push/Pull to remote, View History, Current branch
 Show file diff to view file differences
 Open shell to type commands

Debug Mode

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

Launch debugger mode from origin of error, Open traceback to examine the functions that R called before the error occurred
 Show Traceback, Rerun with Debug

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**

Install package and restart R, Run devtools::load_all() and reload changes
 Run R CMD check, Customize package build options, Clear output and rebuild, Run package tests

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

Navigate recent plots, Open in window, Export plot, Delete plot, Delete all plots

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

Home page of helpful links, Search within help file, Search for help file

GUI **Package** manager lists every installed package

Install Packages, Update Packages, Browse package site, Click to load package with **library()**. Unclick to detach package with **detach()**. Package version installed, Delete from library

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

Stop Shiny app, Publish to shinyapps.io, Posit Connect, Posit Cloud, ... Refresh

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

	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

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

Step through code one line at a time, Step into and out of functions to run, Resume execution, Quit debug mode





Keyboard Shortcuts

RUN CODE

	Windows/Linux	Mac
Search command history	Ctrl+arrow-up	Cmd+arrow-up
Interrupt current command	Esc	Esc
Clear console	Ctrl+L	Ctrl+L

NAVIGATE CODE

	Windows/Linux	Mac
Go to File/Function	Ctrl+.	Ctrl+.

WRITE CODE

	Windows/Linux	Mac
Attempt completion	Tab or Ctrl+Space	Tab or Ctrl+Space
Insert <- (assignment operator)	Alt+-	Option+-
Insert > or %>% (pipe operator)	Ctrl+Shift+M	Cmd+Shift+M
(Un)Comment selection	Ctrl+Shift+C	Cmd+Shift+C

MAKE PACKAGES

	Windows/Linux	Mac
Load All (devtools)	Ctrl+Shift+L	Cmd+Shift+L
Test Package (Desktop)	Ctrl+Shift+T	Cmd+Shift+T
Document Package	Ctrl+Shift+D	Cmd+Shift+D

DOCUMENTS AND APPS

	Windows/Linux	Mac
Knit/Render 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

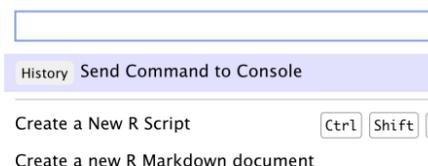
MORE KEYBOARD SHORTCUTS

	Windows/Linux	Mac
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**



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:

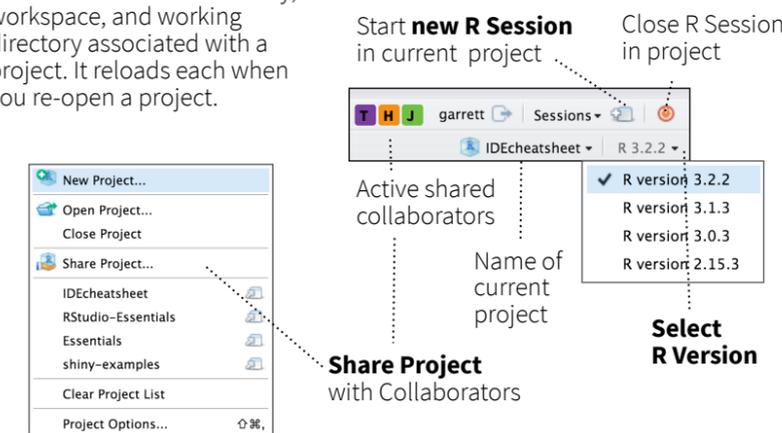
- open and run multiple R sessions at once
- tune your resources to improve performance
- administrative tools for managing user sessions
- collaborate real-time with others in shared projects
- switch easily from one version of R to a different version
- integrate with your authentication, authorization, 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/

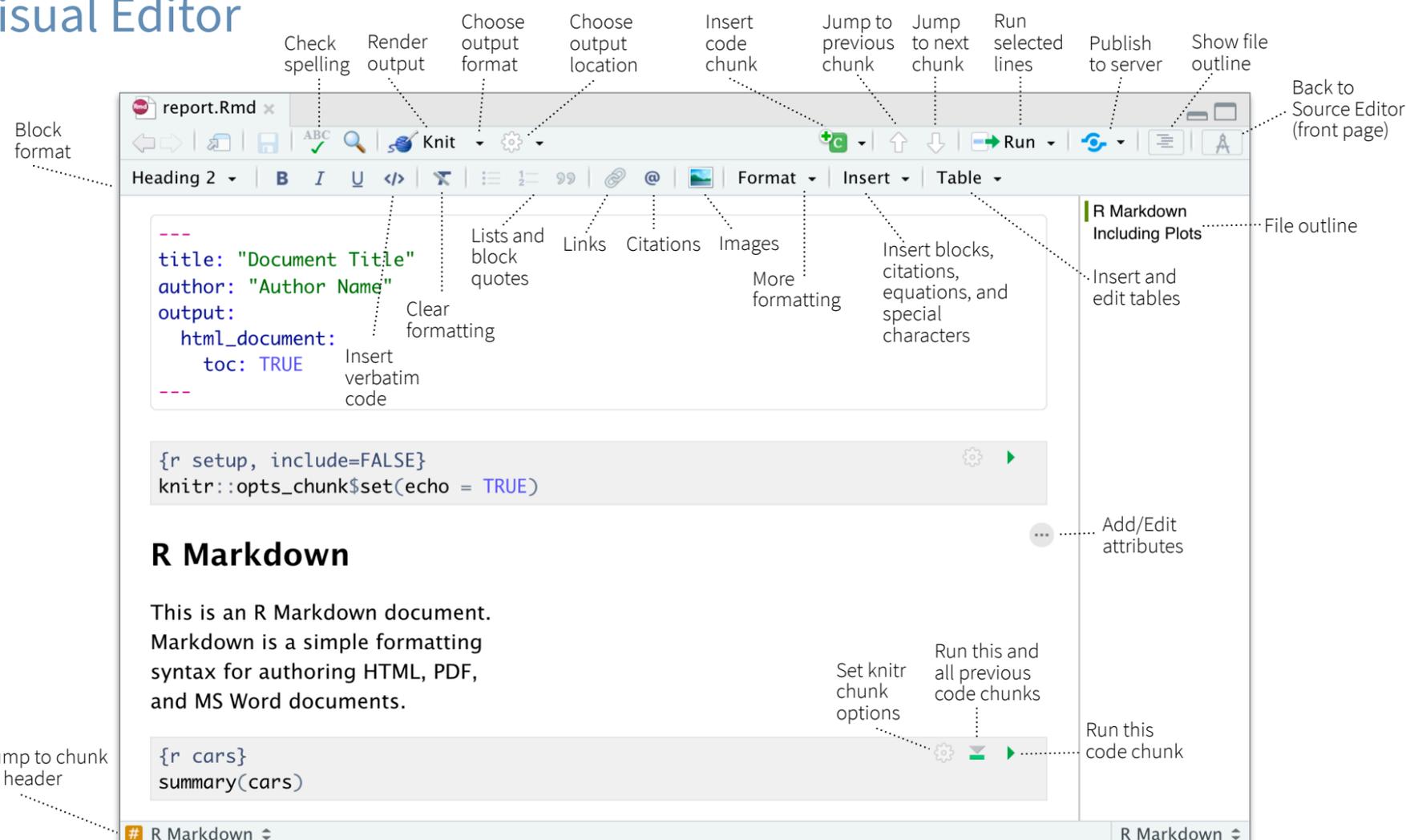
Share Projects

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.

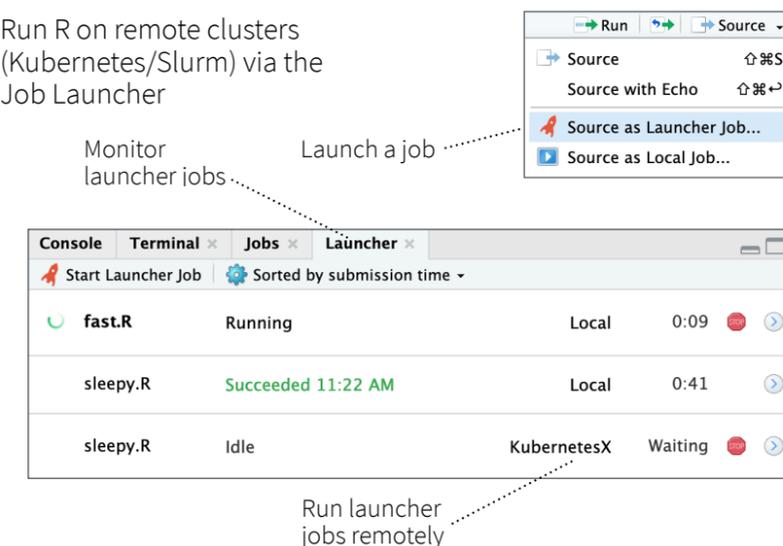


Visual Editor

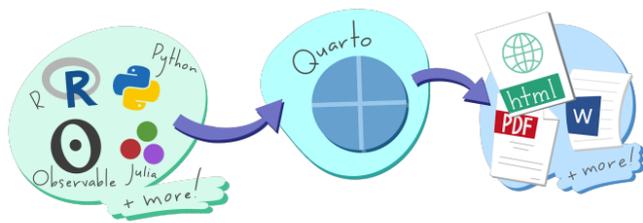


Run Remote Jobs

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



Publish and Share with Quarto : : CHEATSHEET



GET QUARTO

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

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

GET STARTED

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

Author

SOURCE FILE: `hello.qmd`

```
---
title: "Hello, Penguins"
format: html
execute:
  echo: false
---

## Meet the penguins

The `penguins` data contain
from three islands in the P

The three species of penguin have quite distinct
distributions of physical dimensions (@fig-penguins).

```{r}
#| label: fig-penguins
#| fig-cap: "Dimensions of penguins a
#| 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](#)).

**Features for scientific publishing**  
Cross references, citations, equations, and more

**Output integrated into document**  
Control how output appears with special comments in your code

Terminal

```
quarto publish {venue} hello.qmd
```

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



Free publishing service for Quarto content.



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

### USE A TOOL WITH A RICH EDITING EXPERIENCE



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:

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

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

| 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 <code>`r2+2`</code> . | Value is <code>`{python}2+2`</code> . | Value is 4. |

# Set Format and Options

## SET FORMAT OPTIONS

```

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

```

Indent format 2 spaces  
Indent options 4 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

|         | OPTION                       | html/revealjs<br>pdf/beamer<br>docx/pptx | DESCRIPTION                                                                                                   |
|---------|------------------------------|------------------------------------------|---------------------------------------------------------------------------------------------------------------|
| Nav     | <b>toc</b>                   | X X X                                    | Add a table of contents (true or false)                                                                       |
|         | <b>toc-depth</b>             | X X X                                    | Lowest level of headings to add to table of contents (e.g. 2, 3)                                              |
|         | <b>anchor-sections</b>       | X                                        | Show section anchors on mouse hover (true or false)                                                           |
| Style   | <b>highlight-style</b>       | X X X                                    | Syntax highlighting theme (e.g. arrow, pygments, kate, zenburn)                                               |
|         | <b>mainfont, monofont</b>    | X X                                      | Font name. HTML: sets CSS font-family; LaTeX: via fontspec package                                            |
|         | <b>theme</b>                 | X                                        | Bootswatch theme name (e.g. cosmo, darkly, solar etc.)                                                        |
|         | <b>css</b>                   | X                                        | CSS or SCSS file to use to style the document (e.g. "style.css")                                              |
|         | <b>reference-doc</b>         | X                                        | docx/pptx file containing template styles (e.g. file.docx, file.pptx)                                         |
| LaTeX   | <b>include-in-header</b>     | X X                                      | Files of content to include in header of output document, also <b>include-before-body, include-after-body</b> |
|         | <b>keep-md</b>               | X X X                                    | Keep intermediate markdown (true or false), also <b>keep-ipynb, keep-tex</b>                                  |
|         | <b>documentclass</b>         | X                                        | LaTeX document class, set document class options with <b>classoption</b>                                      |
| Code    | <b>pdf-engine</b>            | X                                        | LaTeX engine to produce PDF output (xelatex, pdflatex, lualatex)                                              |
|         | <b>cite-method</b>           | X                                        | Method used to format citations (citeproc, natbib, biblatex)                                                  |
|         | <b>code-fold</b>             | X                                        | Let readers toggle the display of R code (false, true, or show)                                               |
| Figures | <b>code-tools</b>            | X                                        | Add menu for hiding, showing, and downloading code (true or false)                                            |
|         | <b>code-overflow</b>         | X                                        | Display of wide code (scroll, or wrap)                                                                        |
|         | <b>fig-align</b>             | X X /                                    | Alignment of figures (default, left, right, or center)                                                        |
|         | <b>fig-width, fig-height</b> | X X X                                    | Default width and height for figures in inches                                                                |
|         | <b>fig-format</b>            | 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

# Add Content

## FIGURES

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

MARKDOWN

COMPUTATION

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

Or `{r}`

## CROSS REFERENCES

- Add labels**  
Code cell: add option `label: prefix-LABEL`  
Markdown: add attribute `#prefix-LABEL`
- Add references** `@prefix-LABEL`, e.g.

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

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

## TABLES

### MARKDOWN

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

**Insert Table in Visual Editor**

**COMPUTATION** Output a Markdown table or an HTML table from your code

### KNITR

Use `knitr::kable()` to produce Markdown:

```
```{r}
#| label: tbl-LABEL
#| tbl-cap: CAPTION
knitr::kable(head(cars))
```
```

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

**JUPYTER** Add Markdown ( ) to Markdown output:

```
```{python}
#| 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))
```
```

## CITATIONS

- Add a bibliography **file** to the YAML header:

```

bibliography: references.bib

```

- Add citations: `[@citation]`, or `@citation`

Use **Insert Citations** dialog in the **Visual Editor**

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

## CALLOUTS

```
::: { .callout-tip }
Title
```

Text  
:::

Instead of **tip** use one of: note, caution, warning, or 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!**

**1. New File**

**2. Embed Code**

**3. Write Text**

**4. Set Output Format(s) and Options**

**5. Save and Render**

**6. Share**

Visual Editor: modify chunk options, run all previous chunks, run current chunk

**Visual Editor**

add/edit attributes, style options, insert citations

## 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$set(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"
```

| OPTION                            | DEFAULT   | EFFECTS                                                                                                   |
|-----------------------------------|-----------|-----------------------------------------------------------------------------------------------------------|
| <b>echo</b>                       | TRUE      | display code in output document                                                                           |
| <b>error</b>                      | FALSE     | TRUE (display error messages in doc)<br>FALSE (stop render when error occurs)                             |
| <b>eval</b>                       | TRUE      | run code in chunk                                                                                         |
| <b>include</b>                    | TRUE      | include chunk in doc after running                                                                        |
| <b>message</b>                    | TRUE      | display code messages in document                                                                         |
| <b>warning</b>                    | TRUE      | display code warnings in document                                                                         |
| <b>results</b>                    | "markup"  | "asis" (passthrough results)<br>"hide" (don't display results)<br>"hold" (put all results below all code) |
| <b>fig.align</b>                  | "default" | "left", "right", or "center"                                                                              |
| <b>fig.alt</b>                    | NULL      | alt text for a figure                                                                                     |
| <b>fig.cap</b>                    | NULL      | figure caption as a character string                                                                      |
| <b>fig.path</b>                   | "figure/" | prefix for generating figure file paths                                                                   |
| <b>fig.width &amp; fig.height</b> | 7         | plot dimensions in inches                                                                                 |
| <b>out.width</b>                  |           | rescales output width, e.g. "75%", "300px"                                                                |
| <b>collapse</b>                   | FALSE     | collapse all sources & output into a single block                                                         |
| <b>comment</b>                    | "###"     | prefix for each line of results                                                                           |
| <b>child</b>                      | NULL      | file(s) to knit and then include                                                                          |
| <b>purl</b>                       | TRUE      | include or exclude a code chunk when extracting source code with <code>knitr::purl()</code>               |

See more options and defaults by running `str(knitr::opts_chunk$get())`

## RENDERED OUTPUT

file path to output document

find in document

publish to [rpubs.com](https://rpubs.com), [shinyapps.io](https://shinyapps.io), Posit Connect

reload document

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  |

## 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)`.

| eruptions | waiting |
|-----------|---------|
| 3.600     | 79      |
| 1.800     | 54      |
| 3.333     | 74      |
| 2.283     | 62      |

```
```\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**                               | <i>italics</i> and <b>bold</b>                       |
| superscript <sup>2</sup> /subscript <sub>2</sub>     | superscript <sup>2</sup> /subscript <sub>2</sub>     |
| ~strikethrough~                                      | <del>strikethrough</del>                             |
| escaped: \_ \_                                       | escaped: * _ \                                       |
| endash: --, emdash: ---                              | endash: –, emdash: —                                 |

| # Header 1                                                                                                                        | Header 1                                                                                                                                                                                  |
|-----------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ## Header 2                                                                                                                       | Header 2                                                                                                                                                                                  |
| ...                                                                                                                               | ...                                                                                                                                                                                       |
| ##### Header 6                                                                                                                    | Header 6                                                                                                                                                                                  |
| <ul style="list-style-type: none"> <li>unordered list</li> <li>item 2</li> <li>item 2a (indent 1 tab)</li> <li>item 2b</li> </ul> | <ul style="list-style-type: none"> <li>unordered list</li> <li>item 2               <ul style="list-style-type: none"> <li>item 2a (indent 1 tab)</li> <li>item 2b</li> </ul> </li> </ul> |
| <ol style="list-style-type: none"> <li>ordered list</li> <li>item 2</li> <li>item 2a (indent 1 tab)</li> <li>item 2b</li> </ol>   | <ol style="list-style-type: none"> <li>ordered list</li> <li>item 2               <ul style="list-style-type: none"> <li>item 2a (indent 1 tab)</li> <li>item 2b</li> </ul> </li> </ol>   |

| <link url>                                      | <a href="http://www.posit.co/">http://www.posit.co/</a>                               |
|-------------------------------------------------|---------------------------------------------------------------------------------------|
| [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:<br>[id]: link url   |  |
| <br>or ![(Caption)][id2]          | Caption.                                                                              |
| At the end of the document:<br>[id2]: image.png |                                                                                       |

| `verbatim code`                 | verbatim code                   |
|---------------------------------|---------------------------------|
| multiple lines of verbatim code | multiple lines of verbatim code |
| > block quotes                  | block quotes                    |
| equation: $e^{i\pi} + 1 = 0$    | equation: $e^{i\pi} + 1 = 0$    |
| equation block:<br>$E = mc^2$   | equation block:<br>$E = mc^2$   |
| horizontal rule:<br>---         | horizontal rule:<br>-----       |

| Right                 | Left | Default | Center |
|-----------------------|------|---------|--------|
| 12   12   12   12     | 12   | 12      | 12     |
| 123   123   123   123 | 123  | 123     | 123    |
| 1   1   1   1         | 1    | 1       | 1      |

**HTML Tabsets**

```
Results {tabset}
Plots
text

Tables
more text
```

**Results**

Plots Tables

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 <a href="#">Bootswatch</a> and <a href="#">Custom Themes</a> 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`



## 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).



## 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"

```
2. **Call parameters** in code using `params$<name>`.
 

```
data <- df[, params$state]
summary(data)
```
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).
 

```

name: "My Template"

```
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.

.my-css-tag ...

This is a div with some text in it.

### 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	2.00
2	4.00	10.00
3	7.00	4.00
4	7.00	22.00
5	8.00	16.00

Also see Shiny Pre-rendered for better performance. rmarkdown.rstudio.com/authoring_shiny_pre-rendered.

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



Data visualization with ggplot2 : : CHEATSHEET

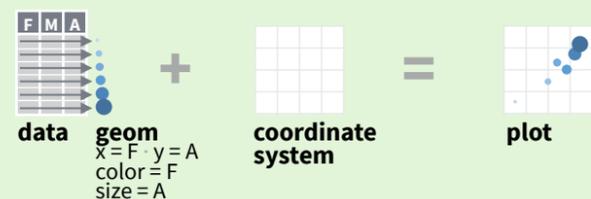


Basics

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>
```

required (for GEOM_FUNCTION, STAT, POSITION)
Not required, sensible defaults supplied (for COORDINATE_FUNCTION, FACET_FUNCTION, SCALE_FUNCTION, THEME_FUNCTION)

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")

size - integer (in mm for size of points and text)

linewidth - integer (in mm for widths of lines)

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, xend, y, yend, 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, label, alpha, angle, color, 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, label, alpha, angle, 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)); l <- ggplot(seals, aes(long, lat))
```

l + geom_contour(aes(z = z))
x, y, z, alpha, color, group, linetype, size, weight

l + geom_contour_filled(aes(fill = z))
x, y, alpha, color, fill, group, linetype, size, subgroup

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_density_2d()
x, y, alpha, color, group, linetype, size

h + geom_hex()
x, y, alpha, color, fill, 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, 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

Draw the appropriate geometric object depending on the simple features present in the data. aes() arguments: map_id, alpha, color, fill, linetype, linewidth.

```
nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"))
```

ggplot(nc) + geom_sf(aes(fill = AREA))

l + geom_raster(aes(fill = z), hjust = 0.5, vjust = 0.5, interpolate = FALSE)
x, y, alpha, fill

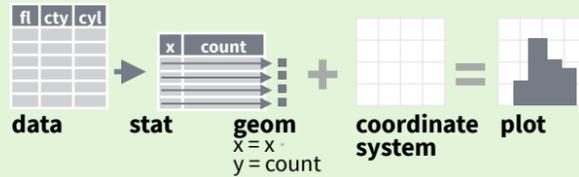
l + geom_tile(aes(fill = z))
x, y, alpha, color, fill, linetype, size, width



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 `after_stat(name)` syntax to map the stat variable `name` to an aesthetic.

geom to use **stat function** **geom mappings**

```
i + stat_density_2d(aes(fill = after_stat(level)), geom = "polygon")
```

variable created by stat

- `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.

```
n <- d + geom_bar(aes(fill = fl))
```

scale **aesthetic to adjust** **prepackaged scale to use** **scale-specific arguments**

```
n + scale_fill_manual(values = c("skyblue", "royalblue", "blue", "navy"), limits = c("d", "e", "p", "r"), breaks = c("d", "e", "p", "r"), name = "fuel", labels = c("D", "E", "P", "R"))
```

range of values to include in **title to use in legend/axis** **labels to use in legend/axis** **breaks to use in legend/axis**

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)

- `n + scale_fill_brewer(palette = "Blues")`
For palette choices: `RColorBrewer::display.brewer.all()`
- `n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")`

COLOR AND FILL SCALES (CONTINUOUS)

- `o <- c + geom_dotplot(aes(fill = x))`
- `o + scale_fill_distiller(palette = "Blues")`
- `o + scale_fill_gradient(low="red", high="yellow")`
- `o + scale_fill_gradient2(low="red", high="blue", mid="white", midpoint = 25)`
- `o + scale_fill_gradientn(colors = topo.colors(6))`
Also: `rainbow()`, `heat.colors()`, `terrain.colors()`, `cm.colors()`, `RColorBrewer::brewer.pal()`

SHAPE AND SIZE SCALES

- ```
p <- e + geom_point(aes(shape = fl, size = cyl))
```
- `p + scale_shape() + scale_size()`
  - `p + scale_shape_manual(values = c(3:7))`  
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
  - `p + scale_radius(range = c(1,6))`
  - `p + scale_size_area(max_size = 6)`

# 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.
 - `r + coord_flip()`
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.
 - `π + coord_sf()` - `xlim`, `ylim`, `crs`. Ensures all layers use a common Coordinate Reference System.

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_classic()`
  - `r + theme_light()`
  - `r + theme_linedraw()`
  - `r + theme_gray()`  
Grey background (default theme).
  - `r + theme_minimal()`  
Minimal theme.
  - `r + theme_dark()`  
Dark for contrast.
  - `r + theme_void()`  
Empty theme.
- `r + theme()` Customize aspects of the theme such as axis, legend, panel, and facet properties.
- ```
r + labs(title = "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(. ~ fl)`  
Facet into columns based on fl.
  - `t + facet_grid(year ~ .)`  
Facet into rows based on year.
  - `t + facet_grid(year ~ fl)`  
Facet into both rows and columns.
  - `t + facet_wrap(~ fl)`  
Wrap facets into a rectangular layout.

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

- ```
t + facet_grid(drv ~ 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 `labeller` to adjust facet label:

- ```
t + facet_grid(. ~ fl, labeller = label_both)
```
- | fl: c      | fl: d      | fl: e      | fl: p      | fl: r      |
|------------|------------|------------|------------|------------|
| $\alpha^c$ | $\alpha^d$ | $\alpha^e$ | $\alpha^p$ | $\alpha^r$ |
- ```
t + facet_grid(fl ~ ., labeller = label_bquote(alpha ^ .(fl)))
```

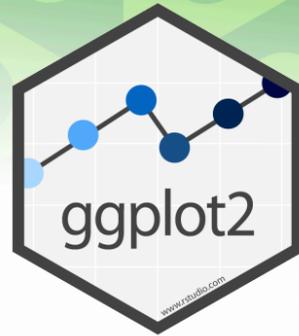
Labels and Legends

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", <AES> = "New <AES> 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

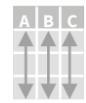
- Without clipping (preferred):**
`t + coord_cartesian(xlim = c(0, 100), ylim = c(10, 20))`
- With clipping (removes unseen data points):**
`t + xlim(0, 100) + ylim(10, 20)`
`t + scale_x_continuous(limits = c(0, 100)) + scale_y_continuous(limits = c(0, 100))`



Data transformation with dplyr : : CHEATSHEET



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



&



pipes

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

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

$x \mid> f(y)$ 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).

summary function

 **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 tidyr cheat sheet for list-column workflow.

 `starwars |> rowwise() |> mutate(film_count = length(films))`

ungroup(x, ...) Returns ungrouped copy of table.

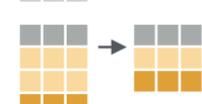
`g_mtcars <- mtcars |> group_by(cyl)`
`ungroup(g_mtcars)`

Manipulate Cases

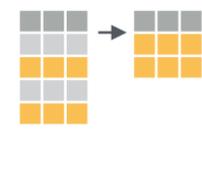
EXTRACT CASES

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)`

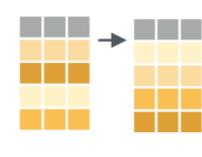
 **slice_head(.data, ..., n, prop)** and **slice_tail()** Select the first or last rows.
`mtcars |> slice_head(n = 5)`

Logical and boolean operators to use with filter()

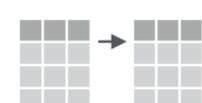
<code>==</code>	<code><</code>	<code><=</code>	<code>is.na()</code>	<code>%in%</code>	<code> </code>	<code>xor()</code>
<code>!=</code>	<code>></code>	<code>>=</code>	<code>!is.na()</code>	<code>!</code>	<code>&</code>	

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)	num_range(prefix, range)	; e.g., <code>mpg:cyl</code>
ends_with(match)	all_of(x)/any_of(x, ..., vars)	! e.g., <code>!gear</code>
starts_with(match)	matches(match)	everything()

MANIPULATE MULTIPLE VARIABLES AT ONCE

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

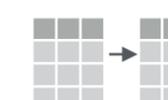
 **across(.cols, .funs, ..., .names = NULL)** Summarize or mutate multiple columns in the same way.
`df |> summarize(across(everything(), mean))`

 **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.



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.



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
1	a t
2	b u
3	c v

 →

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

`tibble::rownames_to_column()`
Move row names into col.

```
a <- mtcars |>
rownames_to_column(var = "C")
```

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

 →

A1	B1	C	A2	B2
a	t		d	w
b	u		b	u
c	v		a	t

`tibble::column_to_rownames()`
Move col into row names.

```
a |> column_to_rownames(var = "C")
```

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

Combine Tables

COMBINE VARIABLES

X
a t 1
b u 2
c v 3

 +

Y
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.

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.

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.

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.

A	B	C	D
a t 1	3		
b u 2	2		
c v 3	NA		
d w	NA	1	

full_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ..., keep = FALSE, na_matches = "na") Join data. Retain all values, all rows.

COLUMN MATCHING FOR JOINS

A	B.x	C	B.y	D
a	t	1	t	3
b	u	2	u	2
c	v	3	NA	NA

 Use **by = c("col1", "col2", ...)** to specify one or more common columns to match on.
`left_join(x, y, by = "A")`

A.x	B.x	C	A.y	B.y
a	t	1	d	w
b	u	2	b	u
c	v	3	a	t

 Use a named vector, **by = c("col1" = "col2")**, to match on columns that have different names in each table.
`left_join(x, y, by = c("C" = "D"))`

A1	B1	C	A2	B2
a	t	1	d	w
b	u	2	b	u
c	v	3	a	t

 Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.
`left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))`

COMBINE CASES

X
a t 1
b u 2

 +

Y
a t 3
d w 4

 =

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

 +

Y
a t 3
b u 2
d w 1

 =

A	B	C
a t 1		
b u 2		

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
c v 3		

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
d	w	4

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

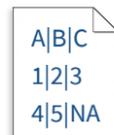
Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

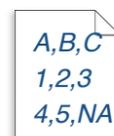
Data import with the tidyverse : : CHEATSHEET

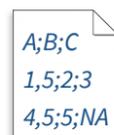


Read Tabular Data with readr

`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`

 `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")`

 `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")`

 `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")`

 `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\n", file = "file.tsv")`

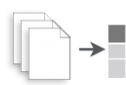
USEFUL READ ARGUMENTS

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"))`

 **Read multiple files into a single table**
`read_csv(c("f1.csv", "f2.csv", "f3.csv"), id = "origin_file")`

1	2	3
4	5	NA

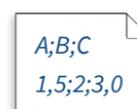
Skip lines
`read_csv("file.csv", skip = 1)`

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"))`

 **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)

USEFUL COLUMN ARGUMENTS

Hide col spec message

`read_*(file, show_col_types = FALSE)`

Select columns to import

Use names, position, or selection helpers.
`read_*(file, col_select = c(age, earn))`

Guess column types

To guess a column type, `read_*`() looks at the first 1000 rows of data. Increase with **guess_max**.
`read_*(file, guess_max = Inf)`

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

→

x1	x2	x3	x4	x5
x	NA	z	8	NA
y	7	NA	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

read_excel(path, sheet = NULL) Specify which sheet to read by position or name.
`read_excel(path, sheet = 1)`
`read_excel(path, sheet = "s1")`

excel_sheets(path) Get a vector of sheet names.
`excel_sheets("excel_file.xlsx")`

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

```
path <- "your_file_path.xlsx" data frame.
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**



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

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

CELL SPECIFICATION FOR READXL AND GOOGLESHEETS4

A	B	C	D	E
1	1	2	3	4
2	x	y	z	
3	6	7		9 10

→

2	3	4
NA	y	z

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()**.

with googlesheets4

READ SHEETS

A	B	C	D	E
1	x1	x2	x3	x4
2	x		z	8
3	y	7		9 10

→

x1	x2	x3	x4	x5
x	NA	z	8	NA
y	7	NA	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

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.



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_sheet()/range_read()` looks at the first 1000 rows of data. Increase with **guess_max**.
`read_sheet(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

l	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 **tidyr** 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
 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(tzone = "") Current time in tz (defaults to system tz). now()

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

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

parse_date_time() Easier strptime.
 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.

```
d ## "2017-11-28"
day(d) ## 28
day(d) <- 1
d ## "2017-11-01"
```

2018-01-31 11:59:59 **date(x)** Date component. date(dt)

2018-01-31 11:59:59 **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)

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

quarter(x) Quarter. quarter(dt)

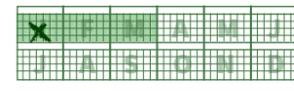
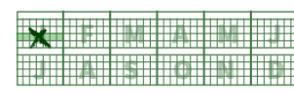
semester(x, with_year = FALSE) Semester. semester(dt)

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

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

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

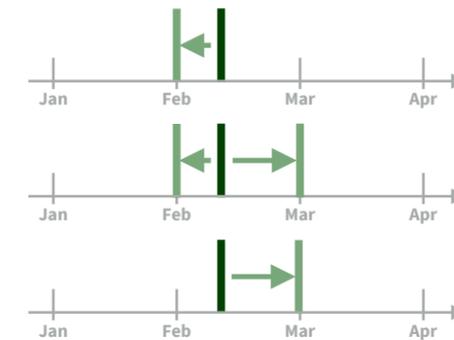
update(object, ..., simple = FALSE)
 update(dt, mday = 2, hour = 1)



12:00:00
 An **hms** is a **time** stored as the number of seconds since 00:00:00

```
t <- hms::as_hms(85)
## "00:01:25"
```

Round Date-times



floor_date(x, unit = "second")
 Round down to nearest unit.
 floor_date(dt, unit = "month")

round_date(x, unit = "second")
 Round to nearest unit.
 round_date(dt, unit = "month")

ceiling_date(x, unit = "second", change_on_boundary = NULL)
 Round up to nearest unit.
 ceiling_date(dt, unit = "month")

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

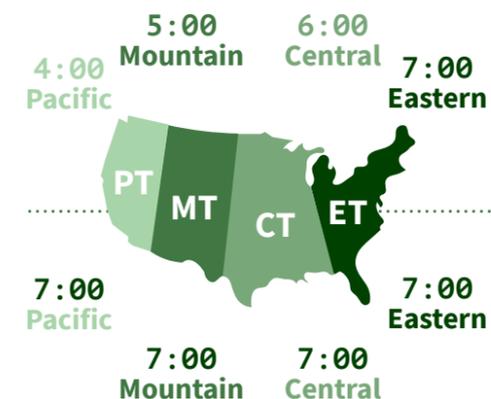
Time Zones

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.



with_tz(time, tzone = "") 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, tzone = "") 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

— Lubridate provides three classes of timespans to facilitate math with dates and date-times.

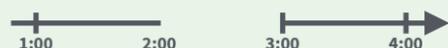


Math with date-times relies on the **timeline**, which behaves inconsistently. Consider how the timeline behaves during:

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")`



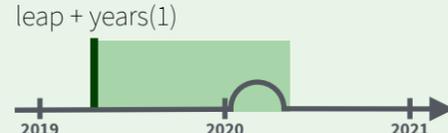
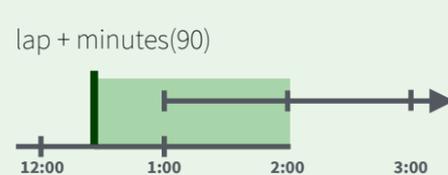
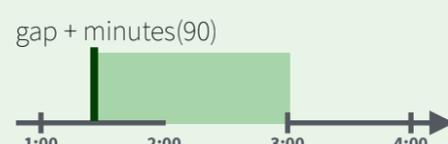
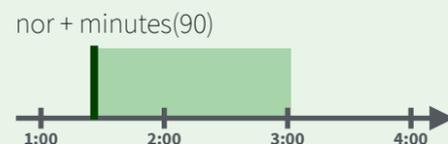
The end of daylight savings (fall back)
`lap <- ymd_hms("2018-11-04 00:30:00",tz="US/Eastern")`



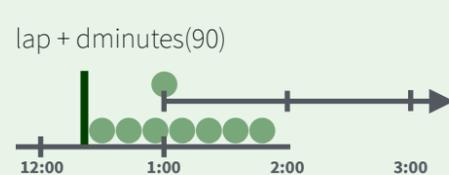
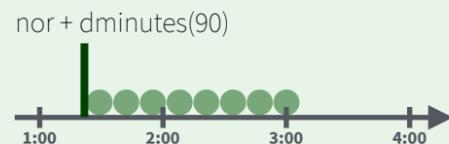
Leap years and leap seconds
`leap <- ymd("2019-03-01")`



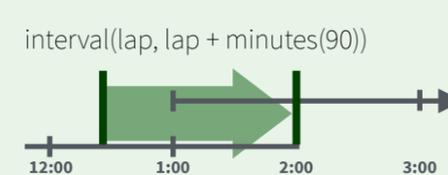
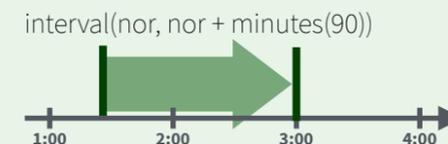
Periods track changes in clock times, which ignore time line irregularities.



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



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



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"
```



- 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. **Diffimes** 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)"
```



- 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)** $x \times 10^{-3}$ seconds.
- dmicroseconds(x = 1)** $x \times 10^{-6}$ seconds.
- dnanoseconds(x = 1)** $x \times 10^{-9}$ seconds.
- dpicoseconds(x = 1)** $x \times 10^{-12}$ 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.diffime()**.
`as.duration(i)`

make_diffime(x) Make diffime with the specified number of units.
`make_diffime(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)
j <- d %--% ymd("2017-12-31")
## 2017-01-01 UTC--2017-11-28 UTC
## 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

TRUE
TRUE
FALSE
TRUE

str_detect(string, **pattern**, negate = FALSE)
 Detect the presence of a pattern match in a string. Also **str_like()**. `str_detect(fruit, "a")`

TRUE
TRUE
FALSE
TRUE

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")`

1
 2
 4

str_which(string, **pattern**, negate = FALSE)
 Find the indexes of strings that contain a pattern match. `str_which(fruit, "a")`

start end
 2 4
 4 7
 NA NA
 3 4

str_locate(string, **pattern**) Locate the positions of pattern matches in a string. Also **str_locate_all()**. `str_locate(fruit, "a")`

0
 3
 1
 2

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

4
 6
 2
 3

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", "-")`

A STRING
 ↓
 a string

str_to_lower(string, locale = "en")¹
 Convert strings to lower case. `str_to_lower(sentences)`

a string
 ↓
 A STRING

str_to_upper(string, locale = "en")¹
 Convert strings to upper case. `str_to_upper(sentences)`

a string
 ↓
 A String

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)`

{xx} {yy}

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

4
 1
 3
 2

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")`

appl<e>
 banana
 p<e>ar

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"))`

TRUE
 TRUE
 FALSE
 TRUE

str_wrap(string, width = 80, indent = 0, exdent = 0) Wrap strings into nicely formatted paragraphs. `str_wrap(sentences, 20)`

This is a long sentence.
 ↓
 This is a long sentence.

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

Need to Know

Pattern arguments in stringr 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 *regexps*, are a concise language for describing patterns in strings.

MATCH CHARACTERS

string (type this)	regex (to mean this)	matches (which matches this)	example
	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 (S 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	digits	see("[:digit:]")
	[:alpha:] ¹	letters	see("[:alpha:]")
	[:lower:] ¹	lowercase letters	see("[:lower:]")
	[:upper:] ¹	uppercase letters	see("[:upper:]")
	[:alnum:] ¹	letters and numbers	see("[:alnum:]")
	[:punct:] ¹	punctuation	see("[:punct:]")
	[:graph:] ¹	letters, numbers, and punctuation	see("[:graph:]")
	[:space:] ¹	space characters (i.e. \s)	see("[:space:]")
	[: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:]`



[:space:]
← new line

[:blank:]
space
tab

[:graph:]

[:punct:] . , : ; ? ! / * @ #	[:symbol:] ` = + ^
- _ " ' [] { } ()	~ < > \$

[:alnum:]

[:digit:]
0 1 2 3 4 5 6 7 8 9

[:alpha:]

[:lower:] a b c d e f	[:upper:] A B C D E F
g h i j k l	G H I J K L
m n o p q r	M N O P Q R
s t u v w x	S T U V W X
y z	Y Z

ALTERNATES

```
alt <- function(rx) str_view("abcde", rx)
```

regex	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

```
anchor <- function(rx) str_view("aaa", rx)
```

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

LOOK AROUNDS

```
look <- function(rx) str_view("bacad", rx)
```

regex	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

```
quant <- function(rx) str_view("a.aa.aaa", rx)
```

regex	matches	example
a?	zero or one	quant("a?")
a*	zero or more	quant("a*")
a+	one or more	quant("a+")
a{n}	exactly n	quant("a{2}")
a{n,}	n or more	quant("a{2,}")
a{n,m}	between n and m	quant("a{2,4}")

GROUPS

```
ref <- function(rx) str_view("abbaab", rx)
```

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

regex	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)	regex (to mean this)	matches (which matches this)	example (the result is the same as ref("abba"))
\\1	\\1 (etc.)	first () group, etc.	ref("(a)(b)\\2\\1")

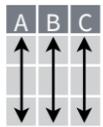


Data tidying with tidyr :: CHEATSHEET



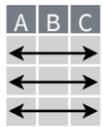
Tidy data is a way to organize tabular data in a consistent data structure across packages.

A table is tidy if:



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

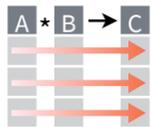
&



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"))`

`tribble(...)` Construct by rows.

```
tribble(~x, ~y,
  1, "a",
  2, "b",
  3, "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

`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")`

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

`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)`

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

`unite(data, col, ..., sep = "_", remove = TRUE, na.rm = FALSE)` Collapse cells across several columns into a single column.

`unite(table5, century, year, col = "year", sep = "")`

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

`separate_wider_delim(data, cols, delim, ..., names = NULL, names_sep = NULL, names_repair = "check_unique", too_few, too_many, cols_remove = TRUE)` Separate each cell in a column into several columns. Also `separate_wider_regex()` and `separate_wider_position()`.

`separate(table3, rate, sep = "/", into = c("cases", "pop"))`

table3

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

`separate_longer_delim(data, cols, delim, ..., width, keep_empty)` Separate each cell in a column into several rows.

`separate_longer_delim(table3, rate, sep = "/")`

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

→

x1	x2
A	1
A	2
B	1
B	2

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

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

x

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)`

Handle Missing Values

Drop or replace explicit missing values (NA).

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
D	3

`drop_na(data, ...)` Drop rows containing NA's in ... columns.

`drop_na(x, x2)`

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	1
C	1
D	3
E	3

`fill(data, ..., .direction = "down")` Fill in NA's in ... columns using the next or previous value.

`fill(x, x2)`

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	2
C	2
D	3
E	2

`replace_na(data, replace)` Specify a value to replace NA in selected columns.

`replace_na(x, list(x2 = 2))`

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.

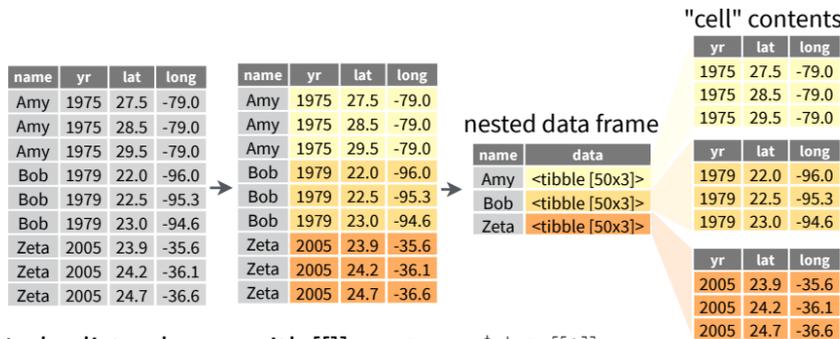
CREATE NESTED DATA

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,
  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)))
```

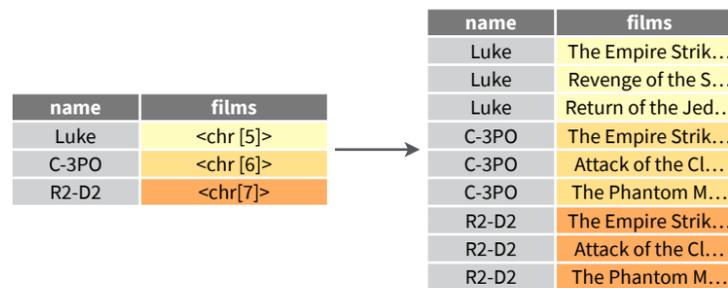
RESHAPE NESTED DATA

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

```
n_storms |> unnest(data)
```

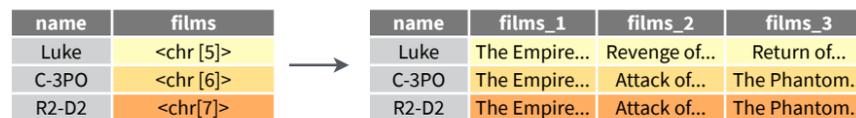
unnest_longer(data, col, values_to = NULL, indices_to = NULL) Turn each element of a list-column into a row.

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



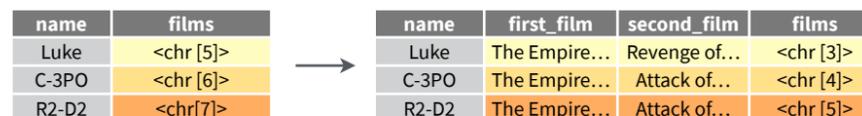
unnest_wider(data, col) Turn each element of a list-column into a regular column.

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



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

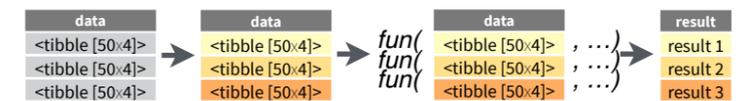
```
starwars |>
  select(name, films) |>
  hoist(films, first_film = 1, second_film = 2)
```



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.