

RStudio IDE Cheat Sheet

learn more at www.rstudio.com



The RStudio IDE is an Integrated Development Environment in R that comes in three versions



Desktop IDE

A local version of the IDE for your desktop



Open Source Server

for larger compute resources and remote access



Professional Server

for teams that share large compute resources, large data, and uniform environments for collaboration

Download all at www.rstudio.com. Each provides the same useful interface:

Documents and Apps

Icons:

Open Shiny, R Markdown, knitr, Sweave, LaTeX, .Rd files and more in Source Pane

Check spelling **Render output** **Choose output format** **Choose output location** **Insert code chunk**

Jump to previous chunk **Jump to next chunk** **Run selected lines** **Publish to server** **Show file outline**

Access markdown guide at Help > Markdown Quick Reference

Jump to chunk **Set knitr chunk options** **Run this and all previous code chunks** **Run this code chunk**

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**

Write Code

Navigate tabs **Open in new window** **Save** **Find and replace** **Compile as notebook** **Run selected code**

Cursors of shared users **Re-run previous code** **Source with or without Echo** **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.

Jump to function in file

Multi-language code snippets to quickly use common blocks of code.

Change file type

Working Directory **Press ↑ to see command history**

Maximize, minimize panes **Drag pane boundaries**

R Support

Import data file with wizard

History of past commands to run/add to source

Display .RPres slideshows

File > New File > R Presentation

Load workspace **Save workspace** **Delete all saved objects** **Search inside environment**

Choose environment to display from list of parent environments

Data **Values** **Functions**

View in data viewer **View function source code**

Displays saved objects by type with short description

Path to displayed directory

A File browser keyed to your working directory. Click on file or directory name to open.

RStudio Pro Features

Share Project with Collaborators

Active shared collaborators

Start new R Session in current project

Close R Session in project

Select R Version

Project System

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.

RStudio opens plots in a dedicated Plots pane

Export plot

GUI Package manager lists every installed package

Install Packages **Update Packages** **Create reproducible package library for your project**

Click to load package with **library(). Unclick to detach package with **detach()****

Package version installed **Delete from library**

Debug Mode

Open with **debug(), **browse()**, or a breakpoint. RStudio will open the debugger mode when it encounters a breakpoint while executing code.**

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

Launch debugger mode from origin of error

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

Console ~ /IDEcheatsheet /

Traceback **Show Traceback** **Rerun with Debug**

Console ~ /IDEcheatsheet /

Next **Continue** **Stop**

Version Control with Git or SVN

Turn on at Tools > Project Options > Git/SVN

G Stage files: **Show file diff** **Commit staged files to remote** **Push/Pull** **View History**

Added **Deleted** **Modified** **Renamed** **Untracked**

Environment **History** **Git**

Staged **Status** **Commit** **Path**

file-with-changes.R

Revert... **Ignore...** **Shell...**

Open shell to type commands

Package Writing

File > New Project > New Directory > R Package

Turn project into package, Enable roxygen documentation with **Tools > Project Options > Build Tools**

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

Help and Documentation

RStudio opens documentation in a dedicated Help pane

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

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

Stop Shiny app **Publish to shinyapps.io, rpubs, RSConnect, ...** **Refresh**

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

Filter **Sort by values** **Search for value**

1 LAYOUT	Windows/Linux	Mac	4 WRITE CODE	Windows /Linux	Mac	5 DEBUG CODE	Windows/Linux	Mac
Move focus to Source Editor	Ctrl+1	Ctrl+1	Attempt completion	Tab or Ctrl+Space	Tab or Cmd+Space	Toggle Breakpoint	Shift+F9	Shift+F9
Move focus to Console	Ctrl+2	Ctrl+2	Navigate candidates	↑/↓	↑/↓	Execute Next Line	F10	F10
Move focus to Help	Ctrl+3	Ctrl+3	Accept candidate	Enter, Tab, or →	Enter, Tab, or →	Step Into Function	Shift+F4	Shift+F4
Show History	Ctrl+4	Ctrl+4	Dismiss candidates	Esc	Esc	Finish Function/Loop	Shift+F6	Shift+F6
Show Files	Ctrl+5	Ctrl+5	Undo	Ctrl+Z	Cmd+Z	Continue	Shift+F5	Shift+F5
Show Plots	Ctrl+6	Ctrl+6	Redo	Ctrl+Shift+Z	Cmd+Shift+Z	Stop Debugging	Shift+F8	Shift+F8
Show Packages	Ctrl+7	Ctrl+7	Cut	Ctrl+X	Cmd+X			
Show Environment	Ctrl+8	Ctrl+8	Copy	Ctrl+C	Cmd+C			
Show Git/SVN	Ctrl+9	Ctrl+9	Paste	Ctrl+V	Cmd+V			
Show Build	Ctrl+0	Ctrl+0	Select All	Ctrl+A	Cmd+A			
2 RUN CODE	Windows/Linux	Mac	Delete Line	Ctrl+D	Cmd+D			
Search command history	Ctrl+↑	Cmd+↑	Select	Shift+[Arrow]	Shift+[Arrow]			
Navigate command history	↑/↓	↑/↓	Select Word	Ctrl+Shift+←→	Option+Shift+←→	6 VERSION CONTROL	Windows/Linux	Mac
Move cursor to start of line	Home	Cmd+←	Select to Line Start	Alt+Shift+←	Cmd+Shift+←	Show diff	Ctrl+Alt+D	Ctrl+Option+D
Move cursor to end of line	End	Cmd+→	Select to Line End	Alt+Shift+→	Cmd+Shift+→	Commit changes	Ctrl+Alt+M	Ctrl+Option+M
Change working directory	Ctrl+Shift+H	Ctrl+Shift+H	Select Page Up/Down	Shift+PageUp/Down	Shift+PageUp/Down	Scroll diff view	Ctrl+↑/↓	Ctrl+↑/↓
Interrupt current command	Esc	Esc	Select to Start/End	Shift+Alt+↑/↓	Cmd+Shift+↑/↓	Stage/Unstage (Git)	Spacebar	Spacebar
Clear console	Ctrl+L	Ctrl+L	Delete Word Left	Ctrl+Opt+Backspace	Ctrl+Opt+Backspace	Stage/Unstage and move to next	Enter	Enter
Quit Session (desktop only)	Ctrl+Q	Cmd+Q	Delete Word Right	Option+Delete				
Restart R Session	Ctrl+Shift+F10	Cmd+Shift+F10	Delete to Line End	Ctrl+K				
Run current line/selection	Ctrl+Enter	Cmd+Enter	Delete to Line Start	Option+Backspace				
Run current (retain cursor)	Alt+Enter	Option+Enter	Indent	Tab (at start of line)	Tab (at start of line)	7 MAKE PACKAGES	Windows/Linux	Mac
Run from current to end	Ctrl+Alt+E	Cmd+Option+E	Outdent	Shift+Tab	Shift+Tab	Build and Reload	Ctrl+Shift+B	Cmd+Shift+B
Run the current function	Ctrl+Alt+F	Cmd+Option+F	Yank line up to cursor	Ctrl+U	Ctrl+U	Load All (devtools)	Ctrl+Shift+L	Cmd+Shift+L
Source a file	Ctrl+Shift+O	Cmd+Shift+O	Yank line after cursor	Ctrl+K	Ctrl+K	Test Package (Desktop)	Ctrl+Shift+T	Cmd+Shift+T
Source the current file	Ctrl+Shift+S	Cmd+Shift+S	Insert yanked text	Ctrl+Y	Ctrl+Y	Test Package (Web)	Ctrl+Alt+F7	Cmd+Alt+F7
Source with echo	Ctrl+Shift+Enter	Cmd+Shift+Enter	Insert <-	Alt+-	Option+	Check Package	Ctrl+Shift+E	Cmd+Shift+E
3 NAVIGATE CODE	Windows /Linux	Mac	Insert %>%	Ctrl+Shift+M	Cmd+Shift+M	Document Package	Ctrl+Shift+D	Cmd+Shift+D
Goto File/Function	Ctrl+.	Ctrl+.	Show help for function	F1	F1	8 DOCUMENTS AND APPS	Windows/Linux	Mac
Fold Selected	Alt+L	Cmd+Option+L	Show source code	F2	F2	Preview HTML (Markdown, etc.)	Ctrl+Shift+K	Cmd+Shift+K
Unfold Selected	Shift+Alt+L	Cmd+Shift+Option+L	New document	Ctrl+Shift+N	Cmd+Shift+N	Knit Document (knitr)	Ctrl+Shift+K	Cmd+Shift+K
Fold All	Alt+O	Cmd+Option+O	New document (Chrome)	Ctrl+Alt+Shift+N	Cmd+Shift+Alt+N	Compile Notebook	Ctrl+Shift+K	Cmd+Shift+K
Unfold All	Shift+Alt+O	Cmd+Shift+Option+O	Open document	Ctrl+O	Cmd+O	Compile PDF (TeX and Sweave)	Ctrl+Shift+K	Cmd+Shift+K
Go to line	Shift+Alt+G	Cmd+Shift+Option+G	Save document	Ctrl+S	Cmd+S	Insert chunk (Sweave and Knitr)	Ctrl+Alt+I	Cmd+Option+I
Jump to	Shift+Alt+J	Cmd+Shift+Option+J	Close document	Ctrl+W	Cmd+W	Insert code section	Ctrl+Shift+R	Cmd+Shift+R
Switch to tab	Ctrl+Shift+.	Ctrl+Shift+.	Close document (Chrome)	Ctrl+Alt+W	Cmd+Option+W	Re-run previous region	Ctrl+Shift+P	Cmd+Shift+P
Previous tab	Ctrl+F11	Ctrl+F11	Close all documents	Ctrl+Shift+W	Cmd+Shift+W	Run current document	Ctrl+Alt+R	Cmd+Option+R
Next tab	Ctrl+F12	Ctrl+F12	Extract function	Ctrl+Alt+X	Cmd+Option+X	Run from start to current line	Ctrl+Alt+B	Cmd+Option+B
First tab	Ctrl+Shift+F11	Ctrl+Shift+F11	Extract variable	Ctrl+Alt+V	Cmd+Option+V	Run the current code section	Ctrl+Alt+T	Cmd+Option+T
Last tab	Ctrl+Shift+F12	Ctrl+Shift+F12	Reindent lines	Ctrl+I	Cmd+I	Run previous Sweave/Rmd code	Ctrl+Alt+P	Cmd+Option+P
Navigate back	Ctrl+F9	Cmd+F9	(Un)Comment lines	Ctrl+Shift+C	Cmd+Shift+C	Run the current chunk	Ctrl+Alt+C	Cmd+Option+C
Navigate forward	Ctrl+F10	Cmd+F10	Reflow Comment	Ctrl+Shift+/	Cmd+Shift+/	Run the next chunk	Ctrl+Alt+N	Cmd+Option+N
Jump to Brace	Ctrl+P	Ctrl+P	Reformat Selection	Ctrl+Shift+A	Cmd+Shift+A	Sync Editor & PDF Preview	Ctrl+F8	Cmd+F8
Select within Braces	Ctrl+Shift+Alt+E	Ctrl+Shift+Alt+E	Select within braces	Ctrl+Shift+E	Ctrl+Shift+E	Previous plot	Ctrl+Alt+F11	Cmd+Option+F11
Use Selection for Find	Ctrl+F3	Cmd+E	Show Diagnostics	Ctrl+Shift+Alt+P	Cmd+Shift+Alt+P	Next plot	Ctrl+Alt+F12	Cmd+Option+F12
Find in Files	Ctrl+Shift+F	Cmd+Shift+F	Transpose Letters	Ctrl+T	Ctrl+T	Show Keyboard Shortcuts	Alt+Shift+K	Option+Shift+K
Find Next	Win: F3, Linux: Ctrl+G	Cmd+G	Move Lines Up/Down	Alt+↑/↓	Option+↑/↓	Why RStudio Server Pro?		
Find Previous	W: Shift+F3, L: Ctrl+Shift	Cmd+Shift+G	Copy Lines Up/Down	Shift+Alt+↑/↓	Cmd+Option+↑/↓	Do everything you would do with the open source server with a commercial license, support, and more.		
Jump to Word	Ctrl+←→	Option+←→	Add New Cursor Above	Ctrl+Alt+Up	Ctrl+Alt+Up	• edit the same project at the same time as others		
Jump to Start/End	Ctrl+↑/↓	Cmd+↑/↓	Add New Cursor Below	Ctrl+Alt+Down	Ctrl+Alt+Down	• switch easily from one version of R to a different version		
			Move Active Cursor Up	Ctrl+Alt+Shift+Up	Ctrl+Alt+Shift+Up	• open and run multiple R sessions simultaneously		
			Move Active Cursor Down	Ctrl+Alt+Shift+Down	Ctrl+Alt+Shift+Down	• see what you and others are doing on your server		
			Find and Replace	Ctrl+F	Cmd+F	• tune your resources to improve performance		
			Use Selection for Find	Ctrl+F3	Cmd+E	• integrate with your authentication, authorization, and audit practices		
			Replace and Find	Ctrl+Shift+J	Cmd+Shift+J	Download a free 45 day evaluation at www.rstudio.com/products/rstudio-server-pro/		

Base R Cheat Sheet

Getting Help

Accessing the help files

?mean

Get help of a particular function.

help.search('weighted mean')

Search the help files for a word or phrase.

help(package = 'dplyr')

Find help for a package.

More about an object

str(iris)

Get a summary of an object's structure.

class(iris)

Find the class an object belongs to.

Using Packages

install.packages('dplyr')

Download and install a package from CRAN.

library(dplyr)

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

dplyr::select

Use a particular function from a package.

data(iris)

Load a built-in dataset into the environment.

Working Directory

getwd()

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

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

Change the current working directory.

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

Vectors

Creating Vectors

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

Vector Functions

sort(x)

Return x sorted.

rev(x)

Return x reversed.

table(x)

See counts of values.

unique(x)

See unique values.

Selecting Vector Elements

By Position

x[4]

The fourth element.

x[-4]

All but the fourth.

x[2:4]

Elements two to four.

x[!(2:4)]

All elements except two to four.

x[c(1, 5)]

Elements one and five.

By Value

x[x == 10]

Elements which are equal to 10.

x[x < 0]

All elements less than zero.

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

Elements in the set 1, 2, 5.

Named Vectors

x['apple']

Element with name 'apple'.

Programming

For Loop

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

Example

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

While Loop

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

Example

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

Functions

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

Example

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

Reading and Writing Data

Also see the **readr** package.

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

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

Types

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

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

Maths Functions

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

Variable Assignment

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

The Environment

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

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

Matrices

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

	<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: $m \cdot x = n$

Lists

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

<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.

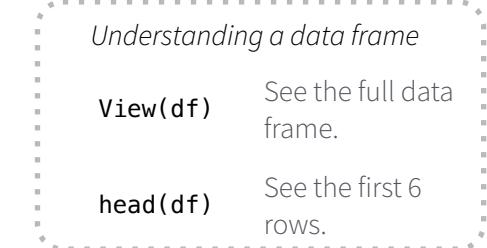
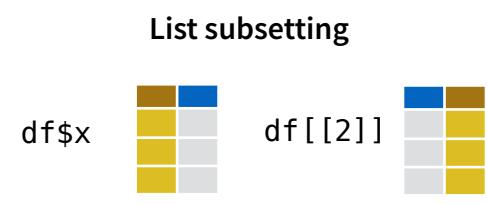
Data Frames

`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>	

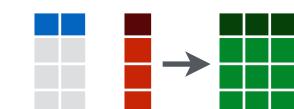


`nrow(df)`
Number of rows.

`ncol(df)`
Number of columns.

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

`cbind` - Bind columns.



`rbind` - Bind rows.



Strings

<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>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.

Distributions

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

Plotting

<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.

Data Import

with `readr`, `tibble`, and `tidyR`

Cheat Sheet



R's **tidyverse** is built around **tidy data** stored in **tibbles**, an enhanced version of a data frame.

The front side of this sheet shows how to read text files into R with `readr`.

The reverse side shows how to create tibbles with `tibble` and to layout tidy data with `tidyR`.

Other types of data

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

- **haven** - SPSS, Stata, and SAS files
- **readxl** - excel files (.xls and .xlsx)
- **DBI** - databases
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)

Write functions

Save `x`, an R object, to `path`, a file path, with:

`write_csv(x, path, na = "NA", append = FALSE, col_names = !append)`

Tibble/df to comma delimited file.

`write_delim(x, path, delim = " ", na = "NA", append = FALSE, col_names = !append)`

Tibble/df to file with any delimiter.

`write_excel_csv(x, path, na = "NA", append = FALSE, col_names = !append)`

Tibble/df to a CSV for excel

`write_file(x, path, append = FALSE)`

String to file.

`write_lines(x, path, na = "NA", append = FALSE)`

String vector to file, one element per line.

`write_rds(x, path, compress = c("none", "gz", "bz2", "xz", ...))`

Object to RDS file.

`write_tsv(x, path, na = "NA", append = FALSE, col_names = !append)`

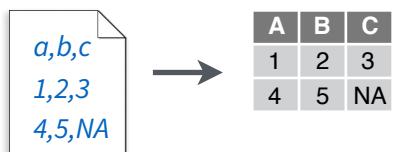
Tibble/df to tab delimited files.

Read functions

Read tabular data to tibbles

These functions share the common arguments:

```
read_*(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"),
      quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max =
      min(1000, n_max), progress = interactive())
```



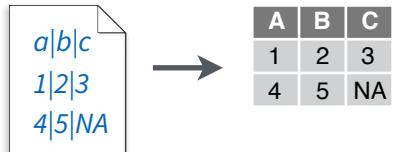
read_csv()

Reads comma delimited files.
`read_csv("file.csv")`



read_csv2()

Reads Semi-colon delimited files.
`read_csv2("file2.csv")`



read_delim()

(delim, quote = "\\"", escape_backslash = FALSE, escape_double = TRUE) Reads files with any delimiter.
`read_delim("file.txt", delim = "|")`



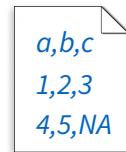
read_fwf()

(col_positions) Reads fixed width files.
`read_fwf("file.fwf", col_positions = c(1, 3, 5))`

read_tsv()

Reads tab delimited files. Also `read_table()`.
`read_tsv("file.tsv")`

Useful arguments



Example file

```
write_csv(path = "file.csv",
          x = read_csv("a,b,c|n1,2,3|n4,5,NA"))
```

1	2	3
4	5	NA

Skip lines

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

A	B	C
1	2	3

Read in a subset

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

A	B	C
1	2	3
NA	NA	NA

Missing Values

```
read_csv("file.csv",
          na = c("4", "5", "!"))
```

Read non-tabular data

read_file(file, locale = default_locale())

Read a file into a single string.

read_file_raw(file)

Read a file into a raw vector.

read_lines(file, skip = 0, n_max = -1L, locale =

default_locale(), na = character(), progress = interactive())

Read each line into its own string.

read_lines_raw(file, skip = 0, n_max = -1L, progress = interactive())

Read each line into a raw vector.

read_log(file, col_names = FALSE, col_types =

NULL, skip = 0, n_max = -1, progress = interactive())

Apache style log files.

Parsing data types

`readr` functions guess the types of each column and convert types when appropriate (but will NOT convert strings to factors automatically).

A message shows the type of each column in the result.

```
## Parsed with column specification:
## cols(
##   age = col_integer(),
##   sex = col_character(),
##   earn = col_double()
## )
```

age is an integer
sex is a character
earn is a double (numeric)

1. Use `problems()` to diagnose problems

```
x <- read_csv("file.csv"); problems(x)
```

2. Use a `col_` function to guide parsing

- `col_guess()` - the default
- `col_character()`
- `col_double()`
- `col_euro_double()`
- `col_datetime(format = "")` Also `col_date(format = "")` and `col_time(format = "")`
- `col_factor(levels, ordered = FALSE)`
- `col_integer()`
- `col_logical()`
- `col_number()`
- `col_numeric()`
- `col_skip()`

```
x <- read_csv("file.csv", col_types = cols(
  A = col_double(),
  B = col_logical(),
  C = col_factor()
))
```

3. Else, read in as character vectors then parse with a `parse_` function.

- `parse_guess(x, na = c("", "NA"), locale = default_locale())`
- `parse_character(x, na = c("", "NA"), locale = default_locale())`
- `parse_datetime(x, format = "", na = c("", "NA"), locale = default_locale())` Also `parse_date()` and `parse_time()`
- `parse_double(x, na = c("", "NA"), locale = default_locale())`
- `parse_factor(x, levels, ordered = FALSE, na = c("", "NA"), locale = default_locale())`
- `parse_integer(x, na = c("", "NA"), locale = default_locale())`
- `parse_logical(x, na = c("", "NA"), locale = default_locale())`
- `parse_number(x, na = c("", "NA"), locale = default_locale())`

```
x$A <- parse_number(x$A)
```

Tibbles - an enhanced data frame

The **tibble** package provides a new S3 class for storing tabular data, the tibble. Tibbles inherit the data frame class, but improve two behaviors:

- **Display** - When you print a tibble, R provides a concise view of the data that fits on one screen.
- **Subsetting** - [always returns a new tibble, [[and \$ always return a vector.
- **No partial matching** - You must use full column names when subsetting

# A tibble: 234 x 6	manufacturer	model	displ	cyl	year
1 audi	a4	1.8	4	1999	
2 audi	a4	1.8	4	2000	
3 audi	a4	2.0	4	2000	
4 audi	a4	2.0	4	2000	
5 audi	a4	2.8	4	2000	
6 audi	a4	2.8	4	2000	
7 audi	a4 quattro	3.1	4	2000	
8 audi	a4 quattro	3.1	4	2000	
9 audi	a4 quattro	3.1	4	2000	
10 audi	a4 quattro	3.1	4	2000	

tibble display

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

data frame display

- Control the default appearance with options:
`options(tibble.print_max = n,
tibble.print_min = m, tibble.width = Inf)`
- View entire data set with `View(x, title)` or `glimpse(x, width = NULL, ...)`
- Revert to data frame with `as.data.frame()` (required for some older packages)

Construct a tibble in two ways

tibble(...)	Construct by columns. <code>tibble(x = 1:3, y = c("a", "b", "c"))</code>	Both make this tibble
tribble(...)	Construct by rows. <code>tribble(~x, ~y, 1, "a", 2, "b", 3, "c")</code>	

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

`enframe(x, name = "name", value = "value")`
Converts named vector to a tibble with a names column and a values column.

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

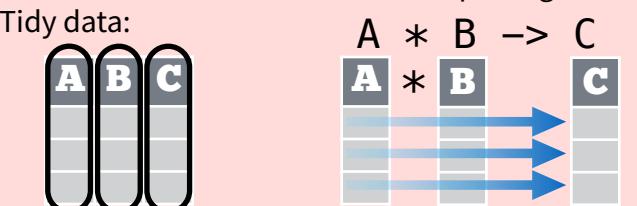
Tidy data is a way to organize tabular data. It provides a consistent data structure across packages. A table is tidy if:



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

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

Tidy Data with tidyr



Makes variables easy to access as vectors

Preserves cases during vectorized operations

Reshape Data - change the layout of values in a table

Use `gather()` and `spread()` to reorganize the values of a table into a new layout. Each uses the idea of a key column: value column pair.

gather(data, key, value, ..., na.rm = FALSE, convert = FALSE, factor_key = FALSE)

Gather moves column names into a key column, gathering the column values into a single value column.

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

key value

`gather(table4a, `1999`, `2000`,
key = "year", value = "cases")`

spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE, sep = NULL)

Spread moves the unique values of a key column into the column names, spreading the values of a value column across the new columns that result.

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

key value

spread(table2, type, count)

Handle Missing Values

drop_na(data, ...)

Drop rows containing NA's in ... columns.

x

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

→

x1	x2
A	1
D	3

`drop_na(x, x2)`

fill(data, ..., .direction = c("down", "up"))

Fill in NA's in ... columns with most recent non-NA values.

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(x, x2)`

replace_na(data, replace = list(), ...)

Replace NA's by column.

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

Split and Combine Cells

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

separate(data, col, into, sep = "[[:alnum:]]+", remove = TRUE, convert = FALSE, extra = "warn", fill = "warn", ...)

Separate each cell in a column to make several columns.

table3

country	year	rate
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

→

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

`separate_rows(table3, rate, into = c("cases", "pop"))`

separate_rows(data, ..., sep = "[[:alnum:]].+", convert = FALSE)

Separate each cell in a column to make several rows. Also `separate_rows_()`.

table3

country	year	rate
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

→

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
C	1999	212K
C	1999	1T
C	2000	213K
C	2000	1T

`separate_rows(table3, rate)`

unite(data, col, ..., sep = "_", remove = TRUE)

Collapse cells across several columns to make a single column.

table5

country	century	year
Afghan	19	99
Afghan	20	0
Brazil	19	99
Brazil	20	0
China	19	99
China	20	0

→

country	year
Afghan	1999
Afghan	2000
Brazil	1999
Brazil	2000
China	1999
China	2000

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

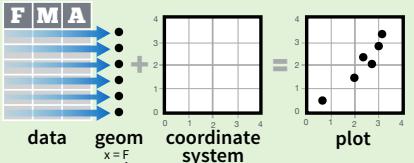
Data Visualization with ggplot2

Cheat Sheet

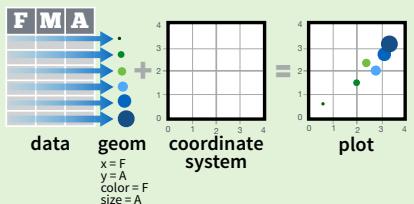


Basics

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



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



Complete the template below to build a graph.

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

Required

Not required, sensible defaults supplied

ggplot(data = mpg, aes(x = cty, y = hwy))

Begins a plot that you finish by adding layers to.
Add one geom function per layer.

aesthetic mappings data geom

qplot(x = cty, y = hwy, data = mpg, geom = "point")

Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

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.

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()
(Useful for expanding limits)

b + geom_curve(aes(yend = lat + 1, xend=long+1, curvature=z)) - 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(group = group))
x, y, alpha, color, fill, group, 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

Continuous X, Continuous Y

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

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE)
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_jitter(height = 2, width = 2)
x, y, alpha, color, fill, shape, size

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, check_overlap = TRUE)
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Discrete X, Continuous Y

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

Discrete X, Discrete Y

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

g + geom_count()
x, y, alpha, color, fill, shape, size, stroke

Three Variables

seals\$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))

l <- ggplot(seals, aes(long, lat))

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

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

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_density2d()
x, y, alpha, colour, group, linetype, size

h + geom_hex()
x, y, alpha, colour, 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

data <- data.frame(murder = USArrests\$Murder, state = tolower(rownames(USArrests)))

map <- map_data("state")

k <- ggplot(data, aes(fill = murder))

k + geom_map(aes(map_id = state), map = map) + expand_limits(x = map\$long, y = map\$lat)
map_id, alpha, color, fill, linetype, size

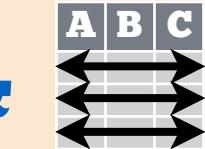
Data Transformation with dplyr Cheat Sheet



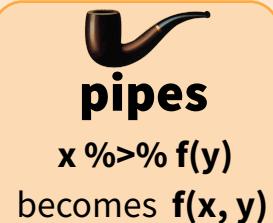
dplyr functions work with pipes and expect **tidy data**. In tidy data:



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



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



Summarise Cases

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



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

count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in ... Also **tally()**.
`count(iris, Species)`

Variations

- **summarise_all()** - Apply funs to every column.
- **summarise_at()** - Apply funs to specific columns.
- **summarise_if()** - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to created a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

`mtcars %>% group_by(cyl) %>% summarise(avg = mean(mpg))`

group_by(.data, ..., add = FALSE)
Returns copy of table grouped by ...
`g_iris <- group_by(iris, Species)`

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

Manipulate Cases

Extract Cases

Row functions return a subset of rows as a new table. Use a variant that ends in _ for non-standard evaluation friendly code.

filter(.data, ...)
Extract rows that meet logical criteria. Also **filter_()**. `filter(iris, Sepal.Length > 7)`

distinct(.data, ..., .keep_all = FALSE)
Remove rows with duplicate values. Also **distinct_()**. `distinct(iris, Species)`

sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame())
Randomly select fraction of rows.
`sample_frac(iris, 0.5, replace = TRUE)`

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame())
Randomly select size rows.
`sample_n(iris, 10, replace = TRUE)`

slice(.data, ...)
Select rows by position. Also **slice_()**.
`slice(iris, 10:15)`

top_n(x, n, wt)
Select and order top n entries (by group if grouped data). `top_n(iris, 5, Sepal.Width)`

Logical and boolean operators to use with filter()

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

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

Arrange Cases

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

Add Cases
add_row(.data, ..., .before = NULL, .after = NULL)
Add one or more rows to a table.
`add_row(faithful, eruptions = 1, waiting = 1)`

Manipulate Variables

Extract Variables

Column functions return a set of columns as a new table. Use a variant that ends in _ for non-standard evaluation friendly code.

select(.data, ...)
Extract columns by name. Also **select_if()**.
`select(iris, Sepal.Length, Species)`

Use these helpers with select(), e.g. `select(iris, starts_with("Sepal"))`

`contains`(match)
`ends_with`(match)
`matches`(match)

`num_range`(prefix, range)
`one_of`(...)
`starts_with`(match)

Make New Variables

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

mutate(.data, ...)
Compute new column(s).
`mutate(mtcars, gpm = 1/mpg)`

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

mutate_all(.tbl, .funs, ...)
Apply funs to every column. Use with **funs()**.
`mutate_all(faithful, funs(log(.), log2(.)))`

mutate_at(.tbl, .cols, .funs, ...)
Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for **select()**.
`mutate_at(iris, vars(-Species), funs(log(.)))`

mutate_if(.tbl, .predicate, .funs, ...)
Apply funs to all columns of one type. Use with **funs()**.
`mutate_if(iris, is.numeric, funs(log(.)))`

add_column(.data, ..., .before = NULL, .after = NULL)
Add new column(s).
`add_column(mtcars, new = 1:32)`

rename(.data, ...)
Rename columns.
`rename(iris, Length = Sepal.Length)`

Vectorized Functions

to use with mutate()

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



Offsets

`dplyr::lag()` - Offset elements by 1
`dplyr::lead()` - Offset elements by -1

Cumulative Aggregates

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

Rankings

`dplyr::cume_dist()` - Proportion of all values <=
`dplyr::dense_rank()` - rank with 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

Misc

`dplyr::between()` - $x \geq \text{left} \& x \leq \text{right}$
`dplyr::case_when()` - multi-case if_else()
`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()
`dplyr::recode()` - Vectorized switch()
`dplyr::recode_factor()` - Vectorized switch() for factors

Summary Functions

to use with summarise()

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



Counts

`dplyr::n()` - number of values/rows
`dplyr::n_distinct()` - # of uniques
`sum(!is.na())` - # of non-NAs

Location

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

Logicals

`mean()` - Proportion of TRUE's
`sum()` - # of TRUE's

Position/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()` - mean absolute deviation
`sd()` - standard deviation
`var()` - variance

Row names

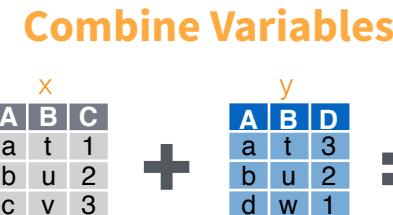
Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

`rownames_to_column()`
Move row names into col.
`a <- rownames_to_column(iris, var = "C")`

`column_to_rownames()`
Move col in row names.
`column_to_rownames(a, var = "C")`

Also `has_rownames()`, `remove_rownames()`

Combine Tables



Use **bind_cols()** to paste tables beside each other as they are.

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

bind_cols(...)

Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

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"), ...)`
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"), ...)`
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"), ...)`
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	4	1

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

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 the column(s) 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 with different names in each data set.

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 suffix to give to duplicate column names.

left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

Combine Cases

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

A	B	C
c	v	3
d	w	4

Use **bind_rows()** to paste tables below each other as they are.

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

A	B	C
c	v	3

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



A	B	C
a	t	1
b	u	2

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



A	B	C

<tbl_r cells="3" ix="1"