

# RStudio IDE :: CHEAT SHEET

## Documents and Apps



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

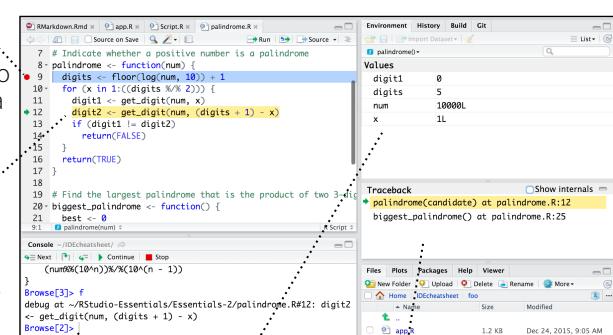
## Debug Mode

Open with **debug()**, **browser()**, 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

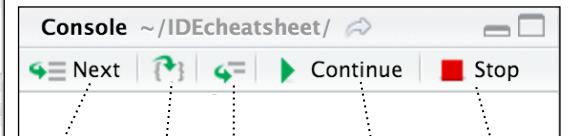


Examine variables in executing environment

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



Step through code one line at a time

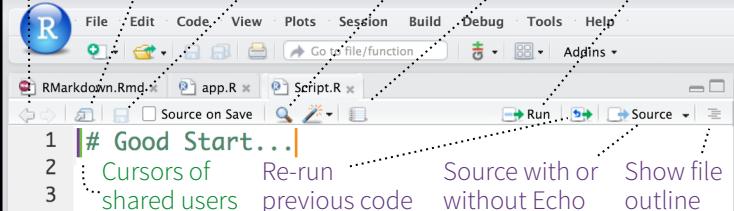
Step into and out of functions to run

Resume execution mode

Quit debug

## 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.

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

Jump to function in file   Change file type

Working Directory   Maximize, minimize panes

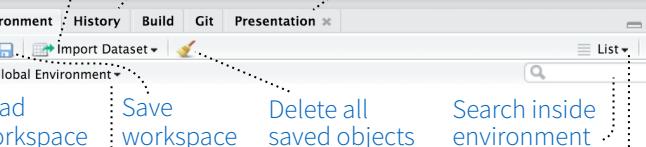
Press **↑** to see command history   Drag pane boundaries

## R Support

**Import data** with wizard

History of past commands to run/copy

Display .RPres slideshows **File > New File > R Presentation**



Choose environment to display from list of parent environments

Display objects as list or grid

Data   150 obs. of 5 variables   Values   Functions

a   1   foo   function (x)

Displays saved objects by type with short description

View in data viewer   View function source code



Path to displayed directory

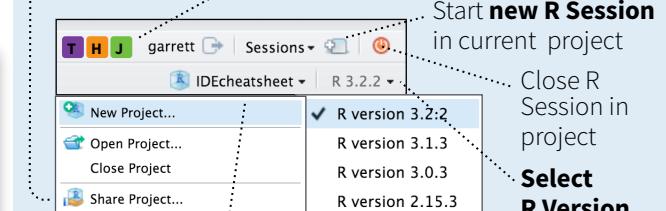
Maximize, minimize panes

Drag pane boundaries

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

## Pro Features

**Share Project** Active shared with Collaborators



Start **new R Session** in current project

Close R Session in project

**Select R Version**

### PROJECT SYSTEM **File > New Project**

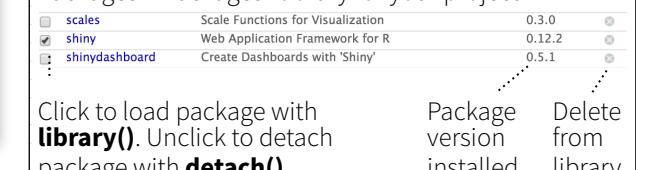
RStudio saves the call history, workspace, and working directory associated with a project. It reloads each when you re-open a project.



RStudio opens plots in a dedicated Plots pane

Open in recent plots   Export plot   Delete plot   Delete all plots

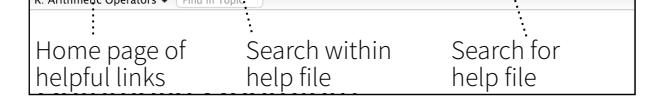
GUI Package manager lists every installed package



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

Package version installed   Delete from library

RStudio opens documentation in a dedicated Help pane



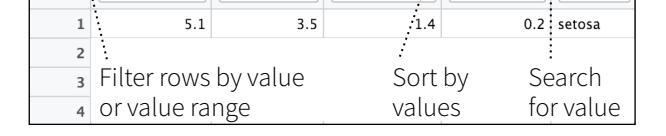
Home page of helpful links   Search within help file   Search for help file

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 rows by value or value range   Sort by values   Search for value



## 1 LAYOUT

Move focus to Source Editor  
Move focus to Console  
Move focus to Help  
Show History  
Show Files  
Show Plots  
Show Packages  
Show Environment  
Show Git/SVN  
Show Build

**Windows/Linux** **Mac**  
Ctrl+1 Ctrl+1  
Ctrl+2 Ctrl+2  
Ctrl+3 Ctrl+3  
Ctrl+4 Ctrl+4  
Ctrl+5 Ctrl+5  
Ctrl+6 Ctrl+6  
Ctrl+7 Ctrl+7  
Ctrl+8 Ctrl+8  
Ctrl+9 Ctrl+9  
Ctrl+0 Ctrl+0

## 2 RUN CODE

### Search command history

Navigate command history  
Move cursor to start of line  
Move cursor to end of line  
Change working directory

### Interrupt current command

### Clear console

Quit Session (desktop only)

### Restart R Session

Run current line/selection  
Run current (retain cursor)  
Run from current to end  
Run the current function  
Source a file

### Source the current file

Source with echo

**Windows/Linux** **Mac**

**Ctrl+↑** **Cmd+↑**  
**↑/↓** **↑/↓**  
Home **Cmd+←**  
End **Cmd+→**  
Ctrl+Shift+H **Ctrl+Shift+H**

**Esc** **Esc**

**Ctrl+L** **Ctrl+L**

Ctrl+Q **Cmd+Q**

**Ctrl+Shift+F10** **Cmd+Shift+F10**

**Ctrl+Enter** **Cmd+Enter**  
Alt+Enter Option+Enter  
Ctrl+Alt+E Cmd+Option+E  
Ctrl+Alt+F Cmd+Option+F  
Ctrl+Alt+G Cmd+Option+G

**Ctrl+Shift+S** **Cmd+Shift+S**

Ctrl+Shift+Enter Cmd+Shift+Enter

## 4 WRITE CODE

### Attempt completion

Navigate candidates  
Accept candidate  
Dismiss candidates  
Undo  
Redo  
Cut  
Copy  
Paste  
Select All  
Delete Line

Select  
Select Word

Select to Line Start  
Select to Line End

Select Page Up/Down

Select to Start/End

Delete Word Left

Delete Word Right

Delete to Line End

Delete to Line Start

Indent

Outdent

Yank line up to cursor

Yank line after cursor

Insert yanked text

**Insert <->**

**Insert %>%**

Show help for function

Show source code

New document

New document (Chrome)

Open document

Save document

Close document

Close document (Chrome)

Close all documents

Extract function

Extract variable

Reindent lines

**(Un)Comment lines**

Reflow Comment

Reformat Selection

Select within braces

Show Diagnostics

Transpose Letters

Move Lines Up/Down

Copy Lines Up/Down

Add New Cursor Above

Add New Cursor Below

Move Active Cursor Up

Move Active Cursor Down

Find and Replace

Use Selection for Find

Replace and Find

## Windows /Linux

### Tab or Ctrl+Space

**↑/↓**  
Enter, Tab, or →  
Esc  
Ctrl+Z  
Ctrl+Shift+Z  
Ctrl+X  
Ctrl+C  
Ctrl+V  
Ctrl+A  
Ctrl+D  
Shift+[Arrow]  
Ctrl+Shift+←/→  
Alt+Shift+←  
Alt+Shift+→  
Shift+PageUp/Down  
Shift+Alt+↑/↓  
Ctrl+Backspace  
Tab (at start of line)  
Shift+Tab  
Ctrl+U  
Ctrl+K  
Ctrl+Y  
Alt+-  
Ctrl+Shift+M  
F1  
F2  
Cmd+Shift+N  
Ctrl+Alt+Shift+N  
Ctrl+O  
Ctrl+S  
Ctrl+W  
Ctrl+Alt+W  
Ctrl+Shift+W  
Ctrl+Alt+X  
Ctrl+Alt+V  
Ctrl+I  
Ctrl+Shift+C  
Ctrl+Shift+/  
Ctrl+Shift+A  
Ctrl+Shift+E  
Ctrl+Shift+P  
Ctrl+T  
Option+↑/↓  
Shift+Alt+↑/↓  
Cmd+Option+↑/↓  
Ctrl+Option+Up  
Ctrl+Option+Down  
Ctrl+Option+Shift+Up  
Ctrl+Opt+Shift+Down  
Ctrl+F  
Ctrl+F3  
Ctrl+Shift+J  
Cmd+F  
Cmd+E  
Cmd+Shift+J

## Mac

### Tab or Cmd+Space

**↑/↓**  
Enter, Tab, or →  
Esc  
Cmd+Z  
Cmd+Shift+Z  
Cmd+X  
Cmd+C  
Cmd+V  
Cmd+A  
Cmd+D  
Shift+[Arrow]  
Option+Shift+←/→  
Cmd+Shift+←  
Cmd+Shift+→  
Shift+PageUp/Down  
Cmd+Shift+↑/↓  
Ctrl+Opt+Backspace  
Option+Delete  
Ctrl+K  
Option+Backspace  
Tab (at start of line)  
Shift+Tab  
Ctrl+U  
Ctrl+K  
Ctrl+Y  
Option+-  
Cmd+Shift+M  
F1  
F2  
Cmd+Shift+N  
Cmd+Shift+Opt+N  
Cmd+O  
Ctrl+S  
Cmd+W  
Cmd+Option+W  
Cmd+Shift+W  
Cmd+Option+X  
Cmd+Option+V  
Cmd+I  
Cmd+Shift+C  
Cmd+Shift+/  
Cmd+Shift+A  
Cmd+Shift+E  
Cmd+Shift+P  
Ctrl+T  
Option+↑/↓  
Shift+Alt+↑/↓  
Cmd+Option+↑/↓  
Ctrl+Option+Up  
Ctrl+Option+Down  
Ctrl+Option+Shift+Up  
Ctrl+Opt+Shift+Down  
Ctrl+F  
Cmd+E  
Cmd+Shift+J

## WHY RSTUDIO SERVER PRO?

RSP extends 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
- edit the same project at the same time as others
- see what you and others are doing on your server
- switch easily from one version of R to a different version
- integrate with your authentication, authorization, and audit practices

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

## 5 DEBUG CODE

<b>Windows/Linux</b>	<b>Mac</b>
Toggle Breakpoint	Shift+F9
Execute Next Line	F10
Step Into Function	Shift+F4
Finish Function/Loop	Shift+F6
Continue	Shift+F5
Stop Debugging	Shift+F8

## 6 VERSION CONTROL

<b>Windows/Linux</b>	<b>Mac</b>
Show diff	Ctrl+Alt+D
Commit changes	Ctrl+Alt+M
Scroll diff view	Ctrl+↑/↓
Stage/Unstage (Git)	Spacebar
Stage/Unstage and move to next	Enter

## 7 MAKE PACKAGES

<b>Windows/Linux</b>	<b>Mac</b>
Build and Reload	Ctrl+Shift+B
<b>Load All (devtools)</b>	<b>Cmd+Shift+L</b>
<b>Test Package (Desktop)</b>	<b>Cmd+Shift+T</b>
Test Package (Web)	Ctrl+Alt+F7
Check Package	Ctrl+Shift+E
<b>Document Package</b>	<b>Cmd+Shift+D</b>

## 8 DOCUMENTS AND APPS

<b>Windows/Linux</b>	<b>Mac</b>
Preview HTML (Markdown, etc.)	Ctrl+Shift+K
<b>Knit Document (knitr)</b>	<b>Cmd+Shift+K</b>
Compile Notebook	Ctrl+Shift+K
Compile PDF (TeX and Sweave)	Ctrl+Shift+K
Insert chunk (Sweave and Knitr)	Ctrl+Alt+I
Insert code section	Ctrl+Shift+R
Re-run previous region	Ctrl+Shift+P
Run current document	Ctrl+Alt+R
<b>Run from start to current line</b>	<b>Cmd+Option+B</b>
<b>Run the current code section</b>	<b>Cmd+Option+T</b>
Run previous Sweave/Rmd code	Ctrl+Alt+P
Run the current chunk	Ctrl+Alt+C
Run the next chunk	Ctrl+Alt+N
Sync Editor & PDF Preview	Ctrl+F8
Previous plot	Ctrl+Alt+F11
Next plot	Ctrl+Alt+F12
<b>Show Keyboard Shortcuts</b>	<b>Alt+Shift+K</b>

# 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>nrow(df)</code> Number of rows.	<code>cbind</code> - Bind columns.
<code>df[2, ]</code>		<code>ncol(df)</code> Number of columns.	<code>rbind</code> - Bind rows.
<code>df[2, 2]</code>		<code>dim(df)</code> Number of columns and 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>t.test(x, y)</code>	Test for a difference between proportions.
<code>glm(y ~ x, data=df)</code>	Generalised linear model.	<code>pairwise.t.test</code>	Perform a t-test for paired data.
<code>summary</code>	Get more detailed information out a model.	<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>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 :: CHEAT SHEET

R's **tidyverse** is built around **tidy data** stored in **tibbles**, which are enhanced data frames.

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)

## Save Data

Save **x**, an R object, to **path**, a file path, as:

### Comma delimited file

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

### File with arbitrary delimiter

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

### CSV for excel

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

### String to file

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

### String vector to file, one element per line

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

### Object to RDS file

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

### Tab delimited files

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



## Read Tabular Data

- 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())
```

### Comma Delimited Files

```
read_csv("file.csv")
```

To make file.csv run:

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

### Semi-colon Delimited Files

```
read_csv2("file2.csv")
```

```
write_file(x = "a;b;c\n1;2;3\n4;5;NA", path = "file2.csv")
```

### Files with Any Delimiter

```
read_delim("file.txt", delim = "|")
```

```
write_file(x = "a|b|c\n1|2|3\n4|5|NA", path = "file.txt")
```

### Fixed Width Files

```
read_fwf("file.fwf", col_positions = c(1, 3, 5))
```

```
write_file(x = "a b c\n1 2 3\n4 5 NA", path = "file.fwf")
```

## USEFUL ARGUMENTS

### Example file

```
write_file("a,b,c\n1,2,3\n4,5,NA","file.csv")  
f <- "file.csv"
```

### Skip lines

```
read_csv(f, skip = 1)
```

### No header

```
read_csv(f, col_names = FALSE)
```

### Read in a subset

```
read_csv(f, n_max = 1)
```

### Provide header

```
read_csv(f, col_names = c("x", "y", "z"))
```

### Missing Values

```
read_csv(f, na = c("1", "?"))
```

## Read Non-Tabular Data

### Read a file into a single string

```
read_file(file, locale = default_locale())
```

### Read each line into its own string

```
read_lines(file, skip = 0, n_max = -1L, na = character(),  
          locale = default_locale(), progress = interactive())
```

### Read Apache style log files

```
read_log(file, col_names = FALSE, col_types = NULL, skip = 0, n_max = -1, progress = interactive())
```

### Read a file into a raw vector

```
read_file_raw(file)
```

### Read each line into a raw vector

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



## 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

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 = "")**, **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()**
  - **parse\_character()**
  - **parse\_datetime()** Also **parse\_date()** and **parse\_time()**
  - **parse\_double()**
  - **parse\_factor()**
  - **parse\_integer()**
  - **parse\_logical()**
  - **parse\_number()**
- 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 three behaviors:

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

# A tibble: 234 × 6	manufacturer	model	displ	cyl	trans
1 audi	a4	1.80	4	auto(l4)	
2 audi	a4	1.80	4	auto(l4)	
3 audi	a4	2.00	4	auto(l4)	
4 audi	a4	2.00	4	auto(l4)	
5 audi	a4	2.00	4	auto(l4)	
6 audi	a4	2.00	4	auto(l4)	
7 audi	a4	3.10	6	quattro	
8 audi	a4	3.10	6	quattro	
9 audi	a4	3.10	6	quattro	
10 audi	a4	3.10	6	quattro	
... with 224 more rows, and 3 more variables: year <int>, cyl <int>, trans <chr>					

**tibble display**

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

**A large table to display**

**data frame display**

- Control the default appearance with options:  
`options(tibble.print_max = n, tibble.print_min = m, tibble.width = Inf)`
- View full data set with **View()** or **glimpse()**
- Revert to data frame with **as.data.frame()**

## CONSTRUCT A TIBBLE IN TWO WAYS

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

**as\_tibble(x, ...)** Convert data frame to tibble.

**enframe(x, name = "name", value = "value")**  
Convert named vector to a tibble

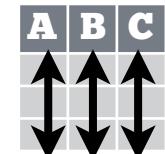
**is\_tibble(x)** Test whether x is a tibble.



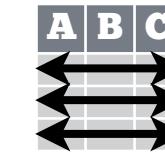
# Tidy Data with tidyverse

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

A table is tidy if:



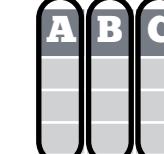
&



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

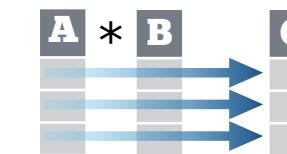
Each **observation**, or **case**, is in its own **row**

Tidy data:



Makes variables easy to access as vectors

$A * B \rightarrow C$



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.

**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")`

table2

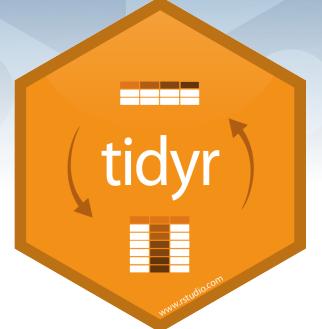
country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
B	1999	cases	37K
B	1999	pop	172M
C	1999	cases	212K
C	1999	pop	1T
A	2000	cases	2K
A	2000	pop	20M
B	2000	cases	80K
B	2000	pop	174M
C	2000	cases	213K
C	2000	pop	1T

key value

`spread(table2, type, count)`

## Split 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

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

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(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	1999	19M
A	2000	2K
A	2000	20M
B	1999	37K/172M
B	1999	37K
B	2000	80K
B	2000	174M
C	1999	212K/1T
C	1999	212K
C	2000	1T
C	2000	213K
C	2000	1T

separate\_rows(table3, rate)

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

**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

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

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

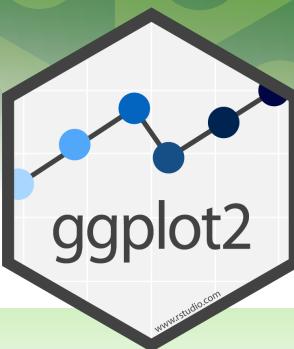
## Handle Missing Values

**drop\_na(data, ...)**

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

x1	x2

# 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, 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(f1))
- 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\_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

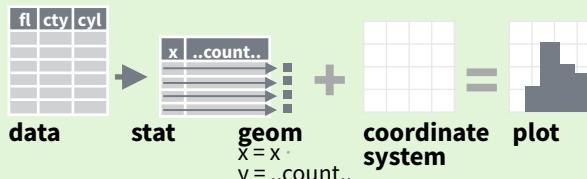
- data <- data.frame(murder = USArrests\$Murder, state = tolower(rownames(USArrests)))  
map <- map\_data("state")  
k <- ggplot(data, aes(fill = murder))

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

# Stats

An alternative way to build a layer

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



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



```

c + stat_bin(binwidth = 1, origin = 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() + stat_function(aes(x = -3:3), n = 99, fun = dnorm, args = list(sd = 0.5)) x | ..x.., ..y..
  
```

```

e + stat_identity(na.rm = TRUE)
ggplot() + stat_qq(aes(sample = 1:100), dist = qt, dparam = list(df = 5)) sample, x, y | ..sample.., ..theoretical..
  
```

```

e + stat_sum() x, y, size | ..n.., ..prop..
  
```

```

e + stat_summary(fun.data = "mean_cl_boot")
  
```

```

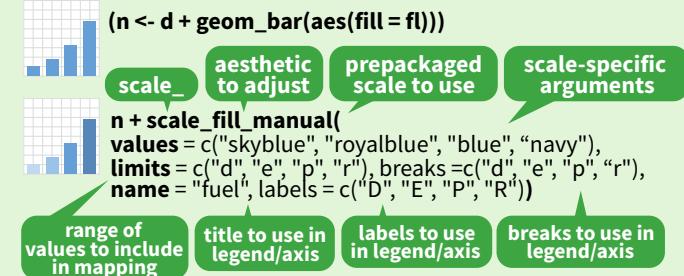
h + stat_summary_bin(fun.y = "mean", geom = "bar")
  
```

```

e + stat_unique()
  
```

# Scales

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



## GENERAL PURPOSE SCALES

Use with most aesthetics

`scale_*_continuous()` - map cont' values to visual ones  
`scale_*_discrete()` - map discrete values to visual ones  
`scale_*_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 x values as date times. Use same arguments as `scale_x_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 direction of x axis  
`scale_x_sqrt()` - Plot x on square root scale

## COLOR AND FILL SCALES (DISCRETE)

`n <- d + geom_bar(aes(fill = fl))`  
`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(colours = 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))`  
The default cartesian coordinate system

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

`r + coord_flip()`  
Flipped Cartesian coordinates

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

`r + coord_trans(xtrans = "sqrt")`  
xtrans, ytrans, limx, limy  
Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.

`π + coord_quickmap()`

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

# Position Adjustments

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

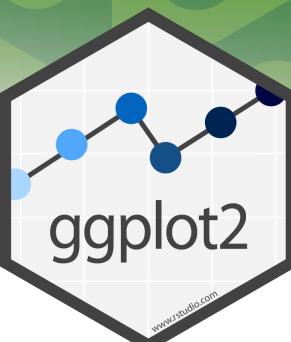
`s <- ggplot(mpg, aes(fl, fill = drv))`  
`s + geom_bar(position = "dodge")`  
Arrange elements side by side  
`s + geom_bar(position = "fill")`  
Stack elements on top of one another, normalize height  
`e + geom_point(position = "jitter")`  
Add random noise to X and Y position of each element to avoid overplotting  
`e + geom_label(position = "nudge")`  
Nudge labels away from points  
`s + geom_bar(position = "stack")`  
Stack elements on top of one another

Each position adjustment can be recast as a function with manual `width` and `height` arguments

`s + geom_bar(position = position_dodge(width = 1))`

# Themes

`r + theme_bw()`  
White background with grid lines  
`r + theme_gray()`  
Grey background (default theme)  
`r + theme_minimal()`  
Minimal themes  
`r + theme_dark()`  
dark for contrast  
`r + theme_void()`  
Empty theme



# Faceting

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

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

`t + facet_grid(cols = vars(fl))`

`t + facet_grid(rows = vars(year))`

`t + facet_grid(rows = vars(year), cols = vars(fl))`

`t + facet_wrap(vars(fl))`

Set scales to let axis limits vary across facets

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

Set `labeler` to adjust facet labels

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

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

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

`αc αd αe αp αr`

# Labels

`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", <AES> = "New <AES> legend title")`

`t + annotate(geom = "text", x = 8, y = 9, label = "A")`

`geom to place manual values for geom's aesthetics`

# Legends

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

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

`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))`

# caret Package

## Cheat Sheet

### Specifying the Model

Possible syntaxes for specifying the variables in the model:

```
train(y ~ x1 + x2, data = dat, ...)
train(x = predictor_df, y = outcome_vector, ...)
train(recipe_object, data = dat, ...)
```

- `rfe`, `sbf`, `gafs`, and `safs` only have the `x/y` interface.
- The `train` formula method will **always** create dummy variables.
- The `x/y` interface to `train` will not create dummy variables (but the underlying model function might).

**Remember** to:

- Have column names in your data.
- Use factors for a classification outcome (not 0/1 or integers).
- Have valid R names for class levels (not "0"/"1")
- Set the random number seed prior to calling `train` repeatedly to get the same resamples across calls.
- Use the `train` option `na.action = na.pass` if you will be imputing missing data. Also, use this option when predicting new data containing missing values.

To pass options to the underlying model function, you can pass them to `train` via the ellipses:

```
train(y ~ ., data = dat, method = "rf",
      # options to `randomForest`:
      importance = TRUE)
```

### Parallel Processing

The `foreach` package is used to run models in parallel. The `train` code does not change but a "`do`" package must be called first.

```
# on Mac OS or Linux      # on Windows
library(doMC)              library(doParallel)
registerDoMC(cores=4)       cl <- makeCluster(2)
                           registerDoParallel(cl)
```

The function `parallel::detectCores` can help too.

### Preprocessing

Transformations, filters, and other operations can be applied to the *predictors* with the `preProc` option.

```
train(..., preProc = c("method1", "method2"), ...)
```

Methods include:

- `"center"`, `"scale"`, and `"range"` to normalize predictors.
- `"BoxCox"`, `"YeoJohnson"`, or `"expoTrans"` to transform predictors.
- `"knnImpute"`, `"bagImpute"`, or `"medianImpute"` to impute.
- `"corr"`, `"nzv"`, `"zv"`, and `"conditionalX"` to filter.
- `"pca"`, `"ica"`, or `"spatialSign"` to transform groups.

`train` determines the order of operations; the order that the methods are declared does not matter.

The `recipes` package has a more extensive list of preprocessing operations.

### Adding Options

Many `train` options can be specified using the `trainControl` function:

```
train(y ~ ., data = dat, method = "cubist",
      trControl = trainControl(<options>))
```

### Resampling Options

`trainControl` is used to choose a resampling method:

```
trainControl(method = <method>, <options>)
```

Methods and options are:

- `"cv"` for K-fold cross-validation (`number` sets the # folds).
- `"repeatedcv"` for repeated cross-validation (`repeats` for # repeats).
- `"boot"` for bootstrap (`number` sets the iterations).
- `"LGOCV"` for leave-group-out (`number` and `p` are options).
- `"L0O"` for leave-one-out cross-validation.
- `"oob"` for out-of-bag resampling (only for some models).
- `"timeslice"` for time-series data (options are `initialWindow`, `horizon`, `fixedWindow`, and `skip`).

### Performance Metrics

To choose how to summarize a model, the `trainControl` function is used again.

```
trainControl(summaryFunction = <R function>,
             classProbs = <logical>)
```

Custom R functions can be used but `caret` includes several: `defaultSummary` (for accuracy, RMSE, etc), `twoClassSummary` (for ROC curves), and `prSummary` (for information retrieval). For the last two functions, the option `classProbs` must be set to `TRUE`.

### Grid Search

To let `train` determine the values of the tuning parameter(s), the `tuneLength` option controls how many values `per tuning` parameter to evaluate.

Alternatively, specific values of the tuning parameters can be declared using the `tuneGrid` argument:

```
grid <- expand.grid(alpha = c(0.1, 0.5, 0.9),
                      lambda = c(0.001, 0.01))
```

```
train(x = x, y = y, method = "glmnet",
      preProc = c("center", "scale"),
      tuneGrid = grid)
```

### Random Search

For tuning, `train` can also generate random tuning parameter combinations over a wide range. `tuneLength` controls the total number of combinations to evaluate. To use random search:

```
trainControl(search = "random")
```

### Subsampling

With a large class imbalance, `train` can subsample the data to balance the classes them prior to model fitting.

```
trainControl(sampling = "down")
```

Other values are `"up"`, `"smote"`, or `"rose"`. The latter two may require additional package installs.

# String manipulation with stringr :: CHEAT SHEET



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

	<code>str_detect(string, pattern)</code> Detect the presence of a pattern match in a string. <code>str_detect(fruit, "a")</code>
	<code>str_which(string, pattern)</code> Find the indexes of strings that contain a pattern match. <code>str_which(fruit, "a")</code>
	<code>str_count(string, pattern)</code> Count the number of matches in a string. <code>str_count(fruit, "a")</code>
	<code>str_locate(string, pattern)</code> Locate the positions of pattern matches in a string. Also <code>str_locate_all</code> . <code>str_locate(fruit, "a")</code>

## Subset Strings

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

## Manage Lengths

	<code>str_length(string)</code> The width of strings (i.e. number of code points, which generally equals the number of characters). <code>str_length(fruit)</code>
	<code>str_pad(string, width, side = c("left", "right", "both"), pad = " ")</code> Pad strings to constant width. <code>str_pad(fruit, 17)</code>
	<code>str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "...")</code> Truncate the width of strings, replacing content with ellipsis. <code>str_trunc(fruit, 3)</code>
	<code>str_trim(string, side = c("both", "left", "right"))</code> Trim whitespace from the start and/or end of a string. <code>str_trim(fruit)</code>

## Mutate Strings

	<code>str_sub()</code> <- value. Replace substrings by identifying the substrings with <code>str_sub()</code> and assigning into the results. <code>str_sub(fruit, 1, 3) &lt;- "str"</code>
	<code>str_replace(string, pattern, replacement)</code> Replace the first matched pattern in each string. <code>str_replace(fruit, "a", "-")</code>
	<code>str_replace_all(string, pattern, replacement)</code> Replace all matched patterns in each string. <code>str_replace_all(fruit, "a", "-")</code>
	<code>str_to_lower(string, locale = "en")<sup>1</sup></code> Convert strings to lower case. <code>str_to_lower(sentences)</code>
	<code>str_to_upper(string, locale = "en")<sup>1</sup></code> Convert strings to upper case. <code>str_to_upper(sentences)</code>
	<code>str_to_title(string, locale = "en")<sup>1</sup></code> Convert strings to title case. <code>str_to_title(sentences)</code>

## Join and Split

	<code>str_c(..., sep = "", collapse = NULL)</code> Join multiple strings into a single string. <code>str_c(letters, LETTERS)</code>
	<code>str_c(..., sep = "", collapse = NULL)</code> Collapse a vector of strings into a single string. <code>str_c(letters, collapse = "")</code>
	<code>str_dup(string, times)</code> Repeat strings times times. <code>str_dup(fruit, times = 2)</code>
	<code>str_split_fixed(string, pattern, n)</code> Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match). Also <code>str_split</code> to return a list of substrings. <code>str_split_fixed(fruit, " ", n=2)</code>
	<code>str_glue(..., .sep = "", .envir = parent.frame())</code> Create a string from strings and {expressions} to evaluate. <code>str_glue("Pi is {pi}")</code>
	<code>str_glue_data(.x, ..., .sep = "", .envir = parent.frame(), .na = "NA")</code> Use a data frame, list, or environment to create a string from strings and {expressions} to evaluate. <code>str_glue_data(mtcars, "{rownames(mtcars)} has {hp} hp")</code>

## Order Strings

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

## Helpers

	<code>str_conv(string, encoding)</code> Override the encoding of a string. <code>str_conv(fruit, "ISO-8859-1")</code>
	<code>str_view(string, pattern, match = NA)</code> View HTML rendering of first regex match in each string. <code>str_view(fruit, "[aeiou]")</code>
	<code>str_view_all(string, pattern, match = NA)</code> View HTML rendering of all regex matches. <code>str_view_all(fruit, "[aeiou]")</code>
	<code>str_wrap(string, width = 80, indent = 0, exdent = 0)</code> Wrap strings into nicely formatted paragraphs. <code>str_wrap(sentences, 20)</code>

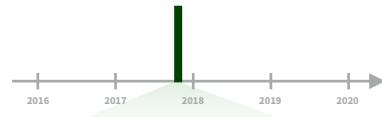
<sup>1</sup> See [bit.ly/ISO639-1](http://bit.ly/ISO639-1) for a complete list of locales.



# Dates and times with lubridate :: CHEAT SHEET



## Date-times



2017-11-28 12:00:00

2017-11-28 12:00:00

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

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

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

1. Identify the order of the year (**y**), month (**m**), day (**d**), hour (**h**), minute (**m**) and second (**s**) elements in your data.
2. Use the function below whose name replicates the order. Each accepts a wide variety of input formats.

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")

2:01

hms::hms() Also lubridate::hms(), hm() and ms(), which return periods.\* hms::hms(sec = 0, min = 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 strftime.  
fast.strptime('9/1/01', '%y/%m/%d')

parse\_date\_time() Easier strftime.  
parse\_date\_time("9/1/01", "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"
```

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
```

### 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

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

2018-01-31 11:59:59

quarter(x, with\_year = FALSE)
Quarter. quarter(dt)

2018-01-31 11:59:59

semester(x, with\_year = FALSE)
Semester. semester(dt)

2018-01-31 11:59:59

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

2018-01-31 11:59:59

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

2018-01-31 11:59:59

leap\_year(x) Is it a leap year?
leap\_year(dt)

2018-01-31 11:59:59

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

## Round Date-times



**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")

**rollback**(dates, roll\_to\_first = FALSE, preserve\_hms = TRUE)  
Roll back to last day of previous month. **rollback**(dt)

**Tip:** use a date with day > 12

## 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"

## 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()**

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

PT MT CT ET  
7:00 Pacific 7:00 Mountain 7:00 Central

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

**with\_tz**(time, tzone = "") Get the same date-time in a new time zone (a new clock time). **with\_tz**(dt, "US/Pacific")

**force\_tz**(time, tzone = "") Get the same clock time in a new time zone (a new date-time). **force\_tz**(dt, "US/Pacific")





# Math with Date-times

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

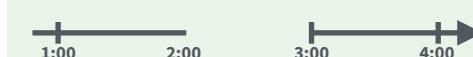
A normal day

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



The start of daylight savings (spring forward)

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



The end of daylight savings (fall back)

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



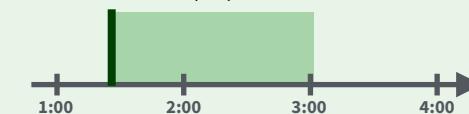
Leap years and leap seconds

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

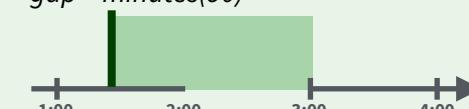


**Periods** track changes in clock times, which ignore time line irregularities.

```
nor + minutes(90)
```



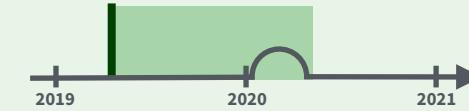
```
gap + minutes(90)
```



```
lap + minutes(90)
```

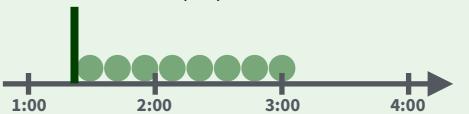


```
leap + years(1)
```

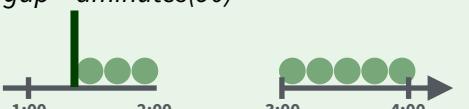


**Durations** track the passage of physical time, which deviates from clock time when irregularities occur.

```
nor + dminutes(90)
```



```
gap + dminutes(90)
```



```
lap + dminutes(90)
```



```
leap + dyears(1)
```



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

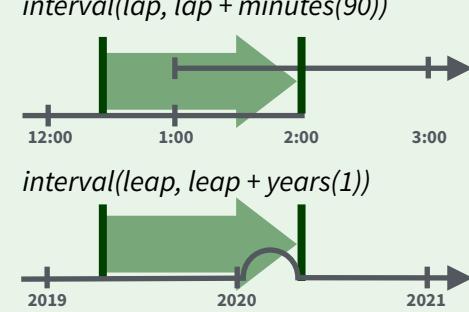
```
interval(nor, nor + minutes(90))
```



```
interval(gap, gap + minutes(90))
```



```
interval(lap, lap + minutes(90))
```



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)
```

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

Number of months Number of days etc.

```
years(x = 1) x years.
```

```
months(x) x months.
```

```
weeks(x = 1) x weeks.
```

```
days(x = 1) x days.
```

```
hours(x = 1) x hours.
```

```
minutes(x = 1) x minutes.
```

```
seconds(x = 1) x seconds.
```

```
milliseconds(x = 1) x milliseconds.
```

```
microseconds(x = 1) x microseconds
```

```
nanoseconds(x = 1) x nanoseconds.
```

```
picoseconds(x = 1) x picoseconds.
```

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

An automation friendly period constructor.

```
period(5, unit = "years")
```

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

Also **is.period**(). **as.period**(i)

**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.

**Difftimes** 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)
```

"1209600s (~2 weeks)"

Exact length in seconds  
Equivalent in common units

```
dyears(x = 1) 31536000x seconds.
```

```
dweeks(x = 1) 604800x seconds.
```

```
ddays(x = 1) 86400x seconds.
```

```
dhours(x = 1) 3600x seconds.
```

```
dminutes(x = 1) 60x seconds.
```

```
dseconds(x = 1) x seconds.
```

```
dmilliseconds(x = 1) x × 10-3 seconds.
```

```
dmicroseconds(x = 1) x × 10-6 seconds.
```

```
dnanoseconds(x = 1) x × 10-9 seconds.
```

```
dpicoseconds(x = 1) x × 10-12 seconds.
```

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

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

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

**make\_difftime**(x) Make difftime with the specified number of units.

```
make_difftime(99999)
```

## INTERVALS

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

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

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

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

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

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

Start Date End Date  
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)

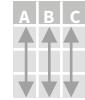
**int\_shift**(int, by) Shifts an interval up or down the timeline by a timespan. **int\_shift**(i, days(-1))

**as.interval**(x, start, ...) Coerce a timespans to an interval with the start date-time. Also **is.interval**(). **as.interval**(days(1), start = now())

# Data Transformation with dplyr :: CHEAT SHEET



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



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



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



`x %>% f(y)` becomes `f(x, y)`

## Summarise Cases

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



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

`count(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 create 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.



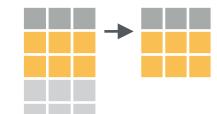
`filter(.data, ...)` Extract rows that meet logical criteria. `filter(iris, Sepal.Length > 7)`



`distinct(.data, ..., .keep_all = FALSE)` Remove rows with duplicate values.  
`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)`



`slice(.data, ...)` Select rows by position.  
`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)`

## Manipulate Variables

### EXTRACT VARIABLES

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



`pull(.data, var = -1)` Extract column values as a vector. Choose by name or index.  
`pull(iris, Sepal.Length)`



`select(.data, ...)` Extract columns as a table. Also `select_if()`.  
`select(iris, Sepal.Length, Species)`

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

`contains(match)`    `num_range(prefix, range)` : e.g. `mpg:cyl`  
`ends_with(match)`    `one_of(...)`    -, e.g. `-Species`  
`matches(match)`    `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()`. Also `mutate_if()`.  
`mutate_all(faithful, funs(log(.), log2(.)))`  
`mutate_if(iris, is.numeric, funs(log(.)))`

`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(.)))`

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

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



# Vector Functions

## TO USE WITH MUTATE ()

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

vectorized function →

## 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  
dplyr::between() - x >= left & x <= right  
dplyr::near() - safe == for floating point numbers

## MISC

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.

summary function →

## COUNTS

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

## 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

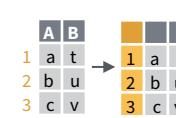
**quantile()** - nth quantile  
**min()** - minimum value  
**max()** - maximum value

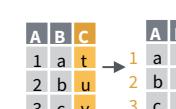
## SPREAD

**IQR()** - Inter-Quartile Range  
**mad()** - 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.

 **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

## COMBINE VARIABLES

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

Use **bind\_cols()** to paste tables beside each other as they are.

**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	<b>left_join(x, y, by = NULL,</b> 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	<b>right_join(x, y, by = NULL, copy =</b> FALSE, suffix=c("x","y"),...) Join matching values from x to y.
---	---

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

A B C D a t 1 3 b u 2 2 d w NA 1	<b>full_join(x, y, by = NULL,</b> copy=FALSE, suffix=c("x","y"),...) Join data. Retain all values, all rows.
---	--

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

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"))**

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	y	=
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.

<b>bind_rows(..., .id = NULL)</b>	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)
-----------------------------------	--

<b>intersect(x, y, ...)</b>	Rows that appear in both x and y.
-----------------------------	-----------------------------------

<b>setdiff(x, y, ...)</b>	Rows that appear in x but not y.
---------------------------	----------------------------------

<b>union(x, y, ...)</b>	Rows that appear in x or y. (Duplicates removed). <b>union_all()</b> retains duplicates.
-------------------------	---

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

## EXTRACT ROWS

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

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

<b>semi_join(x, y, by = NULL, ...)</b>	Return rows of x that have a match in y. USEFUL TO SEE WHAT WILL BE JOINED.
--	--

<b>anti_join(x, y, by = NULL, ...)</b>	Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.
--	--

# h2o:: CHEAT SHEET

## Dataset Operations

### DATA IMPORT / EXPORT

**h2o.uploadFile:** Upload a file into H2O from a client-side path, and parse it.

**h2o.downloadCSV:** Download a H2O dataset to a client-side CSV file.

**h2o.importFile:** Import a file into H2O from a server-side path, and parse it.

**h2o.exportFile:** Export an H2O Data Frame to a server-side file.

**h2o.parseRaw:** Parse a raw data file.

### NATIVE R TO H2O COERCION

**as.h2o:** Convert a R object to an H2O object

### H2O TO NATIVE R COERCION

**as.data.frame:** Check if an object is a data frame, and coerce it if possible.

### DATA GENERATION

**h2o.createFrame:** Creates a data frame in H2O with real-valued, categorical, integer, and binary columns specified by the user, with optional randomization.

**h2o.runif:** Produce a vector of random uniform numbers.

**h2o.interaction:** Create interaction terms between categorical features of an H2O Frame.

**h2o.target\_encode\_apply:** Target encoding map to an H2O Data Frame, which can improve performance of supervised learning models for high cardinality categorical columns.

### DATA SAMPLING / SPLITTING

**h2o.splitFrame:** Split an existing H2O dataset according to user-specified ratios.

### MISSING DATA HANDLING

**h2o.impute:** Impute a column of data using the mean, median, or mode.

**h2o.insertMissingValues:** Replaces a user-specified fraction of entries in an H2O dataset with missing values.

**h2o.na.omit:** Remove Rows With NAs.

## General Operations

### SUBSCRIPTING

Subscripting example to pull (/push) pieces from (/to) a H2O Parsed Data object.

x[j] ## column J	x[i]	<- value	Value Assignment
x[i, j]	x[i, j, ...]	<- value	
x[[i]]	x[[i]]	<- value	
x\$name	x\$i	<- value	

### Selection

### Value Assignment

### SUBSETTING

**h2o.head, h2o.tail:** Object's Start or End.

### DATA ATTRIBUTES

**h2o.names:** Return column names for an H2O Frame. Also: **h2o.colnames**

**names<-:** Set the row or column names of a H2O Frame. Also: **colnames<-**

**h2o.dim:** Retrieve object dimensions.

**h2o.length:** Length of vector, list or factor.

**h2o.nrow:** Number of H2O Frame rows.

**h2o.ncol:** Number of H2O Frame columns.

**h2o.anyFactor:** Check if an H2O Frame object has any categorical data columns.

**is.factor, is.character, is.numeric:** Check Column's Data Type.

### DATA TYPE COERCION:

**h2o.asfactor, as.factor:** Factor.

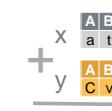
**h2o.as\_date, as.Date:** Date.

**h2o.ascharacter, as.character:** Character.

**h2o.asnumeric, as.numeric:** Numeric.

### BASIC DATA MANIPULATION

**c:** Combine Values into a Vector or List.

 **h2o.cbind; h2o.rbind:** Combine a sequence of H2O datasets by column (cbind) or rows (rbind).

 **h2o.merge:** Merges 2 H2OFrames.

 **h2o.arrange:** Sorts an H2OFrame by columns.

### ELEMENT INDEX SELECTION

**h2o.which:** True Condition's Row Numbers

### CONDITIONAL VALUE SELECTION

**h2o.ifelse:** Apply conditional statements to numeric vectors in an H2O Frame.

## Math Operations

### (math) vectorized function

### MATH

**h2o.abs:** Compute the absolute value of x.

**h2o.sqrt:** Principal Square Root of x,  $\sqrt{x}$ .

**h2o.ceiling:** Take a single numeric argument x and return a numeric vector containing the smallest integers not less than the corresponding elements of x.

**h2o.floor:** Take a single numeric argument x and return a numeric vector containing the largest integers not greater than the corresponding elements of x.

**h2o.trunc:** Take a single numeric argument x and return a numeric vector containing the integers formed by truncating the values in x toward 0.

**h2o.log:** Compute natural logarithms. See also: **h2o.log10, h2o.log2, h2o.log1p**

**h2o.exp:** Compute the exponential function

**h2o.cos, h2o.cosh, h2o.acos, h2o.sin, h2o.tan, h2o.tanh, Math:** ?groupGeneric

**sign:** Return a vector with the signs of the corresponding elements of x (the sign of a real number is 1, 0, or -1 if the number is positive, zero, or negative, respectively).

**&& (Vectorized AND), || (Vectorized OR), !x, %in%, Ops: +, -, \*, /, ^, %%, %/%, ==, !=, <, <=, >=, >, &, |, !**

### CUMULATIVE

**h2o.cummax:** Vector of the cumulative maxima of the elements of the argument.

**h2o.cummin:** Vector of the cumulative minima of the elements of the argument.

**h2o.cumprod:** Vector of the cumulative products of the elements of the argument.

**h2o.cumsum:** Vector of the cumulative sums of the elements of the argument.

### PRECISION

**h2o.round:** Round values to the specified number of decimal places. The default is 0.

**h2o.signif:** Round values to the specified number of significant digits.

## Group By Summaries

### (group by) summary function

**nrow:** Count the number of rows.

**max:** All input argument's Maximum.

**min:** All input argument's Minimum.

**sum:** All argument values Sum.

**mean:** (Trimmed) arithmetic mean.

**sd:** Calculate the standard deviation of a column of continuous real valued data.

**var:** Compute the variance of x.

## Generic Summaries

### NON-GROUP\_BY SUMMARIES

**h2o.median:** Calculate the median of x.

**h2o.range:** Input argument's Min/Max Vector

**h2o.cor:** Correlation Matrix of H2O Frames.

**h2o.quantile:** Obtain and display quantiles for an H2O Frame Column.

 **h2o.hist:** Compute a histogram over a numeric H2O Frame Column.

**h2o.prod:** Product of all arguments values.

**h2o.any:** Given a set of logical vectors, determine if at least one of the values is true.

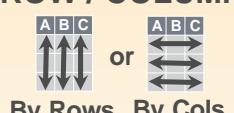
**h2o.all:** Given a set of logical vectors, determine if all of the values are true.

### NON-GROUP\_BY SUMMARIES: GENERIC

**h2o.summary:** Produce result summaries of the results of various model fitting functions.

## Aggregations

### ROW / COLUMN AGGREGATION



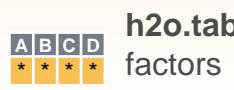
**apply:** Apply a function over an H2O parsed data object (an array) margins.

### GROUP BY AGGREGATION



**h2o.group\_by:** Apply an aggregate function to each group of an H2O dataset.

### TABULATION



**h2o.table:** Use the cross-classifying factors to build a table of counts at each combination of factor levels.

# h2o::CHEAT SHEET



## Data Modeling

### MODEL TRAINING: SUPERVISED LEARNING

**h2o.deeplearning:** Perform Deep Learning Neural Networks on an H2OFrame.

**h2o.gbm:** Build Gradient Boosted Regression Trees or Classification Trees.

**h2o.glm:** Fit a Generalized Linear Model, specified by a response variable, a set of predictors, and the error distribution.

**h2o.naiveBayes:** Compute Naive Bayes classification probabilities on an H2O Frame.

**h2o.randomForest:** Perform Random Forest Classification on an H2O Frame.

**h2o.xgboost:** Build an Extreme Gradient Boosted Model using the XGBoost backend.

**h2o.stackedEnsemble:** Build a stacked ensemble (aka. Super Learner) using the specified H2O base learning algorithms.

**h2o.automl:** Automates the Supervised Machine Learning Model Training Process: Automatically Trains and Cross-validates a set of Models, and trains a Stacked Ensemble.

### MODEL TRAINING: UNSUPERVISED LEARNING

**h2o.prcomp:** Perform Principal Components Analysis on the given H2O Frame.

**h2o.kmeans:** Perform k-means Clustering on the given H2O Frame.

**h2o.anomaly:** Detect anomalies in a H2O Frame using a H2O Deep Learning Model with Auto-Encoding.

**h2o.deepfeatures:** Extract the non-linear features from a H2O Frame using a H2O Deep Learning Model.

**h2o.glrn:** Builds a Generalized Low Rank Decomposition of an H2O Frame.

**h2o.svd:** Singular value decomposition of an H2O Frame using the power method.

**h2o.word2vec:** Trains a word2vec model on a String column of an H2O data frame.

### SURVIVAL MODELS: TIME-TO-EVENT

**h2o.coxph:** Trains a Cox Proportional Hazards Model (CoxPH) on an H2O Frame.

### GRID SEARCH

**h2o.grid:** Efficient method to build multiple models with different hyperparameters.

**h2o.getGrid:** Get a grid object from H2O distributed K/V store.

### MODEL SCORING

**h2o.predict:** Obtain predictions from various fitted H2O model objects.

**h2o.scoreHistory:** Get Model Score History.

### MODEL METRICS

**h2o.make\_metrics:** Given predicted values (target for regression, class-1 probabilities, or binomial or per-class probabilities for multinomial), compute a model metrics object.

### GENERAL MODEL HELPER

**h2o.performance:** Evaluate the predictive performance of a Supervised Learning Regression or Classification Model via various metrics. Set **xval = TRUE** for retrieving the training cross-validation metrics.

### REGRESSION MODEL HELPER

**h2o.mse:** Display the mean squared error calculated from "Predicted Responses" and "Actual (Reference) Responses". Set **xval = TRUE** for retrieving the cross-validation MSE.

### CLASSIFICATION MODEL HELPERS

**h2o.accuracy:** Get Model Accuracy metric.

**h2o.auc:** Retrieve the AUC (area under ROC curve). Set **xval = TRUE** for retrieving the cross-validation AUC.

**h2o.confusionMatrix:** Display prediction errors for classification data ("Predicted" vs "Reference : Real Values").

**h2o.hit\_ratio\_table:** Retrieve the Hit Ratios. Set **xval = TRUE** for retrieving the cross-validation Hit Ratio.

### CLUSTERING MODEL HELPER

**h2o.betweenss:** Get the between cluster Sum of Squares.

**h2o.centers:** Retrieve the Model Centers.

### PREDICTOR VARIABLE IMPORTANCE

**h2o.varimp:** Retrieve the variable importance

**h2o.varimp\_plot:** Plot Variable Importances.

## Data Munging

### GENERAL COLUMN MANIPULATION

**is.na:** Display missing elements.

### FACTOR LEVEL MANIPULATIONS

**h2o.levels:** Display a list of the unique values found in a categorical data column.

**h2o.relevel:** Reorders levels of an H2O factor, similarly to standard R's `relevel`.

**h2o.setLevels:** Set Levels of H2O Factor.

### NUMERIC COLUMN MANIPULATIONS

**h2o.cut:** Convert H2O Numeric Data to Factor by breaking it into Intervals.

### CHARACTER COLUMN MANIPULATIONS

**h2o.strsplit:** "String Split": Splits the given factor column on the input split.

**h2o.tolower:** Convert the characters of a character vector to lower case.

**h2o.toupper:** Convert the characters of a character vector to upper case.

**h2o.trim:** "Trim spaces": Remove leading and trailing white space.

**h2o.gsub:** Match a pattern & replace **all** instances (occurrences) of the matched pattern with the replacement string globally.

**h2o.sub:** Match a pattern & replace the **first** instance (occurrence) of the matched pattern with the replacement string.

### DATE MANIPULATIONS

**h2o.month:** Convert Milliseconds to Months in H2O Datasets (Scale: 0 to 11).

**h2o.year:** Convert Milliseconds to Years in H2O Datasets, indexed starting from 1900.

**h2o.day:** Convert Milliseconds to Day of Month in H2O Datasets (Scale: 1 to 31).

**h2o.hour:** Convert Milliseconds to Hour of Day in H2O Datasets (Scale: 0 to 23).

**h2o.dayOfWeek:** Convert Milliseconds to Day of Week in a H2OFrame (Scale: 0 to 6)

### MATRIX OPERATIONS

**%\*%:** Multiply two conformable matrices.

**t:** Returns the transpose of an H2OFrame.

## Cluster Operations

### H2O KEY VALUE STORE ACCESS

**h2o.assign:** Assign H2O hex.keys to R objects.

**h2o.getFrame:** Get H2O dataset Reference.

**h2o.getModel:** Get H2O model reference.

**h2o.ls:** Display a list of object keys in the running instance of H2O.

**h2o.rm:** Remove specified H2O Objects from the H2O server, but not from the R environment.

**h2o.removeAll:** Remove All H2O Objects from the H2O server, but not from the R environment.

### H2O MODEL IMPORT / EXPORT

**h2o.loadModel:** Load H2OModel from disk.

**h2o.saveModel:** Save H2OModel object to disk.

**h2o.download\_pojo:** Download the Scoring POJO (Plain Old Java Object) of an H2O Model.

**h2o.download\_mojo:** Download the model in MOJO format.

### H2O CLUSTER CONNECTION

**h2o.init:** Connect to a running H2O instance using all CPUs on the host.

**h2o.shutdown:** Shut down the specified H2O instance. All data on the server will be lost!

### H2O CLUSTER INFORMATION

**h2o.clusterInfo:** Display the name, version, uptime, total nodes, total memory, total cores and health of a cluster running H2O.

**h2o.clusterStatus:** Retrieve information on the status of the cluster running H2O.

### H2O LOGGING

**h2o.clearLog:** Clear all H2O R command and error response logs from the local disk.

**h2o.downloadAllLogs:** Download all H2O log files to the local disk.

**h2o.logAndEcho:** Write a message to the H2O Java log file and echo it back.

**h2o.openLog:** Open existing logs of H2O R POST commands and error responses on disk.

**h2o.getLogPath:** Get the file path for the H2O R command and error response logs.

**h2o.startLogging:** Begin logging H2O R POST commands and error responses.

**h2o.stopLogging:** Stop logging H2O R POST commands and error responses.



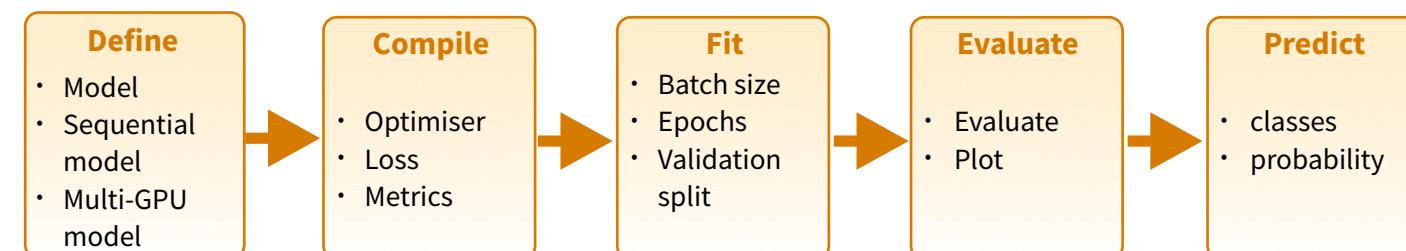
# Deep Learning with Keras :: CHEAT SHEET



## Intro

[Keras](#) is a high-level neural networks API developed with a focus on enabling fast experimentation. It supports multiple backends, including TensorFlow, CNTK and Theano.

TensorFlow is a lower level mathematical library for building deep neural network architectures. The [keras](#) R package makes it easy to use Keras and TensorFlow in R.



<https://keras.rstudio.com>

<https://www.manning.com/books/deep-learning-with-r>

The “Hello, World!”  
of deep learning

## Working with keras models

### DEFINE A MODEL

`keras_model()` Keras Model

`keras_model_sequential()` Keras Model composed of a linear stack of layers

`multi_gpu_model()` Replicates a model on different GPUs

### COMPILE A MODEL

`compile(object, optimizer, loss, metrics = NULL)`

Configure a Keras model for training

### FIT A MODEL

`fit(object, x = NULL, y = NULL, batch_size = NULL, epochs = 10, verbose = 1, callbacks = NULL, ...)`  
Train a Keras model for a fixed number of epochs (iterations)

`fit_generator()` Fits the model on data yielded batch-by-batch by a generator

`train_on_batch(); test_on_batch()` Single gradient update or model evaluation over one batch of samples

### EVALUATE A MODEL

`evaluate(object, x = NULL, y = NULL, batch_size = NULL)` Evaluate a Keras model

`evaluate_generator()` Evaluates the model on a data generator

### PREDICT

`predict()` Generate predictions from a Keras model

`predict_proba() and predict_classes()`

Generates probability or class probability predictions for the input samples

`predict_on_batch()` Returns predictions for a single batch of samples

`predict_generator()` Generates predictions for the input samples from a data generator

### OTHER MODEL OPERATIONS

`summary()` Print a summary of a Keras model

`export_savedmodel()` Export a saved model

`get_layer()` Retrieves a layer based on either its name (unique) or index

`pop_layer()` Remove the last layer in a model

`save_model_hdf5(); load_model_hdf5()` Save/Load models using HDF5 files

`serialize_model(); unserialize_model()`

Serialize a model to an R object

`clone_model()` Clone a model instance

`freeze_weights(); unfreeze_weights()`

Freeze and unfreeze weights

### CORE LAYERS



`layer_input()` Input layer



`layer_dense()` Add a densely-connected NN layer to an output



`layer_activation()` Apply an activation function to an output



`layer_dropout()` Applies Dropout to the input



`layer_reshape()` Reshapes an output to a certain shape



`layer_permute()` Permute the dimensions of an input according to a given pattern



`layer_repeat_vector()` Repeats the input n times



`layer_lambda(object, f)` Wraps arbitrary expression as a layer



`layer_activity_regularization()` Layer that applies an update to the cost function based on input activity



`layer_masking()` Masks a sequence by using a mask value to skip timesteps



`layer_flatten()` Flattens an input

## INSTALLATION

The [keras](#) R package uses the Python [keras](#) library. You can install all the prerequisites directly from R.

[https://keras.rstudio.com/reference/install\\_keras.html](https://keras.rstudio.com/reference/install_keras.html)

```
library(keras)  
install_keras()
```

See `?install_keras`  
for GPU instructions

This installs the required libraries in an Anaconda environment or virtual environment '`r-tensorflow`'.

## TRAINING AN IMAGE RECOGNIZER ON MNIST DATA

# input layer: use MNIST images



`mnist <- dataset_mnist()`

`x_train <- mnist$train$x; y_train <- mnist$train$y`

`x_test <- mnist$test$x; y_test <- mnist$test$y`

# reshape and rescale

`x_train <- array_reshape(x_train, c(nrow(x_train), 784))`

`x_test <- array_reshape(x_test, c(nrow(x_test), 784))`

`x_train <- x_train / 255; x_test <- x_test / 255`

`y_train <- to_categorical(y_train, 10)`

`y_test <- to_categorical(y_test, 10)`

# defining the model and layers

`model <- keras_model_sequential()`

`model %>%`

`layer_dense(units = 256, activation = 'relu',  
 input_shape = c(784)) %>%`

`layer_dropout(rate = 0.4) %>%`

`layer_dense(units = 128, activation = 'relu') %>%`

`layer_dense(units = 10, activation = 'softmax')`

# compile (define loss and optimizer)

`model %>% compile(`

`loss = 'categorical_crossentropy',`

`optimizer = optimizer_rmsprop(),`

`metrics = c('accuracy')`

)

# train (fit)

`model %>% fit(`

`x_train, y_train,`

`epochs = 30, batch_size = 128,`

`validation_split = 0.2`

)

`model %>% evaluate(x_test, y_test)`

`model %>% predict_classes(x_test)`

# More layers

## CONVOLUTIONAL LAYERS

	<code>layer_conv_1d()</code> 1D, e.g. temporal convolution
	<code>layer_conv_2d_transpose()</code> Transposed 2D (deconvolution)
	<code>layer_conv_2d()</code> 2D, e.g. spatial convolution over images
	<code>layer_conv_3d_transpose()</code> Transposed 3D (deconvolution) <code>layer_conv_3d()</code> 3D, e.g. spatial convolution over volumes
	<code>layer_conv_lstm_2d()</code> Convolutional LSTM
	<code>layer_separable_conv_2d()</code> Depthwise separable 2D
	<code>layer_upsampling_1d()</code> <code>layer_upsampling_2d()</code> <code>layer_upsampling_3d()</code> Upsampling layer
	<code>layer_zero_padding_1d()</code> <code>layer_zero_padding_2d()</code> <code>layer_zero_padding_3d()</code> Zero-padding layer
	<code>layer_cropping_1d()</code> <code>layer_cropping_2d()</code> <code>layer_cropping_3d()</code> Cropping layer

## POOLING LAYERS

	<code>layer_max_pooling_1d()</code> <code>layer_max_pooling_2d()</code> <code>layer_max_pooling_3d()</code> Maximum pooling for 1D to 3D
	<code>layer_average_pooling_1d()</code> <code>layer_average_pooling_2d()</code> <code>layer_average_pooling_3d()</code> Average pooling for 1D to 3D
	<code>layer_global_max_pooling_1d()</code> <code>layer_global_max_pooling_2d()</code> <code>layer_global_max_pooling_3d()</code> Global maximum pooling
	<code>layer_global_average_pooling_1d()</code> <code>layer_global_average_pooling_2d()</code> <code>layer_global_average_pooling_3d()</code> Global average pooling

## ACTIVATION LAYERS

	<code>layer_activation()</code> object, activation Apply an activation function to an output
	<code>layer_activation_leaky_relu()</code> Leaky version of a rectified linear unit
	<code>layer_activation_parametric_relu()</code> Parametric rectified linear unit
	<code>layer_activation_thresholded_relu()</code> Thresholded rectified linear unit
	<code>layer_activation_elu()</code> Exponential linear unit

## DROPOUT LAYERS

	<code>layer_dropout()</code> Applies dropout to the input
	<code>layer_spatial_dropout_1d()</code> <code>layer_spatial_dropout_2d()</code> <code>layer_spatial_dropout_3d()</code> Spatial 1D to 3D version of dropout

## RECURRENT LAYERS

	<code>layer_simple_rnn()</code> Fully-connected RNN where the output is to be fed back to input
	<code>layer_gru()</code> Gated recurrent unit - Cho et al
	<code>layer_cudnn_gru()</code> Fast GRU implementation backed by CuDNN

## LOCALLY CONNECTED LAYERS

	<code>layer_locally_connected_1d()</code> <code>layer_locally_connected_2d()</code> Similar to convolution, but weights are not shared, i.e. different filters for each patch
---	--

# Preprocessing

## SEQUENCE PREPROCESSING

	<code>pad_sequences()</code> Pads each sequence to the same length (length of the longest sequence)
	<code>skipgrams()</code> Generates skipgram word pairs
	<code>make_sampling_table()</code> Generates word rank-based probabilistic sampling table

## TEXT PREPROCESSING

	<code>text_tokenizer()</code> Text tokenization utility
	<code>fit_text_tokenizer()</code> Update tokenizer internal vocabulary
	<code>save_text_tokenizer(); load_text_tokenizer()</code> Save a text tokenizer to an external file
	<code>texts_to_sequences(); texts_to_sequences_generator()</code> Transforms each text in texts to sequence of integers
	<code>texts_to_matrix(); sequences_to_matrix()</code> Convert a list of sequences into a matrix
	<code>text_one_hot()</code> One-hot encode text to word indices
	<code>text_hashing_trick()</code> Converts a text to a sequence of indexes in a fixed-size hashing space
	<code>text_to_word_sequence()</code> Convert text to a sequence of words (or tokens)

## IMAGE PREPROCESSING

	<code>image_load()</code> Loads an image into PIL format.
	<code>flow_images_from_data()</code> <code>flow_images_from_directory()</code> Generates batches of augmented/normalized data from images and labels, or a directory
	<code>image_data_generator()</code> Generate minibatches of image data with real-time data augmentation.
	<code>fit_image_data_generator()</code> Fit image data generator internal statistics to some sample data
	<code>generator_next()</code> Retrieve the next item
	<code>image_to_array(); image_array_resize(); image_array_save()</code> 3D array representation

# Pre-trained models

Keras applications are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning.
<code>application_xception()</code> <code>xception_preprocess_input()</code> Xception v1 model

<code>application_inception_v3()</code> <code>inception_v3_preprocess_input()</code> Inception v3 model, with weights pre-trained on ImageNet
---

<code>application_inception_resnet_v2()</code> <code>inception_resnet_v2_preprocess_input()</code> Inception-ResNet v2 model, with weights trained on ImageNet
--

<code>application_vgg16(); application_vgg19()</code> VGG16 and VGG19 models
---

<code>application_resnet50()</code> ResNet50 model
--

<code>application_mobilenet()</code> <code>mobilenet_preprocess_input()</code> <code>mobilenet_decode_predictions()</code> <code>mobilenet_load_model_hdf5()</code> MobileNet model architecture
--

## IMAGENET

[ImageNet](#) is a large database of images with labels, extensively used for deep learning

<code>imagenet_preprocess_input()</code> <code>imagenet_decode_predictions()</code> Preprocesses a tensor encoding a batch of images for ImageNet, and decodes predictions
--

## Callbacks

A callback is a set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during training.

<code>callback_early_stopping()</code> Stop training when a monitored quantity has stopped improving
<code>callback_learning_rate_scheduler()</code> Learning rate scheduler
<code>callback_tensorboard()</code> TensorBoard basic visualizations

# Machine Learning Modelling in R :: CHEAT SHEET

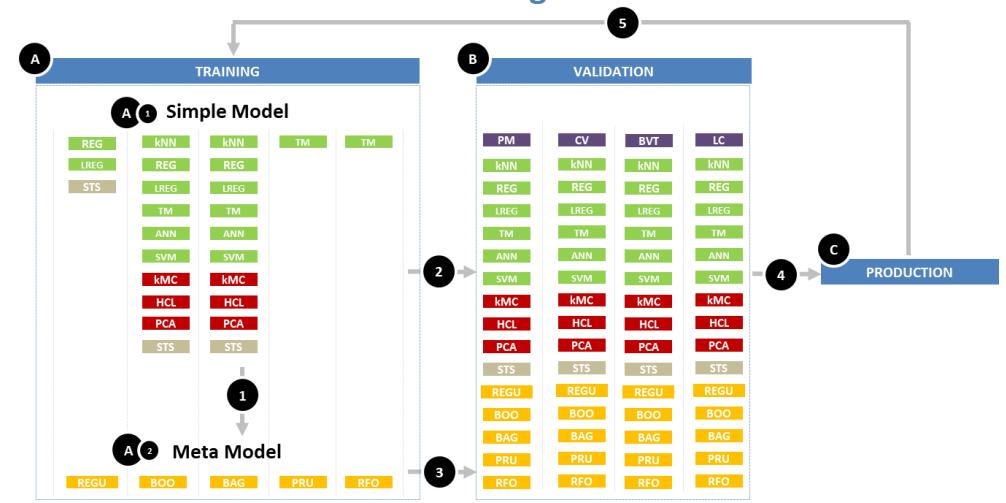
## Supervised & Unsupervised Learning

ALGORITHM	DESCRIPTION	R PACKAGE::FUNCTION	SAMPLE CODE
NBC Naïve Bayes classifier	A classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naïve Bayes classifier assumes that the presence or absence of a particular feature in a class is unrelated to the presence of any other feature	e1071::naiveBayes	naiveBayes(class ~ ., data = x)
kNN k-Nearest Neighbours	A non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression	class::knn	knn(train, test, cl, k = 1, l = 0, prob = FALSE, use.all = TRUE)
LRG Linear Regression	Model the linear relationship between a scalar dependent variable Y and one or more explanatory variables (or independent variables) denoted X	stats::lm	lm(dist ~ speed, data=cars)
LRC Logistic Regression	Used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables.	stats::glm	glm(Y ~ ., family = binomial (link = 'logit'), data = X)
TM Tree-Based Models	The idea is to consecutively divide (branch) the training data into smaller and smaller features until an assignment criterion with respect to the target variable into a "data bucket" (leaf) is reached	rpart::rpart	rpart(Kyphosis ~ Age + Number + Start, data = kyphosis)
ANN Artificial Neural Network	Neural networks are built from units called perceptrons. Perceptrons have one or more inputs, an activation function and an output. An ANN model is built up by combining perceptrons in structured layers.	neuralnet::neuralnet	neuralnet(f,data=train_hidden=(5,3),linear.output=T)
SVM Support Vector Machine	A data classification method that separates data using hyperplanes	e1071::svm	svm(formula, data = NULL, ..., subset, na.action = na.omit, scale = TRUE)
PCA Principal Component Analysis	A procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.	stats::prcomp stats::princomp FactoMineR::PCA ade4::dudi.pca amap::acp	stats::prcomp(formula, data = NULL, subset, na.action, ...) stats::princomp(formula, data = NULL, subset, na.action, ...) FactoMineR::PCA(decatlon, quanti.sup = 11:12, quali.sup = 13) ade4::dudi.pca(deugStab, center = deugCent, scale = FALSE, scan = FALSE) amap::acp(lubisch)
HAC k-Mean Clustering	Aims at partitioning n observations into k clusters in which each observation belongs to the cluster with the nearest mean	stats::kmeans	kmeans(k, centers, iter.max = 10, nstart = 1, algorithm = c("Hartigan-Wong", "Lloyd", "Forgy", "MacQueen"), trace = FALSE)
HCL Hierarchical Clustering	An approach which builds a hierarchy from the bottom-up, and doesn't require the number of clusters to be specified beforehand.	stats::hclust	hclust(d, method = "complete", members = NULL)

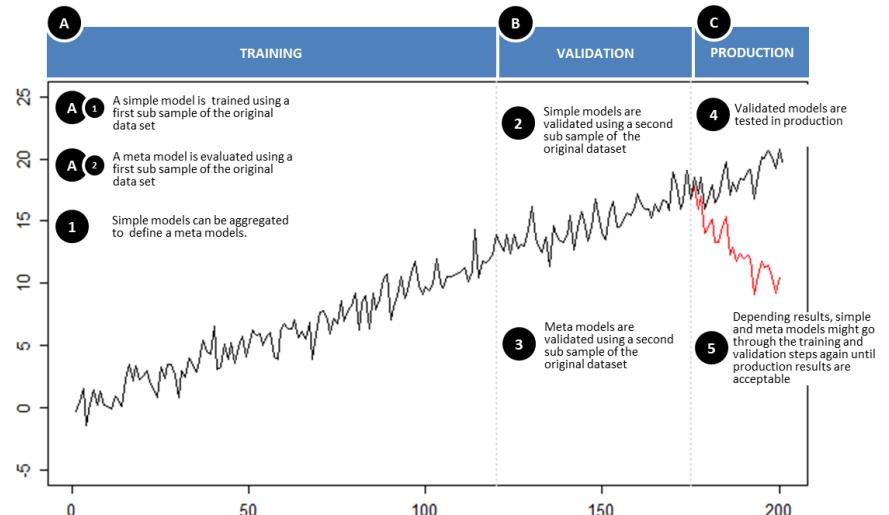
## Meta-Algorithm, Time Series & Model Validation

ALGORITHM	DESCRIPTION	R PACKAGE::FUNCTION	SAMPLE CODE
REGU Regularisation L1 (Lasso) L2 (Ridge)	Regularisation adds a penalty on the different parameters of a model to reduce the freedom of the model. Hence, the model will be less likely to fit the noise of the training data and will improve the generalization abilities of the model	glmnet::glmnet	L1 : glmnet(myMatrixA, myMatrixB, family = "gaussian", alpha = 1) L2 : glmnet(myMatrixA, myMatrixB, family = "gaussian", alpha = 0)
BOO Boosting	A process of iteratively refining, e.g. by reweighting, of estimated regression and classification functions (though it has primarily been applied to the latter), in order to improve predictive ability.	gbm::gbm	gbmboost(Y ~ ., data = curr1[trnidxs,])
BAG Bagging	Bagging is a way to increase the power of a predictive statistical model by taking multiple random samples (with replacement) of the training data set, and using each of them to construct a separate model and separate predictions for the original test set	randomForest::randomForest	foreach : d <- data.frame(x=1:10, y=rnorm(10)) s <- foreach(d=i %in% d, by=row, .combine=rbind, .id=i, .d = i) ipred : bagging(formula, data, subset, na.action=na.rpart, \dots)
PRU Pruning	Pruning is a technique that reduces the size of decision tree by removing sections of the tree that provide little power to classify instances. Pruning reduces the complexity of the final classifier and hence improves predictive accuracy by reducing overfitting	rpart::rpart	prune(x, cp = 0.1)
RFO Random Forest	An ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression)	randomForest::randomForest	randomForest(X ~ ., data = Y, subset = mySub)
STS Time Series	Random sampling of observations for training and testing a model can be an issue when faced with a times dimension. Random sampling may either destroy serial correlation properties in the data which we would like to exploit	stats::xts forecast::spectral spectral::TTR	Auto-correlation: acf(x, lag.max = NULL, type = c("correlation", "covariance", "partial")) Spectral Analysis: spec.pgram(..., spans = NULL) Seasonal Decomposition of Time Series : stl(x, s.window = 7, t.window = 50, t.jump = 1) ....
PM Performance metrics	Depends on the problem: • Regression: squared errors, outliers, error rate... • Classification: Accuracy, precision, recall, F-score...	ROCR::ROC	Regression:stats::outlierTest, stats::qqPlot ... Classification:ROCR::ROC
JVT Bias-Variance Tradeoff	• Simple models with few parameters are easier to compute but may lead to poorer fits ( <b>high bias</b> ). • Complex models may provide more accurate fits but may over-fit the data ( <b>high variance</b> )	caret::confusionMatrix	Tailored to the analysis
CV Cross validation	Cross validation compares the test performances of different model realisations with different sets or values of parameters	caret::createDataPartition caret::createFolds	Tailored to the analysis
LC Learning Curves	Learning curves plot a model's training and test errors, or the chosen performance metric, depending on the training set size	caret::learning_curve_dat	createDataPartition(classes, p = 0.8, list = FALSE) learning_curve_dat(dat, outcome = NULL, proportion = (1:10)/10, test_prop = 0, verbose = TRUE, ...)

## Standard Modelling Workflow



## Time Series View



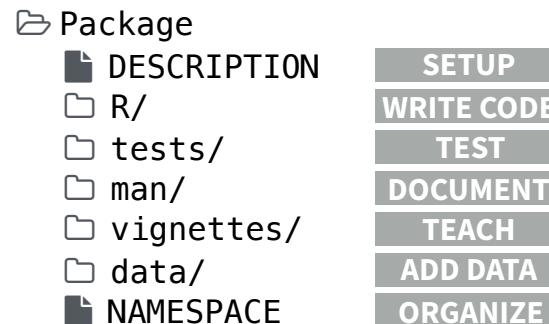
# Package Development: : CHEAT SHEET



## Package Structure

A package is a convention for organizing files into directories.

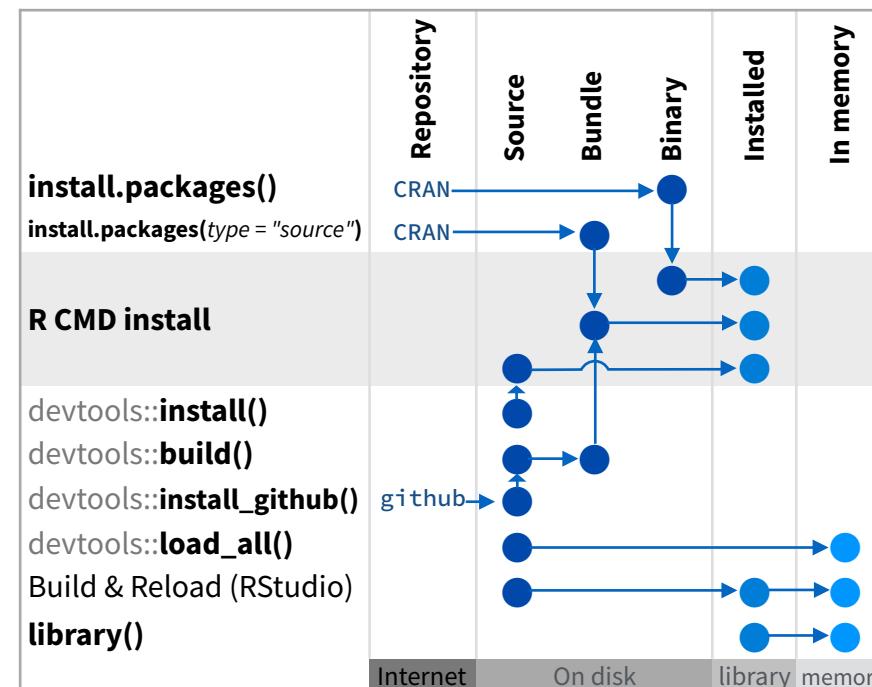
This sheet shows how to work with the 7 most common parts of an R package:



The contents of a package can be stored on disk as a:

- source** - a directory with sub-directories (as above)
- bundle** - a single compressed file (*.tar.gz*)
- binary** - a single compressed file optimized for a specific OS

Or installed into an R library (loaded into memory during an R session) or archived online in a repository. Use the functions below to move between these states.



**devtools::use\_build\_ignore("file")**

Adds file to *.Rbuildignore*, a list of files that will not be included when package is built.

## Setup (DESCRIPTION)

The **DESCRIPTION** file describes your work, sets up how your package will work with other packages, and applies a copyright.

- You must have a **DESCRIPTION** file
- Add the packages that yours relies on with **devtools::use\_package()**  
Adds a package to the Imports or Suggests field

CC0	MIT	GPL-2
No strings attached.	MIT license applies to your code if re-shared.	GPL-2 license applies to your code, <i>and all code anyone bundles with it</i> , if re-shared.

**Package:** mypackage  
**Title:** Title of Package  
**Version:** 0.1.0  
**Authors@R:** person("Hadley", "Wickham", email = "hadley@me.com", role = c("aut", "cre"))  
**Description:** What the package does (one paragraph)  
**Depends:** R (>= 3.1.0)  
**License:** GPL-2  
**LazyData:** true  
**Imports:**

dplyr (>= 0.4.0),  
ggvis (>= 0.2)  
Suggests:  
knitr (>= 0.1.0)

**Import** packages that your package *must* have to work. R will install them when it installs your package.

**Suggest** packages that are not very essential to yours. Users can install them manually, or not, as they like.

## Write Code (R/)

All of the R code in your package goes in **R/**. A package with just an R/ directory is still a very useful package.

- Create a new package project with **devtools::create("path/to/name")**  
Create a template to develop into a package.
- Save your code in **R/** as scripts (extension .R)

### WORKFLOW

1. Edit your code.
2. Load your code with one of  
**devtools::load\_all()**  
Re-loads all saved files in **R/** into memory.  
**Ctrl/Cmd + Shift + L** (keyboard shortcut)  
Saves all open files then calls **load\_all()**.
3. Experiment in the console.
4. Repeat.
  - Use consistent style with **r-pkgs.had.co.nz/r.html#style**
  - Click on a function and press **F2** to open its definition
  - Search for a function with **Ctrl +**.



Visit [r-pkgs.had.co.nz](http://r-pkgs.had.co.nz) to learn much more about writing and publishing packages for R

## Test (tests/)

Use **tests/** to store tests that will alert you if your code breaks.

- Add a **tests/** directory
- Import **testthat** with **devtools::use\_testthat()**, which sets up package to use automated tests with testthat
- Write tests with **context()**, **test()**, and expect statements
- Save your tests as .R files in **tests/testthat/**

### WORKFLOW

1. Modify your code or tests.
2. Test your code with one of  
**devtools::test()**  
Runs all tests in **tests/**  
**Ctrl/Cmd + Shift + T** (keyboard shortcut)
3. Repeat until all tests pass

**Example Test**

```
context("Arithmetic")
test_that("Math works", {
  expect_equal(1 + 1, 2)
  expect_equal(1 + 2, 3)
  expect_equal(1 + 3, 4)
})
```

Expect statement	Tests
<code>expect_equal()</code>	is equal within small numerical tolerance?
<code>expect_identical()</code>	is exactly equal?
<code>expect_match()</code>	matches specified string or regular
<code>expect_output()</code>	prints specified output?
<code>expect_message()</code>	displays specified message?
<code>expect_warning()</code>	displays specified warning?
<code>expect_error()</code>	throws specified error?
<code>expect_is()</code>	output inherits from certain class?
<code>expect_false()</code>	returns FALSE?
<code>expect_true()</code>	returns TRUE?



## Document (📄 man/)

📄 man/ contains the documentation for your functions, the help pages in your package.

- Use roxygen comments to document each function beside its definition
- Document the name of each exported data set
- Include helpful examples for each function

### WORKFLOW

1. Add roxygen comments in your .R files
2. Convert roxygen comments into documentation with one of:

`devtools::document()`

Converts roxygen comments to .Rd files and places them in 📄 man/. Builds NAMESPACE.

**Ctrl/Cmd + Shift + D** (Keyboard Shortcut)

3. Open help pages with ? to preview documentation
4. Repeat

### .Rd FORMATTING TAGS

\emph{italic text}	\email{name@@foo.com}
\strong{bold text}	\href{url}{display}
\code{function(args)}	\url{url}
\pkg{package}	
\dontrun{code}	\link[=dest]{display}
\dontshow{code}	\linkS4class{class}
\donttest{code}	\code{\link{function}}
\deqn{a + b (block)}	\code{\link[package]{function}}
\eqn{a + b (inline)}	\tabular{lcr}{ left \tab centered \tab right \cr cell \tab cell \tab cell \cr}

### ROXYGEN2

The **roxygen2** package lets you write documentation inline in your .R files with a shorthand syntax. devtools implements roxygen2 to make documentation.



- Add roxygen documentation as comment lines that begin with #’.
- Place comment lines directly above the code that defines the object documented.
- Place a roxygen @ tag (right) after #’ to supply a specific section of documentation.
- Untagged lines will be used to generate a title, description, and details section (in that order)

```
#' Add together two numbers.
#'
#' @param x A number.
#' @param y A number.
#' @return The sum of \code{x} and \code{y}.
#' @examples
#' add(1, 1)
#' @export
add <- function(x, y) {
  x + y
}
```

### COMMON ROXYGEN TAGS

@aliases	@inheritParams	<b>@seealso</b>	
@concepts	@keywords	@format	
@describeln	<b>@param</b>	@source	data
<b>@examples</b>	@rdname	@include	
<b>@export</b>	<b>@return</b>	@slot	S4
@family	@section	@field	RC

## Teach (📄 vignettes/)

📄 vignettes/ holds documents that teach your users how to solve real problems with your tools.

- Create a 📄 vignettes/ directory and a template vignette with `devtools::use_vignette()`  
Adds template vignette as vignettes/my-vignette.Rmd.
- Append YAML headers to your vignettes (like right)
- Write the body of your vignettes in R Markdown ([rmarkdown.rstudio.com](http://rmarkdown.rstudio.com))

```
---
```

```
title: "Vignette Title"
author: "Vignette Author"
date: "`r Sys.Date()`"
output: rmarkdown::html_vignette
vignette: >
  \%VignetteIndexEntry{Vignette Title}
  \%VignetteEngine{knitr::rmarkdown}
  \usepackage[utf8]{inputenc}
```

```
---
```

## Add Data (📄 data/)

The 📄 data/ directory allows you to include data with your package.

- Save data as .Rdata files (suggested)
- Store data in one of **data/**, **R/Sysdata.rda**, **inst/extdata**
- Always use **LazyData: true** in your DESCRIPTION file.

`devtools::use_data()`

Adds a data object to data/ (R/Sysdata.rda if **internal = TRUE**)

`devtools::use_data_raw()`

Adds an R Script used to clean a data set to data-raw/. Includes data-raw/ on .Rbuildignore.

### Store data in

- **data/** to make data available to package users
- **R/sysdata.rda** to keep data internal for use by your functions.
- **inst/extdata** to make raw data available for loading and parsing examples. Access this data with **system.file()**

## Organize (📄 NAMESPACE)

The 📄 NAMESPACE file helps you make your package self-contained: it won’t interfere with other packages, and other packages won’t interfere with it.

- Export functions for users by placing **@export** in their roxygen comments
- Import objects from other packages with **package::object** (recommended) or **@import**, **@importFrom**, **@importClassesFrom**, **@importMethodsFrom** (not always recommended)

### WORKFLOW

1. Modify your code or tests.
2. Document your package (`devtools::document()`)
3. Check NAMESPACE
4. Repeat until NAMESPACE is correct

### SUBMIT YOUR PACKAGE

[r-pkgs.had.co.nz/release.html](http://r-pkgs.had.co.nz/release.html)

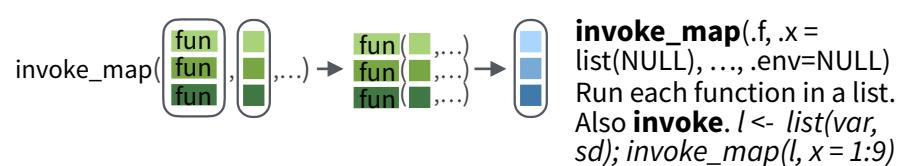
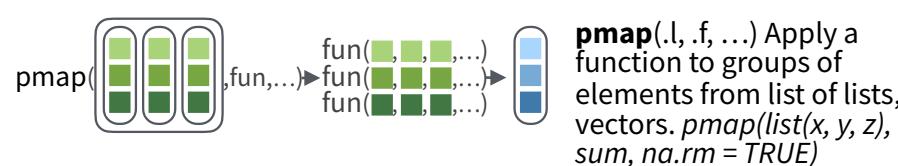
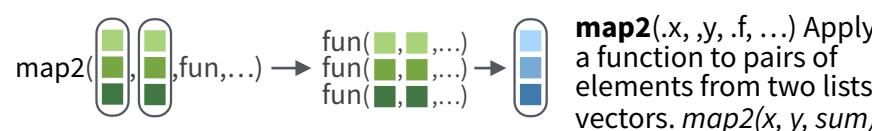
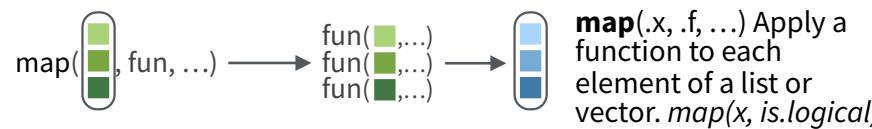


# Apply functions with purrr :: CHEAT SHEET



## Apply Functions

Map functions apply a function iteratively to each element of a list or vector.



**lmap(x, f, ...)** Apply function to each list-element of a list or vector.  
**imap(x, f, ...)** Apply .f to each element of a list or vector and its index.

### OUTPUT

**map()**, **map2()**, **pmap()**, **imap** and **invoke\_map** each return a list. Use a suffixed version to return the results as a specific type of flat vector, e.g. **map2\_chr**, **pmap\_lgl**, etc.

Use **walk**, **walk2**, and **pwalk** to trigger side effects. Each return its input invisibly.

### SHORTCUTS - within a purrr function:

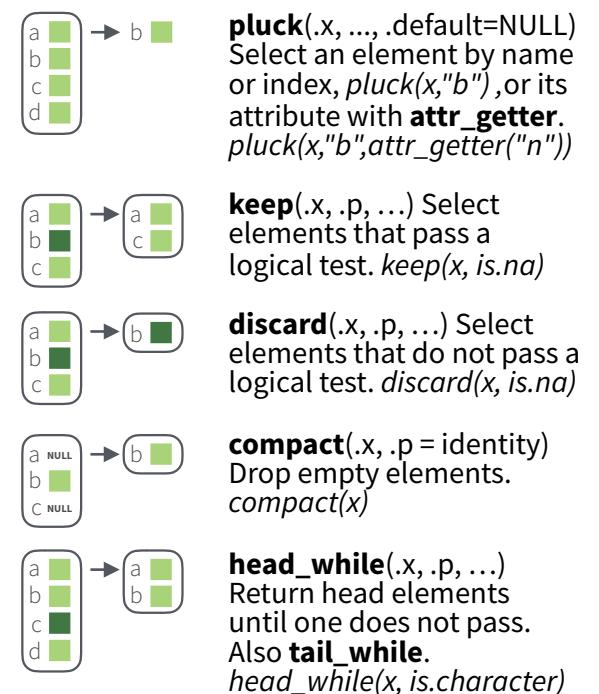
"**name**" becomes **function(x)x[["name"]]**, e.g. **map(l, "a")** extracts *a* from each element of *l*

**~.x** becomes **function(x)x**, e.g. **map(l, ~2+x)** becomes **map(l, function(x) 2+x)**

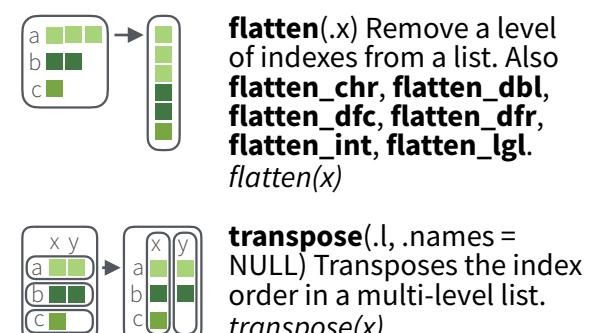
function	returns
<b>map</b>	list
<b>map_chr</b>	character vector
<b>map_dbl</b>	double (numeric) vector
<b>map_dfc</b>	data frame (column bind)
<b>map_dfr</b>	data frame (row bind)
<b>map_int</b>	integer vector
<b>map_lgl</b>	logical vector
<b>walk</b>	triggers side effects, returns the input invisibly

## Work with Lists

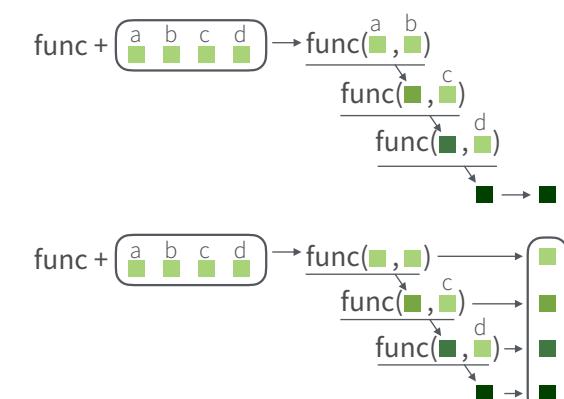
### FILTER LISTS



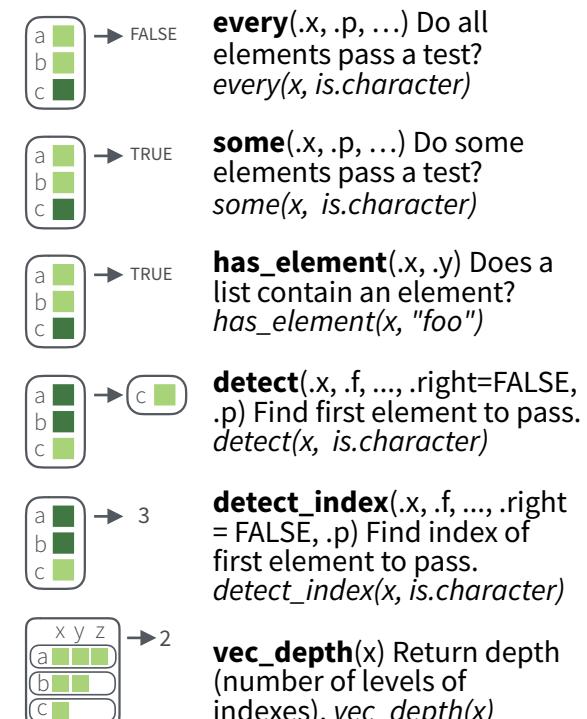
### RESHAPE LISTS



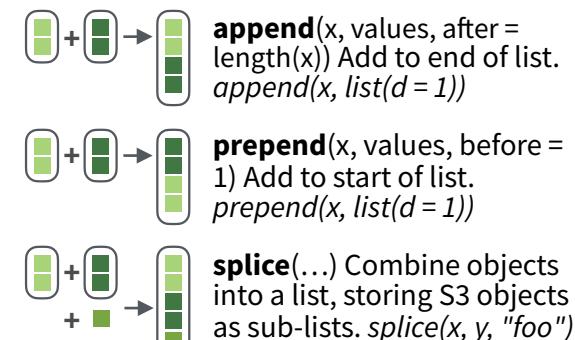
## Reduce Lists



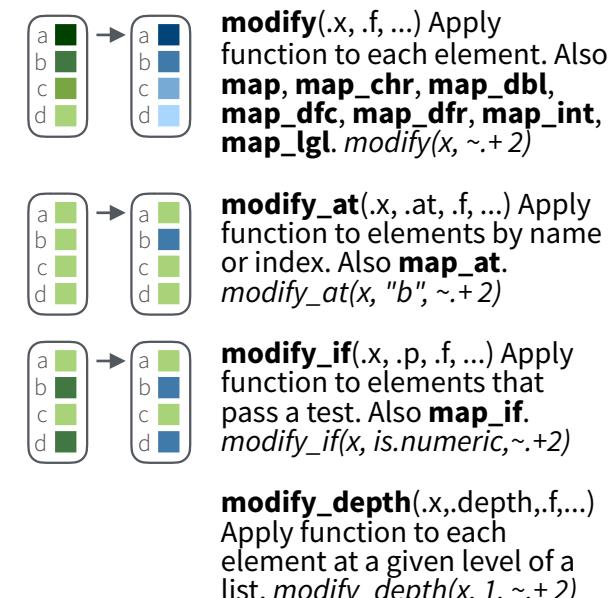
### SUMMARISE LISTS



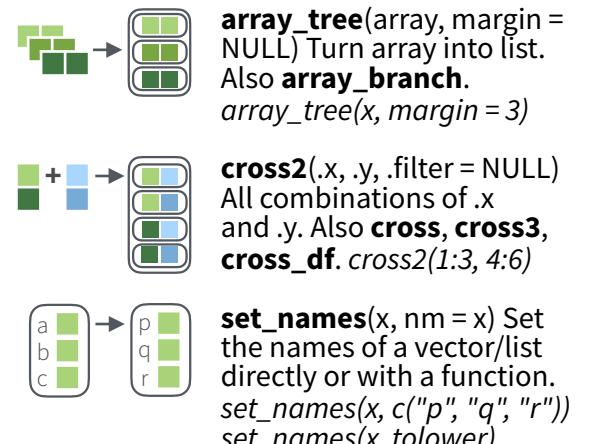
### JOIN (TO) LISTS



### TRANSFORM LISTS



### WORK WITH LISTS



## Modify function behavior

**compose()** Compose multiple functions.

**lift()** Change the type of input a function takes. Also **lift\_dl**, **lift\_dv**, **lift\_ld**, **lift\_lv**, **lift\_vd**, **lift\_vl**.

**rerun()** Rerun expression n times.

**negate()** Negate a predicate function (a pipe friendly !).

**partial()** Create a version of a function that has some args preset to values.

**safely()** Modify func to return list of results whenever an error occurs (instead of error).

**quietly()** Modify function to return list of results, output, messages, warnings.

**possibly()** Modify function to return default value whenever an error occurs (instead of error).



# Nested Data

A **nested data frame** stores individual tables within the cells of a larger, organizing table.

"cell" contents			
Sepal.L	Sepal.W	Petal.L	Petal.W
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2

n\_iris\$data[[1]]

nested data frame		Species	data
setosa	setosa	setosa	<tibble [50 x 4]>
versicolor	versicolor	versicolor	<tibble [50 x 4]>
virginica	virginica	virginica	<tibble [50 x 4]>

n\_iris

Sepal.L	Sepal.W	Petal.L	Petal.W
7.0	3.2	4.7	1.4
6.4	3.2	4.5	1.5
6.9	3.1	4.9	1.5
5.5	2.3	4.0	1.3
6.5	2.8	4.6	1.5

n\_iris\$data[[2]]

Sepal.L	Sepal.W	Petal.L	Petal.W
6.3	3.3	6.0	2.5
5.8	2.7	5.1	1.9
7.1	3.0	5.9	2.1
6.3	2.9	5.6	1.8
6.5	3.0	5.8	2.2

n\_iris\$data[[3]]

Use a nested data frame to:

- preserve relationships between observations and subsets of data
- manipulate many sub-tables at once with the **purrr** functions **map()**, **map2()**, or **pmap()**.

Use a two step process to create a nested data frame:

1. Group the data frame into groups with **dplyr::group\_by()**
2. Use **nest()** to create a nested data frame with one row per group

Species   S.L   S.W   P.L   P.W	Species   S.L   S.W   P.L   P.W
setosa 5.1 3.5 1.4 0.2	setosa 5.1 3.5 1.4 0.2
setosa 4.9 3.0 1.4 0.2	setosa 4.9 3.0 1.4 0.2
setosa 4.7 3.2 1.3 0.2	setosa 4.7 3.2 1.3 0.2
setosa 4.6 3.1 1.5 0.2	setosa 4.6 3.1 1.5 0.2
setosa 5.0 3.6 1.4 0.2	setosa 5.0 3.6 1.4 0.2
versi 7.0 3.2 4.7 1.4	versi 7.0 3.2 4.7 1.4
versi 6.4 3.2 4.5 1.5	versi 6.4 3.2 4.5 1.5
versi 6.9 3.1 4.9 1.5	versi 6.9 3.1 4.9 1.5
versi 5.5 2.3 4.0 1.3	versi 5.5 2.3 4.0 1.3
versi 6.5 2.8 4.6 1.5	versi 6.5 2.8 4.6 1.5
virgini 6.3 3.3 6.0 2.5	virgini 6.3 3.3 6.0 2.5
virgini 5.8 2.7 5.1 1.9	virgini 5.8 2.7 5.1 1.9
virgini 7.1 3.0 5.9 2.1	virgini 7.1 3.0 5.9 2.1
virgini 6.3 2.9 5.6 1.8	virgini 6.3 2.9 5.6 1.8
virgini 6.5 3.0 5.8 2.2	virgini 6.5 3.0 5.8 2.2

n\_iris <- iris %>% group\_by(Species) %>% nest()

**tidy::nest(data, ..., .key = data)**

For grouped data, moves groups into cells as data frames.

Unnest a nested data frame with **unnest()**:

n\_iris %>% unnest()

**tidy::unnest(data, ..., .drop = NA, .id=NULL, .sep=NULL)**

Unnests a nested data frame.

# List Column Workflow

Nested data frames use a **list column**, a list that is stored as a column vector of a data frame. A typical **workflow** for list columns:

## 1 Make a list column

Species	S.L	S.W	P.L	P.W
setosa	5.1	3.5	1.4	0.2
setosa	4.9	3.0	1.4	0.2
setosa	4.7	3.2	1.3	0.2
setosa	4.6	3.1	1.5	0.2
setosa	5.0	3.6	1.4	0.2
versi	7.0	3.2	4.7	1.4
versi	6.4	3.2	4.5	1.5
versi	6.9	3.1	4.9	1.5
versi	5.5	2.3	4.0	1.3
versi	6.5	2.8	4.6	1.5
virgini	6.3	3.3	6.0	2.5
virgini	5.8	2.7	5.1	1.9
virgini	7.1	3.0	5.9	2.1
virgini	6.3	2.9	5.6	1.8
virgini	6.5	3.0	5.8	2.2

```
n_iris <- iris %>%  
  group_by(Species) %>%  
  nest()
```

## 2 Work with list columns

Species	data	model
setosa	<tibble [50x4]>	<S3: lm>
versi	<tibble [50x4]>	<S3: lm>
virgini	<tibble [50x4]>	<S3: lm>

```
mod_fun <- function(df)  
  lm(Sepal.Length ~ ., data = df)
```

```
m_iris <- n_iris %>%  
  mutate(model = map(data, mod_fun))
```

## 3 Simplify the list column

Species	beta
setos	2.35
versi	1.89
virgini	0.69

```
b_fun <- function(mod)  
  coefficients(mod)[[1]]
```

```
m_iris %>% transmute(Species,  
  beta = map_dbl(model, b_fun))
```

## 1. MAKE A LIST COLUMN

You can create list columns with functions in the **tibble** and **dplyr** packages, as well as **tidyR**'s **nest()**

**tibble::tribble(...)**

Makes list column when needed

max	seq
3	<code>int [3]</code>
4	<code>int [4]</code>
5	<code>int [5]</code>

**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 tibble with list cols

`enframe(list('3'=1:3, '4'=1:4, '5'=1:5), 'max', 'seq')`

## 2. WORK WITH LIST COLUMNS

Use the purrr functions **map()**, **map2()**, and **pmap()** to apply a function that returns a result element-wise to the cells of a list column. **walk()**, **walk2()**, and **pwalk()** work the same way, but return a side effect.

**purrr::map(.x, .f, ...)**

Apply `.f` element-wise to `.x` as `.f(x)`

`n_iris %>% mutate(n = map(data, dim))`

**purrr::map2(.x, .y, .f, ...)**

Apply `.f` element-wise to `.x` and `.y` as `.f(x, y)`

`m_iris %>% mutate(n = map2(data, model, list))`

**purrr::pmap(.l, .f, ...)**

Apply `.f` element-wise to vectors saved in `.l`

`m_iris %>% mutate(n = pmap(list(data, model, data), list))`

**map(data, fun, ...)**

Applies `fun` to each element of `data`

`map(<tibble [50x4]>, fun, ...)`

**map2(data, model, fun, ...)**

Applies `fun` to each element of `data` and `model`

`map2(<tibble [50x4]>, <S3: lm>, <S3: lm>, fun, ...)`

**pmap(list, fun, ...)**

Applies `fun` to each element of `list`

`pmap(list(<tibble [50x4]>, <S3: lm>, <S3: lm>, <S3: lm>), fun, ...)`

# R Markdown :: CHEAT SHEET

## What is R Markdown?

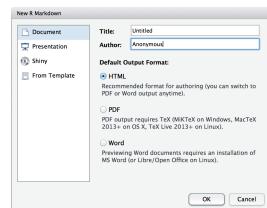


**.Rmd files** • An R Markdown (.Rmd) file is a record of your research. It contains the code that a scientist needs to reproduce your work along with the narration that a reader needs to understand your work.

**Reproducible Research** • At the click of a button, or the type of a command, you can rerun the code in an R Markdown file to reproduce your work and export the results as a finished report.

**Dynamic Documents** • You can choose to export the finished report in a variety of formats, including html, pdf, MS Word, or RTF documents; html or pdf based slides, Notebooks, and more.

## Workflow



① Open a new .Rmd file at File ► New File ► R Markdown. Use the wizard that opens to pre-populate the file with a template

② Write document by editing template

③ Knit document to create report; use knit button or render() to knit

④ Preview Output in IDE window

⑤ Publish (optional) to web server

⑥ Examine build log in R Markdown console

⑦ Use output file that is saved along side .Rmd

## render

Use rmarkdown::render() to render/knit at cmd line. Important args:

**input** - file to render  
**output\_format**

**output\_options** - List of render options (as in YAML)

**output\_file**  
**output\_dir**

**params** - list of params to use

**envir** - environment to evaluate code chunks in

**encoding** - of input file

## Embed code with knitr syntax

### INLINE CODE

Insert with `r <code>`. Results appear as text without code.

Built with `r getRVersion()` → Built with 3.2.3

### CODE CHUNKS

One or more lines surrounded with `{{r}}` and `{{ }}`. Place chunk options within curly braces, after r. Insert with {{getRVersion()}}

```
```{r echo=TRUE}
getRVersion()
```

```

### GLOBAL OPTIONS

Set with knitr::opts\_chunk\$set(), e.g.

```
```{r include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

```

### IMPORTANT CHUNK OPTIONS

**cache** - cache results for future knits (default = FALSE)

**cache.path** - directory to save cached results in (default = "cache/")

**child** - file(s) to knit and then include (default = NULL)

**collapse** - collapse all output into single block (default = FALSE)

**comment** - prefix for each line of results (default = "#")

**dependson** - chunk dependencies for caching (default = NULL)

**echo** - Display code in output document (default = TRUE)

**engine** - code language used in chunk (default = 'R')

**error** - Display error messages in doc (TRUE) or stop render when errors occur (FALSE) (default = FALSE)

**eval** - Run code in chunk (default = TRUE)

Options not listed above: R.options, aniopts, autodep, background, cache.comments, cache.lazy, cache.rebuild, cache.vars, dev, dev.args, dpi, engine.opts, engine.path, fig.asp, fig.env, fig.ext, fig.keep, fig.lp, fig.path, fig.pos, fig.process, fig.retina, fig.scap, fig.show, fig.showtext, fig.subcap, interval, out.extra, out.height, out.width, prompt, purl, ref.label, render, size, split, tidy.opts

**fig.align** - 'left', 'right', or 'center' (default = 'default')

**fig.cap** - figure caption as character string (default = NULL)

**fig.height, fig.width** - Dimensions of plots in inches

**highlight** - highlight source code (default = TRUE)

**include** - Include chunk in doc after running (default = TRUE)

**message** - display code messages in document (default = TRUE)

**results** (default = 'markup')

'asis' - passthrough results

'hide' - do not display results

'hold' - put all results below all code

**tidy** - tidy code for display (default = FALSE)

**warning** - display code warnings in document (default = TRUE)

## .rmd Structure



### YAML Header

Optional section of render (e.g. pandoc) options written as key:value pairs (YAML).

At start of file

Between lines of ---

### Text

Narration formatted with markdown, mixed with:

### Code Chunks

Chunks of embedded code. Each chunk:

Begins with `{{r}}`

ends with `{{ }}`

R Markdown will run the code and append the results to the doc.

It will use the location of the .Rmd file as the working directory

## Parameters

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

1. **Add parameters** • Create and set parameters in the header as sub-values of params

```
---
params:
  n: 100
  d: ! Sys.Date()
---
```

2. **Call parameters** • Call parameter values in code as params\$<name>

Today's date is `r params\$d`

3. **Set parameters** • Set values with Knit with parameters or the params argument of render():

render("doc.Rmd", params = list(n = 1, d = as.Date("2015-01-01")))

Knit to HTML  
Knit to PDF  
Knit to Word  
Knit with Parameters...

## Interactive Documents

Turn your report into an interactive Shiny document in 4 steps

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

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

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

```

How many cars?  
5  
speed dist  
1 4.00 2.00  
2 4.00 10.00  
3 7.00 4.00  
4 7.00 22.00  
5 8.00 16.00

Embed a complete app into your document with shiny::shinyAppDir()

NOTE: Your report will be rendered as a Shiny app, which means you must choose an html output format, like html\_document, and serve it with an active R Session.





# Pandoc's Markdown

Write with syntax on the left to create effect on right (after render)

```
Plain text
End a line with two spaces
to start a new paragraph.
*italics* and **bold**
`verbatim` code
sub/superscript22
~~strikethrough~~
escaped: `*` \\
endash: --, emdash: ---
equation: $A = \pi * r^2$
```

```
equation block:
```

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

```
> block quote
```

```
# Header1 {#anchor}
## Header 2 {#css_id}
### Header 3 {.css_class}
```

```
#### Header 4
```

```
##### Header 5
```

```
##### Header 6
```

```
<!--Text comment-->
```

```
\textbf{Text ignored in HTML}
<em>HTML ignored in pdfs</em>
```

```
<http://www.rstudio.com>
[link](www.rstudio.com)
Jump to [Header 1]{#anchor}
image:
```

```
![Caption](smallorb.png)
```

```
* unordered list
+ sub-item 1
+ sub-item 2
- sub-sub-item 1
```

```
* item 2
```

```
Continued (indent 4 spaces)
```

```
1. ordered list
```

```
2. item 2
i) sub-item 1
A. sub-sub-item 1
```

```
(@) A list whose numbering
```

```
continues after
```

```
2. an interruption
```

```
Term 1
```

```
Definition 1
```

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

```
- slide bullet 1
- slide bullet 2
```

```
(>- to have bullets appear on click)
```

```
horizontal rule/slide break:
```

```
***
```

```
A footnote [^1]
```

```
[^1]: Here is the footnote.
```



# Set render options with YAML

When you render, R Markdown

1. runs the R code, embeds results and text into .md file with knitr
2. then converts the .md file into the finished format with pandoc



Set a document's default output format in the YAML header:

```
---  
output: html_document  
---  
# Body
```

## output value

## creates

|                       |                                  |
|-----------------------|----------------------------------|
| html_document         | html                             |
| pdf_document          | pdf (requires Tex)               |
| word_document         | Microsoft Word (.docx)           |
| odt_document          | OpenDocument Text                |
| rtf_document          | Rich Text Format                 |
| md_document           | Markdown                         |
| github_document       | Github compatible markdown       |
| ioslides_presentation | ioslides HTML slides             |
| slidy_presentation    | slidy HTML slides                |
| beamer_presentation   | Beamer pdf slides (requires Tex) |

Customize output with sub-options (listed to the right):

```
---  
output: html_document:  
  code_folding: hide  
  toc_float: TRUE  
---  
# Body
```

## html tabs

Use tablet css class to place sub-headers into tabs

```
# Tabset {.tabset .tabset-fade .tabset-pills}  
## Tab 1  
text 1  
## Tab 2  
text 2  
### End tabset
```



# Create a Reusable Template

1. **Create a new package** with a `inst/rmarkdown/templates` directory

2. In the directory, **Place a folder** that contains:

**template.yaml** (see below)

**skeleton.Rmd** (contents of the template)

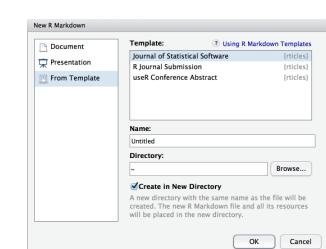
any supporting files

3. **Install the package**

4. **Access template** in wizard at File ▶ New File ▶ R Markdown template.yaml

A footnote <sup>1</sup>

1. Here is the footnote.<sup>1</sup>



## sub-option

## description

|                       |   | html | pdf | word | odt | rtf | md | gitlab | ioslides | slidy | beamer |
|-----------------------|---|------|-----|------|-----|-----|----|--------|----------|-------|--------|
| citation_package      | The LaTeX package to process citations, natbib, biblatex or none                | X    |     |      |     |     |    |        |          |       |        |
| code_folding          | Let readers to toggle the display of R code, "none", "hide", or "show"          |      | X   |      |     |     |    |        |          |       |        |
| colortheme            | Beamer color theme to use   |      |     |      |     |     |    |        |          |       | X      |
| css                   | CSS file to use to style document   |      | X   |      |     |     |    |        | X        | X     |        |
| dev                   | Graphics device to use for figure output (e.g. "png")                           |      | X   | X    |     |     |    |        | X        | X     | X      |
| duration              | Add a countdown timer (in minutes) to footer of slides                          |      |     |      |     |     |    |        |          |       | X      |
| fig_caption           | Should figures be rendered with captions?                                       | X    | X   | X    | X   |     |    |        | X        | X     | X      |
| fig_height, fig_width | Default figure height and width (in inches) for document                        | X    | X   | X    | X   | X   | X  | X      | X        | X     | X      |
| highlight             | Syntax highlighting: "tango", "pygments", "kate", "zenburn", "textmate"         | X    | X   | X    |     |     |    |        | X        | X     |        |
| includes              | File of content to place in document (in_header, before_body, after_body)       | X    | X   | X    | X   | X   | X  | X      | X        | X     |        |
| incremental           | Should bullets appear one at a time (on presenter mouse clicks)?                |      |     |      |     |     |    |        | X        | X     | X      |
| keep_md               | Save a copy of .md file that contains knitr output                              |      | X   | X    | X   | X   |    |        | X        | X     |        |
| keep_tex              | Save a copy of .tex file that contains knitr output                             |      |     |      |     |     |    |        |          |       | X      |
| latex_engine          | Engine to render latex, "pdflatex", "xelatex", or "lualatex"                    |      |     |      |     |     |    |        |          |       | X      |
| lib_dir               | Directory of dependency files to use (Bootstrap, MathJax, etc.)                 |      | X   |      |     |     |    |        | X        | X     |        |
| mathjax               | Set to local or a URL to use a local/URL version of MathJax to render equations |      | X   |      |     |     |    |        | X        | X     |        |
| md_extensions         | Markdown extensions to add to default definition or R Markdown                  | X    | X   | X    | X   | X   | X  | X      | X        | X     |        |
| number_sections       | Add section numbering to headers  |      | X   | X    |     |     |    |        |          |       |        |
| pandoc_args           | Additional arguments to pass to Pandoc  | X    | X   | X    | X   | X   | X  | X      | X        | X     |        |
| preserve_yaml         | Preserve YAML front matter in final document?                                   |      |     |      |     |     |    |        |          |       | X      |
| reference_docx        | docx file whose styles should be copied when producing docx output              |      |     |      |     |     |    |        |          |       | X      |
| self_contained        | Embed dependencies into the doc   |      |     |      |     |     |    |        |          |       | X      |
| slide_level           | The lowest heading level that defines individual slides                         |      |     |      |     |     |    |        |          |       | X      |
| smaller               | Use the smaller font size in the presentation?                                  |      |     |      |     |     |    |        |          |       | X      |
| smart                 | Convert straight quotes to curly, dashes to em-dashes, ... to ellipses, etc.    | X    |     |      |     |     |    |        |          | X     | X      |
| template              | Pandoc template to use when rendering file quarterly_report.html                | X    | X   | X    |     |     |    |        |          | X     | X      |
| theme                 | Bootswatch or Beamer theme to use for page                                      | X    |     |      |     |     |    |        |          |       | X      |
| toc                   | Add a table of contents at start of document                                    | X    | X   | X    | X   | X   | X  | X      | X        | X     |        |
| toc_depth             | The lowest level of headings to add to table of contents                        | X    | X   | X    | X   | X   | X  | X      | X        | X     |        |
| toc_float             | Float the table of contents to the left of the main content                     | X    |     |      |     |     |    |        |          |       |        |

# Table Suggestions

Several functions format R data into tables

| Table with kable    |      |
|---------------------|------|
| <code>erupts</code> |      |
| 1                   | 3.60 |
| 2                   | 1.80 |
| 3                   | 3.33 |
| 4                   | 2.28 |
|                     | 79   |
|                     | 54   |
|                     | 74   |
|                     | 62   |

| erupts  |      |
|---------|------|
| waiting |      |
| 1       | 3.60 |
| 2       | 1.80 |
| 3       | 3.33 |
| 4       | 2.28 |
|         | 79   |
|         | 54   |
|         | 74   |
|         | 62   |

| Table with stargazer |       |
|----------------------|-------|
| <code>erupts</code>  |       |
| waiting              |       |
| 1                    | 3.600 |
| 2                    | 1.800 |
| 3                    | 3.333 |
| 4                    | 2.283 |
|                      | 79    |
|                      | 54    |
|                      | 74    |
|                      | 62    |

```
data <- faithful[1:4,]
```
{r results = 'asis'}
knitr::kable(data, caption = "Table with kable")
```
```
{r results = 'asis'}
print(xtable::xtable(data, caption = "Table with xtable"),
      type = "html", html.table.attributes = "border=0")
```
```
{r results = 'asis'}
stargazer::stargazer(data, type = "html", title = "Table with stargazer")
```

```

Learn more in the [stargazer](#), [xtable](#), and [knitr](#) packages.

# Citations and Bibliographies

Create citations with .bib, .bibtex, .copac, .enl, .json, .medline, .mods, .ris, .wos, and .xml files

1. Set **bibliography file** and CSL 1.0

Style file (optional) in the YAML header

`bibliography: refs.bib</code`



# R Markdown Reference Guide

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

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

Contents:

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

## Syntax

Plain text

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

\*italics\* and \_italics\_

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

superscript<sup>2</sup>

~~strikethrough~~

[link](www.rstudio.com)

# Header 1

## Header 2

### Header 3

#### Header 4

##### Header 5

##### Header 6

endash: --

emdash: ---

ellipsis: ...

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

image: 

horizontal rule (or slide break):

\*\*\*

> block quote

\* unordered list

\* item 2

+ sub-item 1

+ sub-item 2

1. ordered list

2. item 2

+ sub-item 1

+ sub-item 2

| Table Header | Second Header |
|--------------|---------------|
|--------------|---------------|

|            |        |
|------------|--------|
| Table Cell | Cell 2 |
|------------|--------|

|        |        |
|--------|--------|
| Cell 3 | Cell 4 |
|--------|--------|

## Becomes

Plain text

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

*italics* and *italics*

**bold** and **bold**

superscript<sup>2</sup>

~~strikethrough~~

[link](#)

## Header 1

## Header 2

## Header 3

## Header 4

### Header 5

### Header 6

endash: –

emdash: —

ellipsis: ...

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



horizontal rule (or slide break):

block quote

- unordered list

- item 2

- sub-item 1

- sub-item 2

- 1. ordered list

- 2. item 2

- sub-item 1

- sub-item 2

| Table Header | Second Header |
|--------------|---------------|
|--------------|---------------|

|            |        |
|------------|--------|
| Table Cell | Cell 2 |
|------------|--------|

|        |        |
|--------|--------|
| Cell 3 | Cell 4 |
|--------|--------|



# R Markdown Reference Guide

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

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

## Contents:

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

## Syntax

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

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

```

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

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

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

```

## Becomes

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

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

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

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

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

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

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

## Chunk options

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



# R Markdown Reference Guide

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

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

## Contents:

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

## Chunk options (Continued)

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



# R Markdown Reference Guide

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

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

## Contents:

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

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

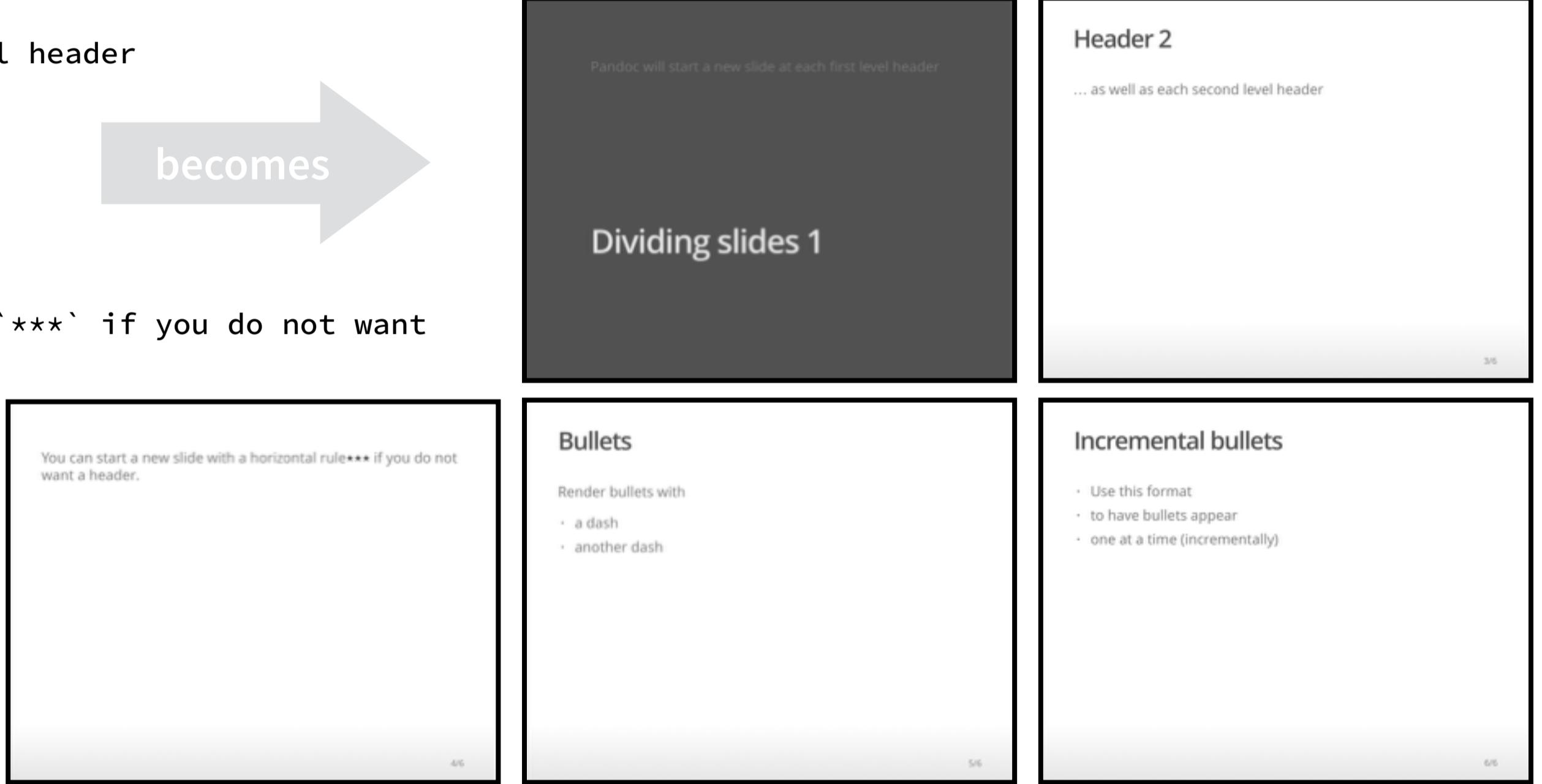
## Syntax for slide formats (ioslides, slidify, beamer)

```
# Dividing slides 1
Pandoc will start a new slide at each first level header
## Header 2
... as well as each second level header
***
You can start a new slide with a horizontal rule`***` if you do not want
a header.

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

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

becomes



## Slide display modes

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

### ioslides

- |          |                              |
|----------|------------------------------|
| <b>f</b> | - enable fullscreen mode     |
| <b>w</b> | - toggle widescreen mode     |
| <b>o</b> | - enable overview mode       |
| <b>h</b> | - enable code highlight mode |
| <b>p</b> | - show presenter notes       |

### slidy

- |          |   |
|----------|---|
| <b>C</b> | - show table of contents                  |
| <b>F</b> | - toggle display of the footer            |
| <b>A</b> | - toggle display of current vs all slides |
| <b>S</b> | - make fonts smaller                      |
| <b>B</b> | - make fonts bigger                       |

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

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



# R Markdown Reference Guide

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

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

## Contents:

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

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



# Shiny :: CHEAT SHEET

## Basics

A **Shiny** app is a web page (**UI**) connected to a computer running a live R session (**Server**)



Users can manipulate the UI, which will cause the server to update the UI's displays (by running R code).

## APP TEMPLATE

Begin writing a new app with this template. Preview the app by running the code at the R command line.

```
library(shiny)
ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist"))
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}
shinyApp(ui = ui, server = server)
```

- **ui** - nested R functions that assemble an HTML user interface for your app
- **server** - a function with instructions on how to build and rebuild the R objects displayed in the UI
- **shinyApp** - combines **ui** and **server** into an app. Wrap with **runApp()** if calling from a sourced script or inside a function.

## SHARE YOUR APP

The easiest way to share your app is to host it on shinyapps.io, a cloud based service from RStudio

1. Create a free or professional account at <http://shinyapps.io>
2. Click the **Publish** icon in the RStudio IDE or run:  
`rsconnect::deployApp("<path to directory>")`

**Build or purchase your own Shiny Server**  
at [www.rstudio.com/products/shiny-server/](http://www.rstudio.com/products/shiny-server/)



## Building an App

Complete the template by adding arguments to `fluidPage()` and a body to the server function.

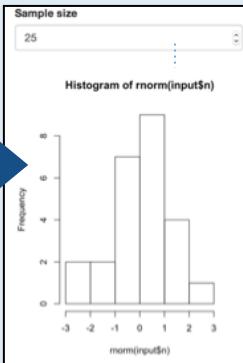
Add inputs to the UI with `*Input()` functions

Add outputs with `*Output()` functions

Tell server how to render outputs with R in the server function. To do this:

1. Refer to outputs with `output$<id>`
2. Refer to inputs with `input$<id>`
3. Wrap code in a `render*>()` function before saving to output

```
library(shiny)
ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist"))
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}
shinyApp(ui = ui, server = server)
```



Save your template as **app.R**. Alternatively, split your template into two files named **ui.R** and **server.R**.

```
library(shiny)
ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist"))
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}
shinyApp(ui = ui, server = server)
```

```
# ui.R
fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist"))

# server.R
function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}
```

**ui.R** contains everything you would save to ui.

**server.R** ends with the function you would save to server.

No need to call **shinyApp()**.

Save each app as a directory that holds an **app.R** file (or a **server.R** file and a **ui.R** file) plus optional extra files.

● ● ● **app-name**

- app.R** (optional) defines objects available to both ui.R and server.R
- global.R** (optional) used in showcase mode
- DESCRIPTION** (optional) data, scripts, etc.
- README** (optional) directory of files to share with web browsers (images, CSS, .js, etc.) Must be named "**www**"
- <other files>
- www**

The directory name is the name of the app  
(optional) defines objects available to both ui.R and server.R  
(optional) used in showcase mode  
(optional) data, scripts, etc.  
(optional) directory of files to share with web browsers (images, CSS, .js, etc.) Must be named "www"

Launch apps with  
`runApp(<path to directory>)`

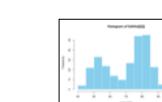
## Outputs - `render*`() and `*Output()` functions work together to add R output to the UI



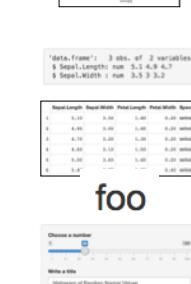
**DT::renderDataTable(expr, options, callback, escape, env, quoted)**



**renderImage(expr, env, quoted, deleteFile)**



**renderPlot(expr, width, height, res, ..., env, quoted, func)**



**renderPrint(expr, env, quoted, func, width)**

**renderTable(expr, ..., env, quoted, func)**

**renderText(expr, env, quoted, func)**

**renderUI(expr, env, quoted, func)**

works with

**dataTableOutput(outputId, icon, ...)**

**imageOutput(outputId, width, height, click, dblclick, hover, hoverDelay, inline, hoverDelayType, brush, clickId, hoverId)**

**plotOutput(outputId, width, height, click, dblclick, hover, hoverDelay, inline, hoverDelayType, brush, clickId, hoverId)**

**verbatimTextOutput(outputId)**

**tableOutput(outputId)**

**textOutput(outputId, container, inline)**

**uiOutput(outputId, inline, container, ...)**

**htmlOutput(outputId, inline, container, ...)**

## Inputs

collect values from the user

Access the current value of an input object with `input$<inputId>`. Input values are **reactive**.

Action

**ActionButton**(inputId, label, icon, ...)

Link

**actionLink**(inputId, label, icon, ...)

- Choice 1
- Choice 2
- Choice 3
- Check me

**checkboxInput**(inputId, label, value)

**dateInput**(inputId, label, value, min, max, format, startview, weekstart, language)

**dateRangeInput**(inputId, label, start, end, min, max, format, startview, weekstart, language, separator)

**fileInput**(inputId, label, multiple, accept)

**numericInput**(inputId, label, value, min, max, step)

**passwordInput**(inputId, label, value)

**radioButtons**(inputId, label, choices, selected, inline)

**selectInput**(inputId, label, choices, selected, multiple, selectize, width, size) (also **selectizeInput()**)

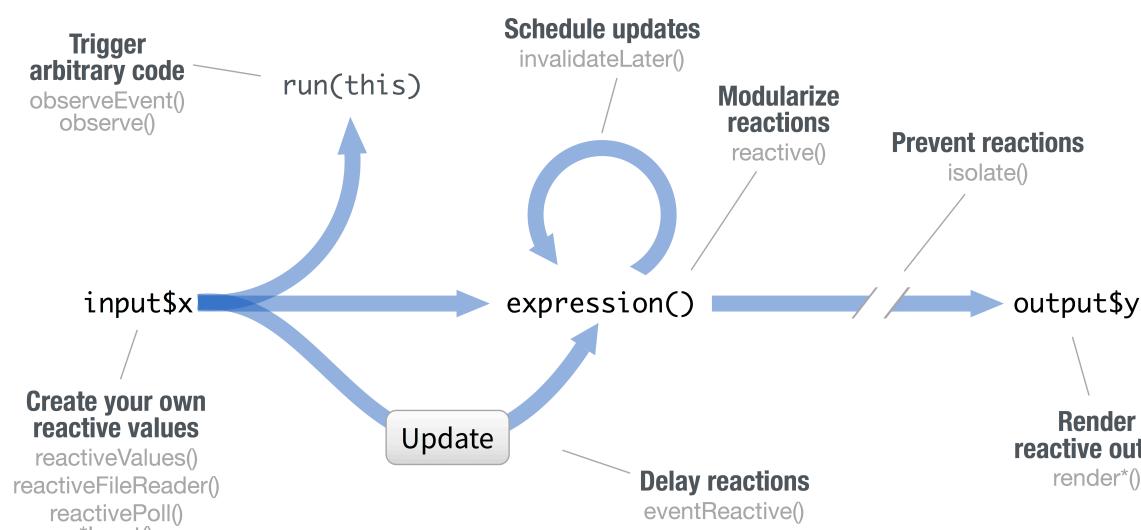
**sliderInput**(inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post)

**submitButton**(text, icon) (Prevents reactions across entire app)

**textInput**(inputId, label, value)

# Reactivity

Reactive values work together with reactive functions. Call a reactive value from within the arguments of one of these functions to avoid the error **Operation not allowed without an active reactive context**.



## CREATE YOUR OWN REACTIVE VALUES

```
# example snippets
ui <- fluidPage(
  textInput("a","","A"))

server <-
function(input,output){
  rv <- reactiveValues()
  rv$number <- 5
}

reactiveValues() creates a list of reactive values whose values you can set.
```

## PREVENT REACTIONS

```
library(shiny)
ui <- fluidPage(
  textInput("a","","A"),
  textOutput("b"))

server <-
function(input,output){
  output$b <-
  renderText({
    isolate({input$a})
  })
}

shinyApp(ui, server)
```

**isolate(expr)**  
Runs a code block. Returns a **non-reactive** copy of the results.

## RENDERS REACTIVE OUTPUT

```
library(shiny)
ui <- fluidPage(
 textInput("a","","A"),
  textOutput("b"))

server <-
function(input,output){
  output$b <-
  renderText({
    input$a
  })
}

shinyApp(ui, server)
```

**render\*() functions**  
(see front page)

Builds an object to display. Will rerun code in body to rebuild the object whenever a reactive value in the code changes.  
Save the results to **output\$<outputId>**

## TRIGGER ARBITRARY CODE

```
library(shiny)
ui <- fluidPage(
  textInput("a","","A"),
  actionButton("go","Go"))

server <-
function(input,output){
  observeEvent(input$go,{
    print(input$a)
  })
}

shinyApp(ui, server)
```

**observeEvent(eventExpr, handlerExpr, event.env, event.quoted, handler.env, handler.quoted, label, suspended, priority, domain, autoDestroy, ignoreNULL)**

Runs code in 2nd argument when reactive values in 1st argument change. See **observe()** for alternative.

## MODULARIZE REACTIONS

```
ui <- fluidPage(
  textInput("a","","A"),
  textInput("z","","Z"),
  textOutput("b"))

server <-
function(input,output){
  re <- reactive({
    paste(input$a,input$z)
  })
  output$b <-
  renderText({
    re()
  })
}

shinyApp(ui, server)
```

**reactive(x, env, quoted, label, domain)**  
Creates a **reactive expression** that  
• **caches** its value to reduce computation  
• can be called by other code  
• notifies its dependencies when it has been invalidated  
Call the expression with function syntax, e.g. **re()**

## DELAY REACTIONS

```
library(shiny)
ui <- fluidPage(
  textInput("a","","A"),
  actionButton("go","Go"),
  textOutput("b"))

server <-
function(input,output){
  re <- eventReactive(
    input$go,{input$a})
  output$b <-
  renderText({
    re()
  })
}

shinyApp(ui, server)
```

**eventReactive(eventExpr, valueExpr, event.env, event.quoted, value.env, value.quoted, label, domain, ignoreNULL)**

Creates reactive expression with code in 2nd argument that only invalidates when reactive values in 1st argument change.

## UI

An app's UI is an HTML document.

Use Shiny's functions to assemble this HTML with R.

```
fluidPage(
  textInput("a",""))
## <div class="container-fluid">
##   <div class="form-group shiny-input-container">
##     <label for="a"></label>
##     <input id="a" type="text"
##           class="form-control" value="" />
##   </div>
## </div>
```



Add static HTML elements with **tags**, a list of functions that parallel common HTML tags, e.g. **tags\$a()**. Unnamed arguments will be passed into the tag; named arguments will become tag attributes.

|                  |                  |               |                |                |
|------------------|------------------|---------------|----------------|----------------|
| tags\$a          | tags\$data       | tags\$h6      | tags\$nav      | tags\$span     |
| tags\$abbr       | tags\$datalist   | tags\$head    | tags\$noscript | tags\$strong   |
| tags\$address    | tags\$dd         | tags\$header  | tags\$object   | tags\$style    |
| tags\$area       | tags\$del        | tags\$group   | tags\$ol       | tags\$sub      |
| tags\$article    | tags\$details    | tags\$hr      | tags\$optgroup | tags\$summary  |
| tags\$aside      | tags\$dfn        | tags\$HTML    | tags\$option   | tags\$sup      |
| tags\$audio      | tags\$div        | tags\$i       | tags\$output   | tags\$table    |
| tags\$b          | tags\$dl         | tags\$iframe  | tags\$p        | tags\$tbody    |
| tags\$base       | tags\$dt         | tags\$img     | tags\$param    | tags\$td       |
| tags\$bdi        | tags\$em         | tags\$input   | tags\$pre      | tags\$textarea |
| tags\$bdo        | tags\$embed      | tags\$ins     | tags\$progress | tags\$tfoot    |
| tags\$blockquote | tags\$events     | tags\$kbd     | tags\$q        | tags\$th       |
| tags\$body       | tags\$fieldset   | tags\$keygen  | tags\$ruby     | tags\$thead    |
| tags\$br         | tags\$figcaption | tags\$label   | tags\$rp       | tags\$time     |
| tags\$button     | tags\$figure     | tags\$legend  | tags\$rt       | tags\$title    |
| tags\$canvas     | tags\$footer     | tags\$li      | tags\$ss       | tags\$track    |
| tags\$caption    | tags\$form       | tags\$link    | tags\$small    | tags\$u        |
| tags\$cite       | tags\$h1         | tags\$mark    | tags\$meta     | tags\$ul       |
| tags\$code       | tags\$h2         | tags\$map     | tags\$meter    | tags\$var      |
| tags\$col        | tags\$h3         | tags\$section | tags\$source   | tags\$video    |
| tags\$colgroup   | tags\$h4         | tags\$select  |                | tags\$wbr      |
| tags\$command    | tags\$h5         | tags\$meta    |                |                |

The most common tags have wrapper functions. You do not need to prefix their names with **tags\$**

```
ui <- fluidPage(
  h1("Header 1"),
  hr(),
  br(),
  p(strong("bold")),
  p(em("italic")),
  p(code("code")),
  a(href="", "link"),
  HTML("<p>Raw html</p>"))
```



To include a CSS file, use **includeCSS()**, or  
1. Place the file in the **www** subdirectory  
2. Link to it with

```
tags$head(tags$link(rel = "stylesheet",
  type = "text/css", href = "<file name>"))
```



To include JavaScript, use **includeScript()** or  
1. Place the file in the **www** subdirectory  
2. Link to it with

```
tags$head(tags$script(src = "<file name>"))
```



To include an image  
1. Place the file in the **www** subdirectory  
2. Link to it with **img(src=<file name>")**

## Layouts

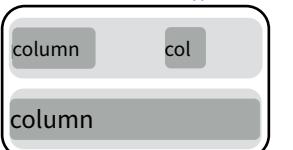


Combine multiple elements into a "single element" that has its own properties with a panel function, e.g.

```
wellPanel(dateInput("a", ""),
  submitButton())
```

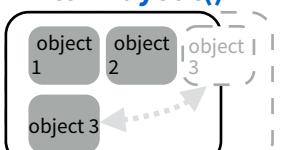
Organize panels and elements into a layout with a layout function. Add elements as arguments of the layout functions.

### fluidRow()



```
ui <- fluidPage(
  fluidRow(column(width = 4),
    column(width = 2, offset = 3)),
  fluidRow(column(width = 12)))
```

### flowLayout()



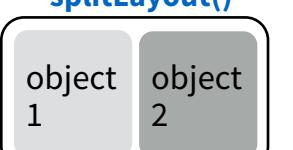
```
ui <- fluidPage(
  flowLayout(object 1,
    object 2,
    object 3))
```

### sidebarLayout()



```
ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(),
    mainPanel()))
```

### splitLayout()

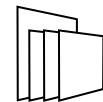


```
ui <- fluidPage(
  splitLayout(object 1,
    object 2))
```

### verticalLayout()



```
ui <- fluidPage(
  verticalLayout(object 1,
    object 2,
    object 3))
```



Layer tabPanels on top of each other, and navigate between them, with:

```
ui <- fluidPage(tabsetPanel(
  tabPanel("tab 1", "contents"),
  tabPanel("tab 2", "contents"),
  tabPanel("tab 3", "contents")))
```

```
ui <- fluidPage(navlistPanel(
  tabPanel("tab 1", "contents"),
  tabPanel("tab 2", "contents"),
  tabPanel("tab 3", "contents")))
```

```
ui <- navbarPage(title = "Page",
  tabPanel("tab 1", "contents"),
  tabPanel("tab 2", "contents"),
  tabPanel("tab 3", "contents"))
```

# Data Science in Spark with sparklyr :: CHEAT SHEET

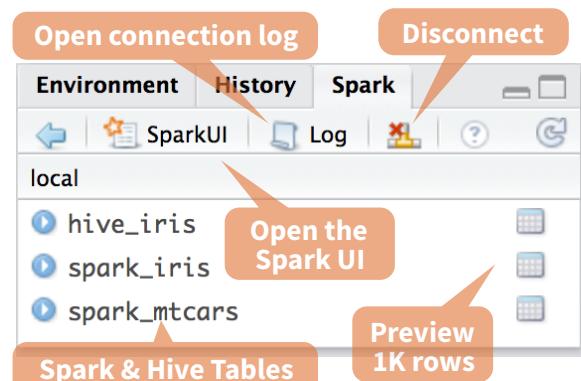


## Intro

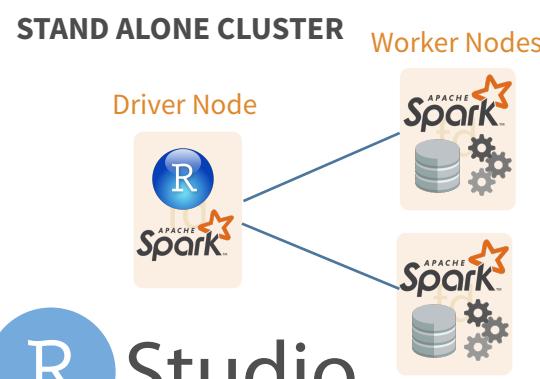
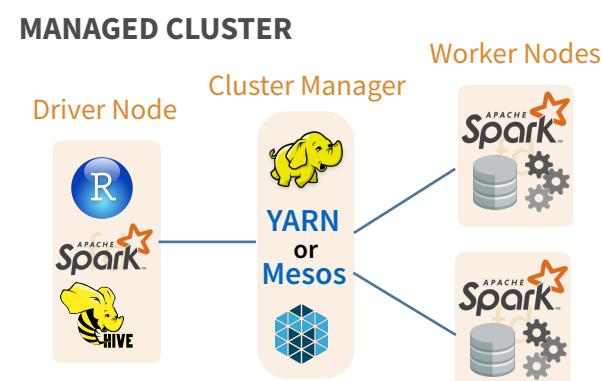
**sparklyr** is an R interface for Apache Spark™, it provides a complete **dplyr** backend and the option to query directly using **Spark SQL** statement. With sparklyr, you can orchestrate distributed machine learning using either **Spark's MLLib** or **H2O** Sparkling Water.

Starting with **version 1.044, RStudio Desktop, Server and Pro include integrated support for the sparklyr package**. You can create and manage connections to Spark clusters and local Spark instances from inside the IDE.

### RStudio Integrates with sparklyr

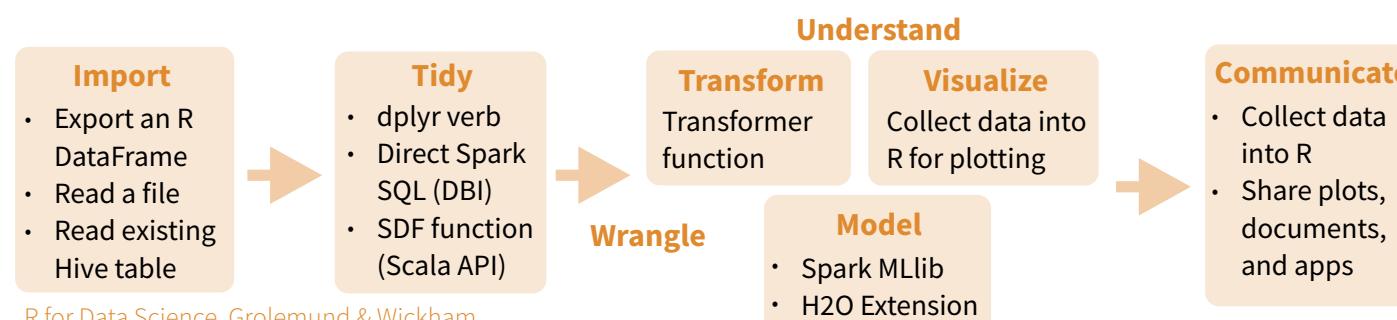


## Cluster Deployment



R Studio

## Data Science Toolchain with Spark + sparklyr



## Getting Started

### LOCAL MODE (No cluster required)

1. Install a local version of Spark:  
`spark_install ("2.0.1")`
2. Open a connection  
`sc <- spark_connect (master = "local")`

### ON A MESOS MANAGED CLUSTER

1. Install RStudio Server or Pro on one of the existing nodes
2. Locate path to the cluster's Spark directory, it normally is "/usr/lib/spark"
3. Open a connection  
`spark_connect(master="mesos URL", version = "1.6.2", spark_home = [Cluster's Spark path])`

### USING LIVY (Experimental)

1. The Livy REST application should be running on the cluster
2. Connect to the cluster  
`sc <- spark_connect(method = "livy", master = "http://host:port")`

## Tuning Spark

### EXAMPLE CONFIGURATION

```
config <- spark_config()  
config$spark.executor.cores <- 2  
config$spark.executor.memory <- "4G"  
sc <- spark_connect (master="yarn-client",  
  config = config, version = "2.0.1")
```

### IMPORTANT TUNING PARAMETERS with defaults

- spark.yarn.am.cores
- spark.yarn.am.memory **512m**
- spark.network.timeout **120s**
- spark.executor.memory **1g**
- spark.executor.cores **1**
- spark.executor.instances
- spark.executor.extraJavaOptions
- spark.executor.heartbeatInterval **10s**
- sparklyr.shell.executor-memory
- sparklyr.shell.driver-memory

## Using sparklyr

A brief example of a data analysis using Apache Spark, R and sparklyr in local mode

```
library(sparklyr); library(dplyr); library(ggplot2);  
library(tidyr);  
set.seed(100)
```

**Install Spark locally**

```
spark_install("2.0.1")
```

**Connect to local version**

```
sc <- spark_connect(master = "local")
```

```
import_iris <- copy_to(sc, iris, "spark_iris",  
  overwrite = TRUE)
```

**Copy data to Spark memory**

```
partition_iris <- sdf_partition(  
  import_iris, training=0.5, testing=0.5)
```

**Partition data**

```
sdf_register(partition_iris,  
c("spark_iris_training","spark_iris_test"))
```

**Create a hive metadata for each partition**

```
tidy_iris <-tbl(sc,"spark_iris_training") %>%  
  select(Species, Petal_Length, Petal_Width)
```

**Spark ML Decision Tree Model**

```
model_iris <- tidy_iris %>%  
  ml_decision_tree(response="Species",  
    features=c("Petal_Length","Petal_Width"))
```

```
test_iris <-tbl(sc,"spark_iris_test")
```

**Create reference to Spark table**

```
pred_iris <- sdf_predict(  
  model_iris, test_iris) %>%  
  collect
```

```
pred_iris %>%  
  inner_join(data.frame(prediction=0:2,  
    lab=model_iris$model.parameters$labels)) %>%  
  ggplot(aes(Petal_Length, Petal_Width, col=lab)) +  
  geom_point()
```

**Bring data back into R memory for plotting**

```
spark_disconnect(sc)
```

**Disconnect**



# Reactivity

## COPY A DATA FRAME INTO SPARK

`sdf_copy_to(sc, iris, "spark_iris")`

`sdf_copy_to(sc, x, name, memory, repartition, overwrite)`

## IMPORT INTO SPARK FROM A FILE

Arguments that apply to all functions:

`sc, name, path, options = list(), repartition = 0, memory = TRUE, overwrite = TRUE`

**CSV** `spark_read_csv(header = TRUE, columns = NULL, infer_schema = TRUE, delimiter = "", quote = "", escape = "\\", charset = "UTF-8", null_value = NULL)`

**JSON** `spark_read_json()`

**PARQUET** `spark_read_parquet()`

## SPARK SQL COMMANDS

`DBI::dbWriteTable(sc, "spark_iris", iris)`

`DBI::dbWriteTable(conn, name, value)`

## FROM A TABLE IN HIVE

`my_var <- tbl_cache(sc, name = "hive_iris")`

`tbl_cache(sc, name, force = TRUE)`  
Loads the table into memory

`my_var <- dplyr::tbl(sc, name = "hive_iris")`

`dplyr::tbl(sc, ...)`  
Creates a reference to the table without loading it into memory

# Wrangle

## SPARK SQL VIA DPLYR VERBS

Translates into Spark SQL statements

`my_table <- my_var %>%  
filter(Species == "setosa") %>%  
sample_n(10)`

## DIRECT SPARK SQL COMMANDS

`my_table <- DBI::dbGetQuery(sc, "SELECT *  
FROM iris LIMIT 10")`

`DBI::dbGetQuery(conn, statement)`

## SCALA API VIA SDF FUNCTIONS

`sdf_mutate(.data)`

Works like dplyr mutate function

`sdf_partition(x, ..., weights = NULL, seed = sample (.Machine$integer.max, 1))`

`sdf_partition(x, training = 0.5, test = 0.5)`

`sdf_register(x, name = NULL)`

Gives a Spark DataFrame a table name

`sdf_sample(x, fraction = 1, replacement = TRUE, seed = NULL)`

`sdf_sort(x, columns)`  
Sorts by >=1 columns in ascending order

`sdf_with_unique_id(x, id = "id")`

`sdf_predict(object, newdata)`  
Spark DataFrame with predicted values

## ML TRANSFORMERS

`ft_binarizer(my_table, input.col = "Petal_Length", output.col = "petal_large", threshold = 1.2)`

Arguments that apply to all functions:  
`x, input.col = NULL, output.col = NULL`

`ft_binarizer(threshold = 0.5)`  
Assigned values based on threshold

`ft_bucketizer(splits)`  
Numeric column to discretized column

`ft_discrete_cosine_transform(inverse = FALSE)`  
Time domain to frequency domain

`ft_elementwise_product(scaling.col)`  
Element-wise product between 2 cols

`ft_index_to_string()`  
Index labels back to label as strings

`ft_one_hot_encoder()`  
Continuous to binary vectors

`ft_quantile_discretizer(n.buckets = 5L)`  
Continuous to binned categorical values

`ft_sql_transformer(sql)`

`ft_string_indexer(params = NULL)`  
Column of labels into a column of label indices.

`ft_vectorAssembler()`  
Combine vectors into single row-vector

# Visualize & Communicate

## DOWNLOAD DATA TO R MEMORY

`r_table <- collect(my_table)`

`plot(Petal_Width ~ Petal_Length, data = r_table)`

`dplyr::collect(x)`

Download a Spark DataFrame to an R DataFrame

`sdf_read_column(x, column)`

Returns contents of a single column to R

## SAVE FROM SPARK TO FILE SYSTEM

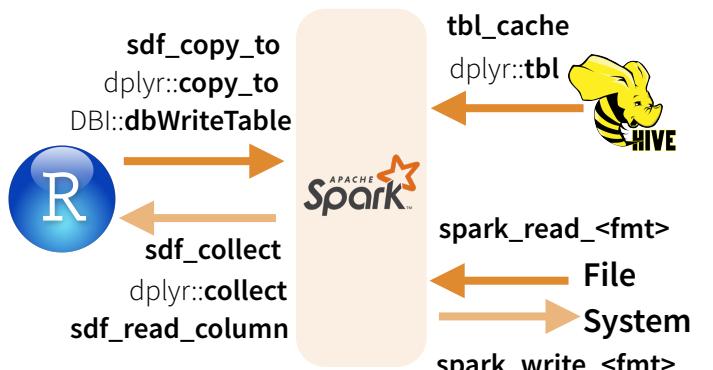
Arguments that apply to all functions: `x, path`

**CSV** `spark_read_csv(header = TRUE, delimiter = "", quote = "\"", escape = "\\\", charset = "UTF-8", null_value = NULL)`

**JSON** `spark_read_json(mode = NULL)`

**PARQUET** `spark_read_parquet(mode = NULL)`

# Reading & Writing from Apache Spark



# Extensions

Create an R package that calls the full Spark API & provide interfaces to Spark packages.

## CORE TYPES

`spark_connection()` Connection between R and the Spark shell process

`spark_jobj()` Instance of a remote Spark object

`spark_dataframe()` Instance of a remote Spark DataFrame object

## CALL SPARK FROM R

`invoke()` Call a method on a Java object

`invoke_new()` Create a new object by invoking a constructor

`invoke_static()` Call a static method on an object

## MACHINE LEARNING EXTENSIONS

`ml_create_dummy_variables()` `ml_options()`

`ml_prepare_dataframe()` `ml_model()`

`ml_prepare_response_features_intercept()`

# Model (MLlib)

`ml_decision_tree(my_table, response = "Species", features = c("Petal_Length", "Petal_Width"))`

`ml_als_factorization(x, user.column = "user", rating.column = "rating", item.column = "item", rank = 10L, regularization.parameter = 0.1, iter.max = 10L, ml.options = ml_options())`

`ml_decision_tree(x, response, features, max.bins = 32L, max.depth = 5L, type = c("auto", "regression", "classification"), ml.options = ml_options())` Same options for: `ml_gradient_boosted_trees`

`ml_generalized_linear_regression(x, response, features, intercept = TRUE, family = gaussian(link = "identity"), iter.max = 100L, ml.options = ml_options())`

`ml_kmeans(x, centers, iter.max = 100, features = dplyr::tbl_vars(x), compute.cost = TRUE, tolerance = 1e-04, ml.options = ml_options())`

`ml_lda(x, features = dplyr::tbl_vars(x), k = length(features), alpha = (50/k) + 1, beta = 0.1 + 1, ml.options = ml_options())`

`ml_linear_regression(x, response, features, intercept = TRUE, alpha = 0, lambda = 0, iter.max = 100L, ml.options = ml_options())`  
Same options for: `ml_logistic_regression`

`ml_multilayer_perceptron(x, response, features, layers, iter.max = 100, seed = sample(.Machine$integer.max, 1), ml.options = ml_options())`

`ml_naive_bayes(x, response, features, lambda = 0, ml.options = ml_options())`

`ml_one_vs_rest(x, classifier, response, features, ml.options = ml_options())`

`ml_pca(x, features = dplyr::tbl_vars(x), ml.options = ml_options())`

`ml_random_forest(x, response, features, max.bins = 32L, max.depth = 5L, num.trees = 20L, type = c("auto", "regression", "classification"), ml.options = ml_options())`

`ml_survival_regression(x, response, features, intercept = TRUE, censor = "censor", iter.max = 100L, ml.options = ml_options())`

`ml_binary_classification_eval(predicted_tbl_spark, label, score, metric = "areaUnderROC")`

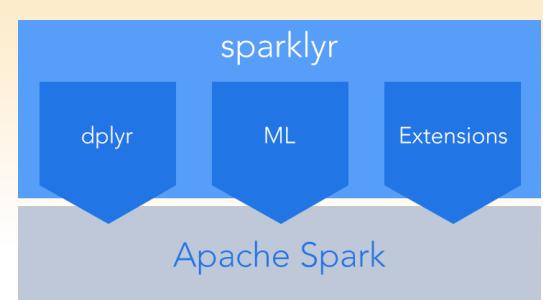
`ml_classification_eval(predicted_tbl_spark, label, predicted_lbl, metric = "f1")`

`ml_tree_feature_importance(sc, model)`

## sparklyr

is an R interface  
for

Apache  
Spark™



# R Syntax Comparison :: CHEAT SHEET

## Dollar sign syntax

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

### SUMMARY STATISTICS:

one continuous variable:

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

one categorical variable:

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

two categorical variables:

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

one continuous, one categorical:

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

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

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

### PLOTTING:

one continuous variable:

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

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

one categorical variable:

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

two continuous variables:

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

two categorical variables:

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

one continuous, one categorical:

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

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

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

```
boxplot(mtcars$disp[mtcars$cyl==4])
boxplot(mtcars$disp[mtcars$cyl==6])
boxplot(mtcars$disp[mtcars$cyl==8])
```

### WRANGLING:

subsetting:

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

making a new variable:

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

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

## Formula syntax

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

### SUMMARY STATISTICS:

one continuous variable:

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

one categorical variable:

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

two categorical variables:

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

one continuous, one categorical:

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

tilde

### PLOTTING:

one continuous variable:

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

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

one categorical variable:

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

two continuous variables:

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

two categorical variables:

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

one continuous, one categorical:

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

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

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

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

## Tidyverse syntax

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

### SUMMARY STATISTICS:

one continuous variable:

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

one categorical variable:

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

two categorical variables:

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

one continuous, one categorical:

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

the pipe

### PLOTTING:

one continuous variable:

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

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

one categorical variable:

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

two continuous variables:

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

two categorical variables:

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

one continuous, one categorical:

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

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

### WRANGLING:

subsetting:

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

making a new variable:

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

# R Syntax Comparison :: CHEAT SHEET

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

The three most prevalent R syntaxes are:

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

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

## Internet research tip:

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

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



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

## Even more ways to say the same thing

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

### quickplot

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

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

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

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

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

### ggplot

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

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

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

plus adds layers

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

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

### ggformula

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

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

### formulas in base plots

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

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

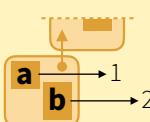


# Tidy evaluation with rlang :: CHEAT SHEET

## Vocabulary

**Tidy Evaluation (Tidy Eval)** is not a package, but a framework for doing non-standard evaluation (i.e. delayed evaluation) that makes it easier to program with tidyverse functions.

**pi**



**Symbol** - a name that represents a value or object stored in R. `is_symbol(expr(pi))`

**Environment** - a list-like object that binds symbols (names) to objects stored in memory. Each env contains a link to a second, **parent** env, which creates a chain, or search path, of environments. `is_environment(current_env())`

`rlang::caller_env(n = 1)` Returns calling env of the function it is in.

`rlang::child_env(.parent, ...)` Creates new env as child of .parent. Also **env**.

`rlang::current_env()` Returns execution env of the function it is in.

**1**

**abs ( 1 )**

**pi** — code  
3.14 — result

**Constant** - a bare value (i.e. an atomic vector of length 1). `is_bare_atomic(1)`

**Call object** - a vector of symbols/constants/calls that begins with a function name, possibly followed by arguments. `is_call(expr(abs(1)))`

**Code** - a sequence of symbols/constants/calls that will return a result if evaluated. Code can be:

1. Evaluated immediately (**Standard Eval**)
  2. Quoted to use later (**Non-Standard Eval**)
- `is_expression(expr(pi))`

**Expression** - an object that stores quoted code without evaluating it. `is_expression(expr(a + b))`

**Quosure**- an object that stores both quoted code (without evaluating it) and the code's environment. `is_quosure(quo(a + b))`

`a b` **rlang::quo\_get\_env(quo)** Return the environment of a quosure.

`a b` **rlang::quo\_set\_env(quo, expr)** Set the environment of a quosure.

`a + b` **rlang::quo\_get\_expr(quo)** Return the expression of a quosure.

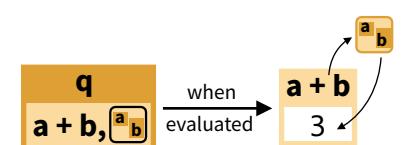
**Expression Vector** - a list of pieces of quoted code created by base R's `expression` and `parse` functions. Not to be confused with **expression**.

R Studio

## Quoting Code

Quote code in one of two ways (if in doubt use a quosure):

### QUOSURES



**Quosure**- An expression that has been saved with an environment (aka a closure).

A quosure can be evaluated later in the stored environment to return a predictable result.

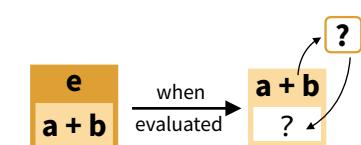
`rlang::quo(expr)` Quote contents as a quosure. Also **quos** to quote multiple expressions. `a <- 1; b <- 2; q <- quo(a + b); qs <- quos(a, b)`

`rlang::enquo(arg)` Call from within a function to quote what the user passed to an argument as a quosure. Also **enquos** for multiple args.  
`quote_this <- function(x) enquo(x)`  
`quote_these <- function(...) enquos(...)`

`rlang::new_quosure(expr, env = caller_env())` Build a quosure from a quoted expression and an environment.  
`new_quosure(expr(a + b), current_env())`



### EXPRESSION



**Quoted Expression** - An expression that has been saved by itself.

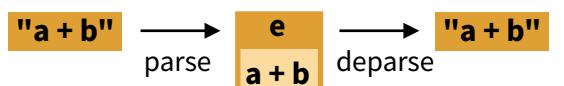
A quoted expression can be evaluated later to return a result that will depend on the environment it is evaluated in

`rlang::expr(expr)` Quote contents. Also **exprs** to quote multiple expressions. `a <- 1; b <- 2; e <- expr(a + b); es <- exprs(a, b, a + b)`

`rlang::enexpr(arg)` Call from within a function to quote what the user passed to an argument. Also **enexprs** to quote multiple arguments.  
`quote_that <- function(x) enexpr(x)`  
`quote_those <- function(...) enexprs(...)`

`rlang::ensym(x)` Call from within a function to quote what the user passed to an argument as a symbol, accepts strings. Also **ensyms**.  
`quote_name <- function(name) ensym(name)`  
`quote_names <- function(...) ensyms(...)`

## Parsing and Deparsing



**Parse** - Convert a string to a saved expression.

• • •

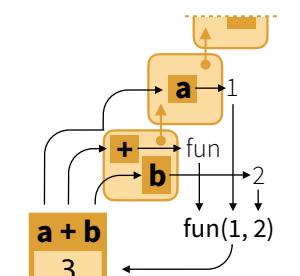
`rlang::parse_expr(x)` Convert a string to an expression. Also **parse\_exprs**, **sym**, **parse\_quo**, **parse\_quos**. `e <- parse_expr("a+b")`

**Deparse** - Convert a saved expression to a string.

• • •

`rlang::expr_text(expr, width = 60L, nlines = Inf)` Convert expr to a string. Also **quo\_name**. `expr_text(e)`

## Evaluation



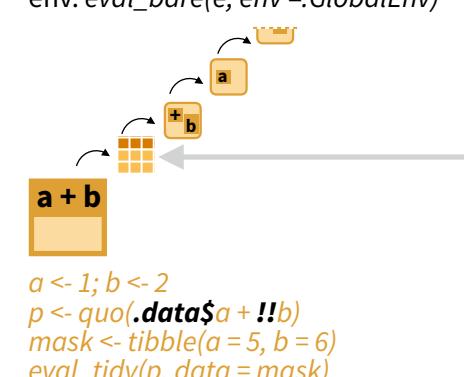
To evaluate an expression, R :

1. Looks up the symbols in the expression in the active environment (or a supplied one), followed by the environment's parents
2. Executes the calls in the expression

**The result of an expression depends on which environment it is evaluated in.**

### QUOTED EXPRESSION

`rlang::eval_bare(expr, env = parent.frame())` Evaluate expr in env. `eval_bare(e, env = GlobalEnv)`



### QUOSURES (and quoted exprs)

`rlang::eval_tidy(expr, data = NULL, env = caller_env())` Evaluate expr in env, using data as a **data mask**. Will evaluate quosures in their stored environment. `eval_tidy(q)`

**Data Mask** - If data is non-NULL, `eval_tidy` inserts data into the search path before env, matching symbols to names in data.

Use the pronoun `.data$` to force a symbol to be matched in data, and `!!` (see back) to force a symbol to be matched in the environments.

## Building Calls

`rlang::call2(fn, ..., .ns = NULL)` Create a call from a function and a list of args. Use **exec** to create and then evaluate the call. (See back page for !!!) `args <- list(x = 4, base = 2)`

`log (x = 4, base = 2)`

**2**

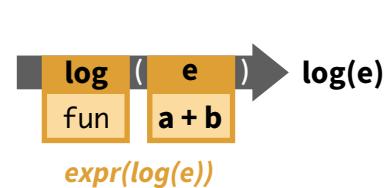
```
call2("log", x = 4, base = 2)
call2("log", !!!args)
exec("log", x = 4, base = 2)
exec("log", !!!args)
```



# Quasiquotation (!!, !!!, :=)

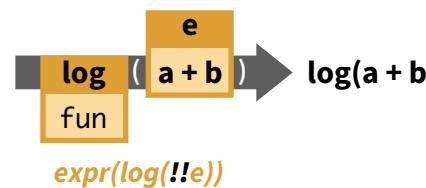
## QUOTATION

Storing an expression without evaluating it.  
e <- expr(a + b)



## QUASIUOTATION

Quoting some parts of an expression while evaluating and then inserting the results of others (**unquoting** others).  
e <- expr(a + b)

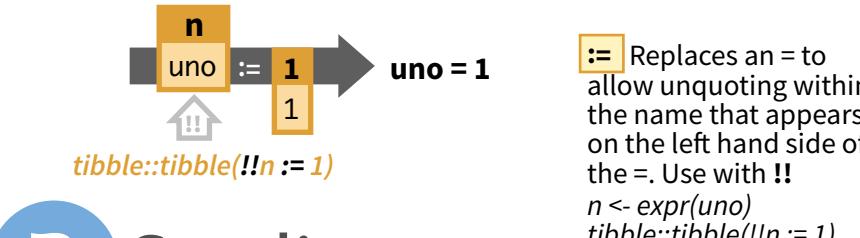
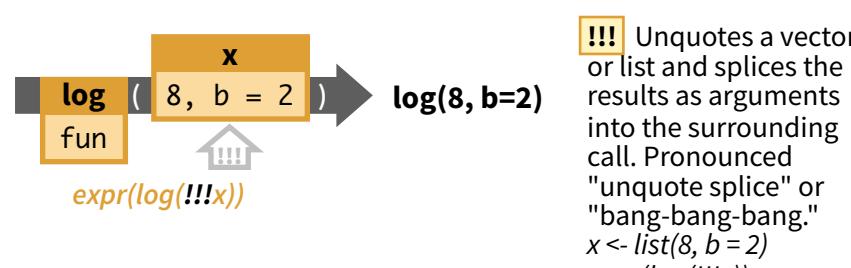
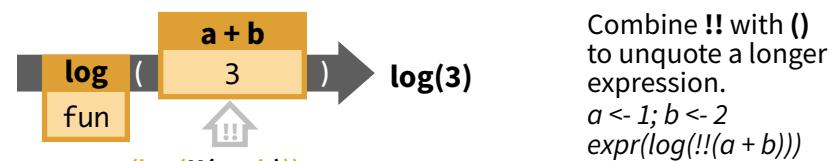
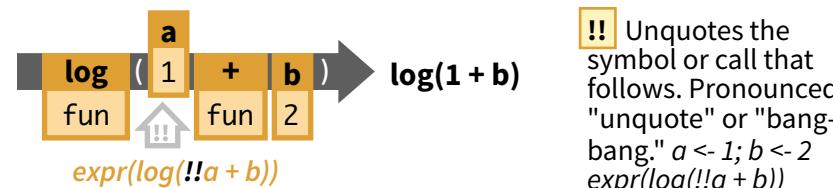


rlang provides !!, !!!, and := for doing quasiquotation.

!!, !!!, and := are not functions but syntax (symbols recognized by the functions they are passed to). Compare this to how

- . is used by magrittr::%>%()
- . is used by stats::lm()
- .x is used by purrr::map(), and so on.

!!, !!!, and := are only recognized by some rlang functions and functions that use those functions (such as tidyverse functions).



# Programming Recipes

**Quoting function**- A function that quotes any of its arguments internally for delayed evaluation in a chosen environment. You must take **special steps to program safely** with a quoting function.

**How to spot a quoting function?**  
A function quotes an argument if the argument returns an error when run on its own.

Many tidyverse functions are quoting functions: e.g. **filter**, **select**, **mutate**, **summarise**, etc.

```
dplyr::filter(cars, speed == 25)
      speed dist
      1     25    85
```

```
speed == 25
      Error!
```

## PROGRAM WITH A QUOTING FUNCTION

```
data_mean <- function(data, var) {
  require(dplyr)
  var <- rlang::enquo(var) 1
  data %>%
    summarise(mean = mean (!!var)) 2
}
```

1. Capture user argument that will be quoted with rlang::enquo.
2. Unquote the user argument into the quoting function with !!.

## PASS MULTIPLE ARGUMENTS TO A QUOTING FUNCTION

```
group_mean <- function(data, var, ...) {
  require(dplyr)
  var <- rlang::enquo(var)
  group_vars <- rlang::enquos(...) 1
  data %>%
    group_by (!!group_vars) %>%
    summarise(mean = mean (!!var)) 2
}
```

1. Capture user arguments that will be quoted with rlang::enquos.
2. Unquote splice the user arguments into the quoting function with !!!.

## MODIFY USER ARGUMENTS

```
my_do <- function(f, v, df) {
  f <- rlang::enquo(f)
  v <- rlang::enquo(v)
  todo <- rlang::quo (!!f)(!!v)) 2
  rlang::eval_tidy(todo, df) 3
}
```

1. Capture user arguments with rlang::enquo.
2. **Unquote** user arguments into a new expression or quo to use
3. **Evaluate** the new expression/ quo instead of the original argument

## APPLY AN ARGUMENT TO A DATA FRAME

```
subset2 <- function(df, rows) {
  rows <- rlang::enquo(rows) 1
  vals <- rlang::eval_tidy(rows, data = df)
  df[vals, , drop = FALSE] 2
}
```

1. Capture user argument with rlang::enquo.
2. Evaluate the argument with rlang::eval\_tidy. Pass the data frame to **data** to use as a data mask.
3. **Suggest** in your documentation that your users use the **.data** and **.env** pronouns.

## WRITE A FUNCTION THAT RECOGNIZES QUASIUOTATION (!!, !!!, :=)

1. Capture the quasiquotation-aware argument with rlang::enquo.
2. Evaluate the arg with rlang::eval\_tidy.

```
add1 <- function(x) {
  q <- rlang::enquo(x)
  rlang::eval_tidy(q) + 1
}
```

1  
2

## PASS TO ARGUMENT NAMES OF A QUOTING FUNCTION

```
named_mean <- function(data, var) {
  require(dplyr)
  var <- rlang::ensym(var)
  data %>%
    summarise (!!name := mean (!!var)) 2
}
```

1. Capture user argument that will be quoted with rlang::ensym.
2. Unquote the name into the quoting function with !! and :=.

## PASS CRAN CHECK

```
#' @importFrom rlang .data
  mutate_y <- function(df) {
    dplyr::mutate(df, y = .data$a + 1)
  }
```

1  
2

Quoted arguments in tidyverse functions can trigger an **R CMD check** NOTE about undefined global variables. To avoid this:

1. Import rlang::.data to your package, perhaps with the roxygen2 tag **@importFrom rlang .data**
2. Use the **.data\$** pronoun in front of variable names in tidyverse functions