

A blue-tinted photograph of the Austin, Texas skyline at dusk. In the foreground, the Congress Avenue Bridge spans the Colorado River, with many small birds flying around its towers. The city skyline features several prominent skyscrapers, including the Frost Bank Tower and the Driskill Hotel. The overall atmosphere is hazy and architectural.

rstudio::conf

2019 / CHEATSHEETS

from  R Studio



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+↑
↑/↓	↑/↓

Windows/Linux	Mac
Home	Cmd+←

Windows/Linux	Mac
End	Cmd+→

Windows/Linux	Mac
Ctrl+Shift+H	Ctrl+Shift+H

Windows/Linux	Mac
Esc	Esc

Windows/Linux	Mac
Ctrl+L	Ctrl+L

Windows/Linux	Mac
Ctrl+Q	Cmd+Q

Windows/Linux	Mac
Ctrl+Shift+F10	Cmd+Shift+F10

Windows/Linux	Mac
Ctrl+Enter	Cmd+Enter

Windows/Linux	Mac
Alt+Enter	Option+Enter

Windows/Linux	Mac
Ctrl+Alt+E	Cmd+Option+E

Windows/Linux	Mac
Ctrl+Alt+F	Cmd+Option+F

Windows/Linux	Mac
Ctrl+Alt+G	Cmd+Option+G

Windows/Linux	Mac
Ctrl+Shift+S	Cmd+Shift+S

Windows/Linux	Mac
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

5 WRITE CODE

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

Ctrl+Opt+Backspace

Option+Delete

Ctrl+K

Option+Backspace

Tab (at start of line)

Shift+Tab

Ctrl+U

Ctrl+K

Ctrl+Y

Alt+-

Option+-

Ctrl+Shift+M

F1

F2

Cmd+Shift+N

Ctrl+Alt+Shift+N

Ctrl+O

Cmd+O

Cmd+S

Cmd+W

Cmd+Option+W

Cmd+Shift+W

Ctrl+Shift+W

Cmd+Option+X

Cmd+Option+V

Ctrl+I

Cmd+Shift+C

Ctrl+Shift+/

Cmd+Shift+A

Ctrl+Shift+E

Ctrl+Shift+Alt+P

Ctrl+T

Option+↑/↓

Shift+Alt+↑/↓

Ctrl+Option+↑/↓

Ctrl+Alt+Up

Ctrl+Option+Down

Ctrl+Option+Shift+Up

Ctrl+Alt+Shift+Down

Ctrl+Opt+Shift+Down

Ctrl+F

Cmd+F

Cmd+E

Cmd+Shift+J

6 DEBUG CODE

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+←

Cmd+Shift+→

Shift+PageUp/Down

Shift+Alt+↑/↓

Ctrl+Opt+Backspace

Ctrl+Opt+Delete

Ctrl+K

Option+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

Cmd+O

Cmd+S

Cmd+W

Cmd+Option+W

Cmd+Shift+W

Ctrl+Shift+W

Cmd+Option+X

Cmd+Option+V

Ctrl+I

Cmd+Shift+C

Ctrl+Shift+/

Cmd+Shift+A

Ctrl+Shift+E

Ctrl+Shift+Alt+P

Ctrl+T

Option+↑/↓

Shift+Alt+↑/↓

Ctrl+Option+↑/↓

Ctrl+Alt+Up

Ctrl+Option+Down

Ctrl+Opt+Shift+Down

Ctrl+F

Cmd+F

Cmd+E

Cmd+Shift+J

7 MAKE PACKAGES

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+←

Cmd+Shift+→

Shift+PageUp/Down

Shift+Alt+↑/↓

Ctrl+Opt+Backspace

Ctrl+Opt+Delete

Ctrl+K

Option+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

Cmd+O

Cmd+S

Cmd+W

Cmd+Option+W

Cmd+Shift+W

Ctrl+Shift+W

Cmd+Option+X

Cmd+Option+V

Ctrl+I

Cmd+Shift+C

Ctrl+Shift+/

Cmd+Shift+A

Ctrl+Shift+E

Ctrl+Shift+Alt+P

Ctrl+T

Option+↑/↓

Shift+Alt+↑/↓

Ctrl+Option+↑/↓

Ctrl+Alt+Up

Ctrl+Option+Down

Ctrl+Opt+Shift+Down

Ctrl+F

Cmd+F

Cmd+E

Cmd+

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()
server <- function(input, output){}
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/



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

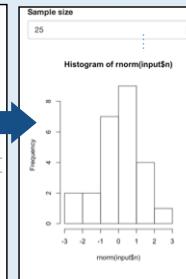
Tell server how to render outputs with R in the server function. To do this:

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

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

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

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



`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
global.R
DESCRIPTION
README
<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



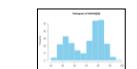
`DT::renderDataTable(expr, options, callback, escape, env, quoted)`

works with

`dataTableOutput(outputId, icon, ...)`

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

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



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

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

foo
bar
text
checkbox
radio
select
slider
submit
text

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

`verbatimTextOutput(outputId)`

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

`tableOutput(outputId)`

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

`textOutput(outputId, container, inline)`

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

`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**.

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

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

`checkboxGroupInput(inputId, label, choices, selected, inline)`

`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())`

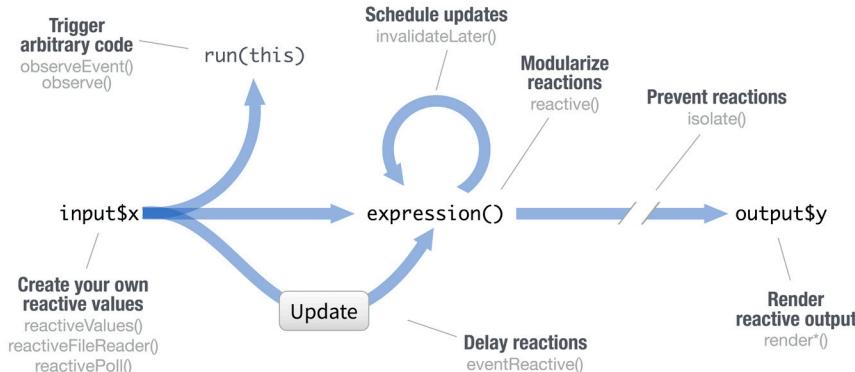
`sliderInput(inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post)`

`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
}

# *Input() functions
# (see front page)
reactiveValues(...)

# Each input function
# creates a reactive value
# stored as input$<inputId>
# 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.quoted, handler.env, event.quoted, handler.env, 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$a, {input$z})
  output$b <-
  renderText({
    re()
  })
}

shinyApp(ui, server)
```

eventReactive(eventExpr, valueExpr, 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>
```

Returns HTML



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\$h6	tags\$nav	tags\$span
tags\$abbr	tags\$head	tags\$noscript	tags\$strong
tags\$address	tags\$header	tags\$object	tags\$style
tags\$area	tags\$group	tags\$ol	tags\$sub
tags\$article	tags\$details	tags\$hr	tags\$optgroup
tags\$aside	tags\$dfn	tags\$option	tags\$summary
tags\$audio	tags\$div	tags\$output	
tags\$b	tags\$dl	tags\$iframe	
tags\$base	tags\$dt	tags\$img	
tags\$bdi	tags\$sem	tags\$input	
tags\$blockquote	tags\$source	tags\$progress	
tags\$body	tags\$events	tags\$tbody	
tags\$br	tags\$fields	tags\$td	
tags\$button	tags\$events	tags\$th	
tags\$canvas	tags\$figure	tags\$thead	
tags\$caption	tags\$foot	tags\$time	
tags\$cite	tags\$li	tags\$title	
tags\$code	tags\$form	tags\$track	
tags\$h1	tags\$mark	tags\$section	
tags\$h2	tags\$map	tags\$small	
tags\$col	tags\$menu	tags\$select	
tags\$colgroup	tags\$meta	tags\$var	
tags\$h3	tags\$script	tags\$video	
tags\$command	tags\$h5	tags\$source	

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

```
ui <- fluidPage(
  h1("Header 1"),
  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())
)
```

↓

```
absolutePanel()
conditionalPanel()
fixedPanel()
headerPanel()
inputPanel()
mainPanel()
navlistPanel()
sidebarPanel()
tabPanel()
tabletPanel()
titlePanel()
wellPanel()
```



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"))
```



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

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 C

```
```{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)

**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)

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

## .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.)

```

params:
 n: 100
 d: !r Sys.Date()

```

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

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     | 4.00 |
| 3     | 7.00 |
| 4     | 7.00 |
| 5     | 8.00 |

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/superscript12
~~strikethrough~~
escaped: `_\`_
endash: --, emdash: ---
equation: $A = \rho * r^2$
```

```
equation block:
```

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

```
E = mc2
```

```
> block quote
```

```
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}
HTML ignored in pdfs
```

```
http://www.rstudio.com
link
Jump to Header 1
image:
```

## Header1

## Header 2

## Header 3

### Header 4

#### Header 5

#### Header 6

HTML ignored in pdfs

http://www.rstudio.com

link

Jump to Header 1

image:



Caption

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

item 2

Continued (indent 4 spaces)

1. ordered list
  1. item 2
    - Continued (indent 4 spaces)

item 2

Continued (indent 4 spaces)

item 2

i. sub-item 1
 

- A. sub-sub-item 1

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.

1. Here is the footnote.  $\leftrightarrow$

# 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
```

Indent 2 spaces  
Indent 4 spaces

### 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
```

Tabset

Tab 1 Tab 2

text 1

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.Rmd` (contents of the template)

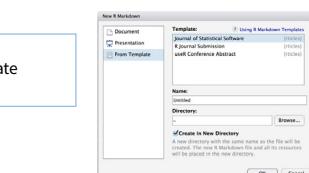
any supporting files

3. Install the package

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

name: My Template

—



### sub-option

### description

| sub-option            | description                                                                                                          | html | pdf | word | odt | rft | md | github | 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                                                                                            |      |     |      |     |     |    |        |          |       |        |
| css                   | CSS file to use to style document                                                                                    | X    |     |      |     |     |    |        |          |       |        |
| dev                   | Graphics device to use for figure output (e.g. "png")                                                                | 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   |     |    |        |          |       |        |
| fig_height, fig_width | Default figure height and width (in inches) for document                                                             | X    | X   | X    | X   | X   | X  | X      | X        | X     |        |
| highlight             | Syntax highlighting: "tango", "pygments", "kate", "zenburn", "textmate"                                              | X    | X   | X    |     |     |    |        |          |       |        |
| includes              | File of content to place in document ( <code>in_header</code> , <code>before_body</code> , <code>after_body</code> ) | X    | X   | X    | X   | X   | X  | X      | X        | X     |        |
| incremental           | Should bullets appear one at a time (on presenter mouse clicks)?                                                     |      |     |      |     |     |    |        |          | X     | X      |
| keep_md               | Save a copy of .md file that contains knitr output                                                                   | 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      |
| mathjax               | Set to local or a URL to use a local/URL version of MathJax to render equations                                      | 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      |
| template              | Pandoc template to use when rendering file <code>quarterly_report.html</code> .                                      | X    | X   | X    |     |     |    |        |          |       | X      |
| theme                 | Bootswatch or Beamer theme to use for page                                                                           | X    |     |      |     |     |    |        |          |       |        |
| toc                   | Add a table of contents at start of document                                                                         | 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`

`eruptions waiting`

|       |    |
|-------|----|
| 3.60  | 79 |
| 1.80  | 54 |
| 3.33  | 74 |
| 2.28  | 62 |
| 2.283 | 62 |

Table with `xtable`

`eruptions waiting`

|       |      |       |
|-------|------|-------|
| 1     | 3.60 | 79.00 |
| 2     | 1.80 | 54.00 |
| 3     | 3.33 | 74.00 |
| 4     | 2.28 | 62.00 |
| 2.283 | 62   |       |

Table with `stargazer`

`eruptions waiting`

|       |      |       |
|-------|------|-------|
| 1     | 3.60 | 79.00 |
| 2     | 1.80 | 54.00 |
| 3     | 3.33 | 74.00 |
| 4     | 2.28 | 62.00 |
| 2.283 | 62   |       |

Table with `knitr`

`eruptions waiting`

|       |      |       |
|-------|------|-------|
| 1     | 3.60 | 79.00 |
| 2     | 1.80 | 54.00 |
| 3     | 3.33 | 74.00 |
| 4     | 2.28 | 62.00 |
| 2.283 | 62   |       |

Table with `grid`

`eruptions waiting`

|       |      |       |
|-------|------|-------|
| 1     | 3.60 | 79.00 |
| 2     | 1.80 | 54.00 |
| 3     | 3.33 | 74.00 |
| 4     | 2.28 | 62.00 |
| 2.283 | 62   |       |

Table with `grid`

`eruptions waiting`

|       |      |       |
|-------|------|-------|
| 1     | 3.60 | 79.00 |
| 2     | 1.80 | 54.00 |
| 3     | 3.33 | 74.00 |
| 4     | 2.28 | 62.00 |
| 2.283 | 62   |       |

Table with `grid`

`eruptions waiting`

|       |      |       |
|-------|------|-------|
| 1     | 3.60 | 79.00 |
| 2     | 1.80 | 54.00 |
| 3     | 3.33 | 74.00 |
| 4     | 2.28 | 62.00 |
| 2.283 | 62   |       |

Table with `grid`

`eruptions waiting`

|       |      |       |
|-------|------|-------|
| 1     | 3.60 | 79.00 |
| 2     | 1.80 | 54.00 |
| 3     | 3.33 | 74.00 |
| 4     | 2.28 | 62.00 |
| 2.283 | 62   |       |

Table with `grid`

`eruptions waiting`

|       |      |       |
|-------|------|-------|
| 1     | 3.60 | 79.00 |
| 2     | 1.80 | 54.00 |
| 3     | 3.33 | 74.00 |
| 4     | 2.28 | 62.00 |
| 2.283 | 62   |       |

Table with `grid`

`eruptions waiting`

|  |  |  |
| --- | --- | --- |
| 1 | 3.60 | 79.00 |



<tbl\_r cells="3" ix="3" maxcspan="1" maxrspan="



# 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

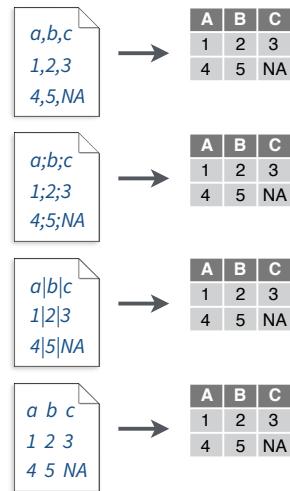
```
write_lines(x, path, na = "NA", append = FALSE)
```

### Object to RDS file

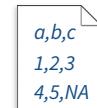
```
write_rds(x, path, compress = c("none", "gz",
 "bz2", "xz"), ...)
```

### Tab delimited files

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



## USEFUL ARGUMENTS



### Example file

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



### No header

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



### Provide header

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

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

### Skip lines

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

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

### Read in a subset

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

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

### Missing Values

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

## Read Non-Tabular Data

### Read a file into a single string

```
read_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

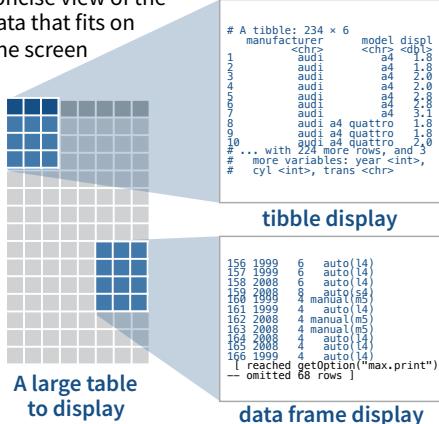
The **tibble** package provides a new S3 class for storing tabular data, the tibble. Tibbles inherit the data frame class, but improve three behaviors:



- 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



- 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

**tibble(...)**

Construct by columns.

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

Both make this tibble

**tribble(...)**

Construct by rows.

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

| 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

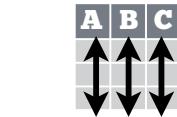
**is\_tibble(x)** Test whether x is a tibble.



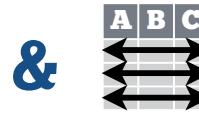
# Tidy Data with tidyverse

**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**

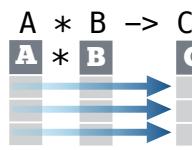


Each **observation**, or **case**, is in its own **row**

Tidy data:



Makes variables easy to access as vectors



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

**table2**

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

key value

**spread(table2, type, count)**

## 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  |

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

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

**separate\_rows(data, ..., sep = "[^[:alnum:]].+", convert = FALSE)**

Separate each cell in a column to make several rows. Also **separate\_rows\_()**.

**table3**

| country | year | rate     |
|---------|------|----------|
| A       | 1999 | 0.7K/19M |
| A       | 2000 | 2K/20M   |
| B       | 1999 | 37K/172M |
| B       | 2000 | 80K/174M |
| C       | 1999 | 212K/1T  |
| B       | 2000 | 80K      |
| B       | 2000 | 174M     |
| C       | 1999 | 212K     |
| C       | 1999 | 1T       |
| C       | 2000 | 213K     |
| C       | 2000 | 1T       |

| country | year | rate |
|---------|------|------|
| A       | 1999 | 0.7K |
| A       | 1999 | 19M  |
| A       | 2000 | 2K   |
| B       | 1999 | 37K  |
| B       | 2000 | 20M  |
| B       | 2000 | 174M |
| C       | 1999 | 212K |
| C       | 1999 | 1T   |
| C       | 2000 | 213K |
| C       | 2000 | 1T   |

**separate\_rows(table3, rate)**

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

Collapse cells across several columns to make a single column.

**table5**

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

| country | year |
|---------|------|
| Afghan  | 1999 |
| Afghan  | 2000 |
| Brazil  | 1999 |
| Brazil  | 2000 |
| China   | 1999 |
| China   | 2000 |

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

## Expand Tables - quickly create tables with combinations of values

**complete(data, ..., fill = list())**

Adds to the data missing combinations of the values of the variables listed in ...

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

**expand(data, ...)**

Create new tibble with all possible combinations of the values of the variables listed in ...

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



# 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**



`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).

### summary function



`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)`

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

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

See `?base::logic` and `?Comparison` for help.

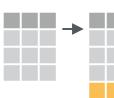
### ARRANGE CASES



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

`arrange(mtcars, mpg)`  
`arrange(mtcars, desc(mpg))`

### ADD CASES



`add_row(.data, ..., .before = NULL, .after = NULL)`  
Add one or more rows to a table.

`add_row(faithful, eruptions = 1, waiting = 1)`

## Manipulate Variables

### EXTRACT VARIABLES

Column functions return a set of columns as a new 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"))`

|                               |                                                                   |
|-------------------------------|-------------------------------------------------------------------|
| <code>contains(match)</code>  | <code>num_range(prefix, range)</code> ; e.g. <code>mpg:cyl</code> |
| <code>ends_with(match)</code> | -, e.g. <code>-Species</code>                                     |
| <code>matches(match)</code>   | <code>one_of(...)</code>                                          |
|                               | <code>starts_with(match)</code>                                   |

### 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).

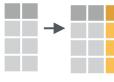
### vectorized function



`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<br>a t 1<br>b u 2<br>c v 3 | A B D<br>a t 3<br>b u 2<br>d w 1 | A B C A B D<br>a t 1 a t 3<br>b u 2 b u 2<br>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.

| x                                          | y                                          | =                                                                                                             |
|--------------------------------------------|--------------------------------------------|---------------------------------------------------------------------------------------------------------------|
| A B C D<br>a t 1 3<br>b u 2 2<br>c v 3 N A | A B C D<br>a t 1 3<br>b u 2 2<br>d w N A 1 | left_join(x, y, by = NULL,<br>copy = FALSE, suffix = c(".x", ".y"), ...)<br>Join matching values from y to x. |

| x                                          | y                                          | =                                                                                                              |
|--------------------------------------------|--------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| A B C D<br>a t 1 3<br>b u 2 2<br>d w N A 1 | A B C D<br>a t 1 3<br>b u 2 2<br>d w N A 1 | right_join(x, y, by = NULL, copy =<br>FALSE, suffix = c(".x", ".y"), ...)<br>Join matching values from x to y. |

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

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

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

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

| x                                                        | y                                | =                                                                                                                                                                        |
|----------------------------------------------------------|----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| A 1 B 1 C A 2 B 2<br>a t 1 d w<br>b u 2 b u<br>c v 3 a t | A B C<br>a t 1<br>b u 2<br>c v 3 | Use <b>suffix</b> to specify the suffix to give to unmatched columns that have the same name in both tables.<br>left_join(x, y, by = c("C" = "D"), suffix = c("1", "2")) |

## COMBINE CASES

| x                                | y                       | =                                         |
|----------------------------------|-------------------------|-------------------------------------------|
| A B C<br>a t 1<br>b u 2<br>c v 3 | A B C<br>C V 3<br>D W 4 | A B C<br>a t 1<br>b u 2<br>c v 3<br>D W 4 |

Use **bind\_rows()** to paste tables below each other as they are.

| x                                | y                       | =                                                                                                                                                                          |
|----------------------------------|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| A B C<br>a t 1<br>b u 2<br>c v 3 | A B C<br>C V 3<br>D W 4 | bind_rows(..., .id = NULL)<br>Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured) |

| x                                | y                                | =                                                         |
|----------------------------------|----------------------------------|-----------------------------------------------------------|
| A B C<br>a t 1<br>b u 2<br>c v 3 | A B C<br>a t 1<br>b u 2<br>c v 3 | intersect(x, y, ...)<br>Rows that appear in both x and y. |

| x                                | y                                | =                                                      |
|----------------------------------|----------------------------------|--------------------------------------------------------|
| A B C<br>a t 1<br>b u 2<br>c v 3 | A B C<br>a t 1<br>b u 2<br>c v 3 | setdiff(x, y, ...)<br>Rows that appear in x but not y. |

| x                                         | y                                         | =                                                                                                           |
|-------------------------------------------|-------------------------------------------|-------------------------------------------------------------------------------------------------------------|
| A B C<br>a t 1<br>b u 2<br>c v 3<br>d w 4 | A B C<br>a t 1<br>b u 2<br>c v 3<br>d w 4 | union(x, y, ...)<br>Rows that appear in x or y.<br>(Duplicates removed). union_all()<br>retains duplicates. |

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

## EXTRACT ROWS

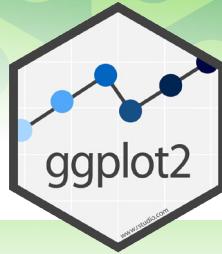
| x                                | y                                | = |
|----------------------------------|----------------------------------|---|
| A B C<br>a t 1<br>b u 2<br>c v 3 | A B D<br>a t 3<br>b u 2<br>d w 1 |   |

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

| x                                | y                                | =                                                                                                                 |
|----------------------------------|----------------------------------|-------------------------------------------------------------------------------------------------------------------|
| A B C<br>a t 1<br>b u 2<br>c v 3 | A B D<br>a t 3<br>b u 2<br>d w 1 | semi_join(x, y, by = NULL, ...)<br>Return rows of x that have a match in y.<br>USEFUL TO SEE WHAT WILL BE JOINED. |

| x                                | y                                | =                                                                                                                            |
|----------------------------------|----------------------------------|------------------------------------------------------------------------------------------------------------------------------|
| A B C<br>a t 1<br>b u 2<br>c v 3 | A B C<br>a t 1<br>b u 2<br>c v 3 | anti_join(x, y, by = NULL, ...)<br>Return rows of x that do not have a match in y.<br>USEFUL TO SEE WHAT WILL NOT BE JOINED. |

# 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>
```

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

aesthetic mappings    data    geom

**qplot(x = cyl, 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, y, xend, alpha, angle, color, curvature, linetype, size
- a + geom\_path(lineend = "butt", linejoin = "round", linemtire = 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(ymax = unemploy - 900, ymin = 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, lineneight, 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, lineneight, 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
- l + geom\_raster(aes(fill = z), hjust = 0.5, vjust = 0.5, interpolate = FALSE)** - x, y, alpha, fill
- l + geom\_tile(aes(fill = z))** - x, y, alpha, color, fill, linetype, size, width

### 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, y, max, 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, ymin, ymax, alpha, color, group, linetype, size, 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







# 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     |         |

| nested data frame |                 |  |  |  |
|-------------------|-----------------|--|--|--|
| Species           | data            |  |  |  |
| setosa            | <tibble [50x4]> |  |  |  |
| versicolor        | <tibble [50x4]> |  |  |  |
| virginica         | <tibble [50x4]> |  |  |  |

n\_iris

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

| Species | S.L | S.W | P.L | P.W |
|---------|-----|-----|-----|-----|
| setos   | 5.1 | 3.5 | 1.4 | 0.2 |
| setos   | 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()**

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

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

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

n\_iris %>% **unnest()**

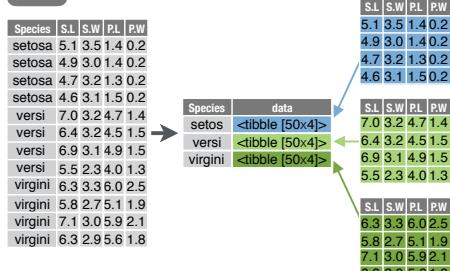
**tidyverse::unnest**(data, ..., .drop = NA, .id=NULL, .sep=NULL)

Unnests a nested data frame.

# List Column Workflow

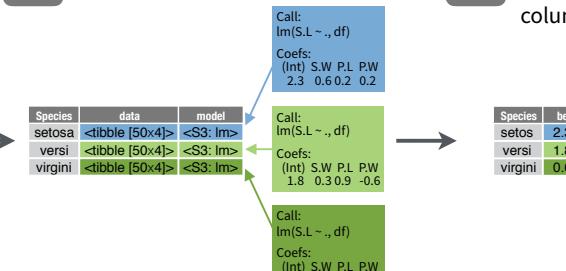
Nested data frames use a **list column**, a list that is stored as a column vector of a data frame. A typical **workflow** for list columns:

## 1 Make a list column



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

## 2 Work with list columns



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

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

```
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 **tidy**'s **nest()**

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

Makes list column when needed

```
tribble(~max, ~seq,
 3, 1:3,
 4, 1:4,
 5, 1:5)
```

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

Saves list input as list columns

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

**dplyr::mutate(.data, ...)** Also **transmute()**

Returns list col when result returns list.

```
mtcars %>% mutate(seq = map(cyl, seq))
```

**dplyr::summarise(.data, ...)**

Returns list col when result is wrapped with **list()**

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

## 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(<list>, fun, ...)**

fun{<list>, ...}

result 1  
result 2  
result 3

**map2(<list>, <list>, fun, ...)**

fun{<list>, <list>, ...}

result 1  
result 2  
result 3

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

fun{<list>, <list>, <list>, ...}

result 1  
result 2  
result 3

## 3. SIMPLIFY THE LIST COLUMN (into a regular column)

Use the purrr functions **map\_lgl()**,

**map\_int()**, **map\_dbl()**, **map\_chr()**,

as well as tidy's **unnest()** to reduce a list column into a regular column.

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

Apply .f element-wise to .x, return a logical vector

```
n_iris %>% transmute(n = map_lgl(data, is.matrix))
```

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

Apply .f element-wise to .x, return an integer vector

```
n_iris %>% transmute(n = map_int(data, nrow))
```

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

Apply .f element-wise to .x, return a double vector

```
n_iris %>% transmute(n = map_dbl(data, nrow))
```

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

Apply .f element-wise to .x, return a character vector

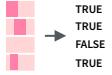
```
n_iris %>% transmute(n = map_chr(data, nrow))
```

# 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



`str_detect(string, pattern)` Detect the presence of a pattern match in a string.  
`str_detect(fruit, "a")`



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



`str_count(string, pattern)` Count the number of matches in a string.  
`str_count(fruit, "a")`



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

## Mutate Strings



`str_sub()` <- value. Replace substrings by identifying the substrings with `str_sub()` and assigning into the results.  
`str_sub(fruit, 1, 3) <- "str"`



`str_replace(string, pattern, replacement)` Replace the first matched pattern in each string. `str_replace(fruit, "a", "-")`



`str_replace_all(string, pattern, replacement)` Replace all matched patterns in each string. `str_replace_all(fruit, "a", "-")`

A STRING  
↓  
a string

a string  
↓  
A STRING

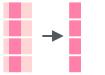
a string  
↓  
A String

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

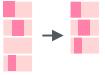
`str_to_upper(string, locale = "en")1` Convert strings to upper case.  
`str_to_upper(sentences)`

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

## Subset Strings



`str_sub(string, start = 1L, end = -1L)` Extract substrings from a character vector.  
`str_sub(fruit, 1, 3); str_sub(fruit, -2)`



`str_subset(string, pattern)` Return only the strings that contain a pattern match.  
`str_subset(fruit, "b")`



`str_extract(string, pattern)` Return the first pattern match found in each string, as a vector. Also `str_extract_all` to return every pattern match. `str_extract(fruit, "[aeiou]")`



`str_match(string, pattern)` Return the first pattern match found in each string, as a matrix with a column for each () group in pattern. Also `str_match_all`.  
`str_match(sentences, "(a|the) ([^ ]+)")`

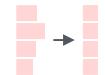
## Manage Lengths



`str_length(string)` The width of strings (i.e. number of code points, which generally equals the number of characters). `str_length(fruit)`



`str_pad(string, width, side = c("left", "right", "both"), pad = " ")` Pad strings to constant width. `str_pad(fruit, 17)`



`str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "...")` Truncate the width of strings, replacing content with ellipsis.  
`str_trunc(fruit, 3)`



`str_trim(string, side = c("both", "left", "right"))` Trim whitespace from the start and/or end of a string. `str_trim(fruit)`

## Join and Split



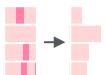
`str_c(..., sep = "", collapse = NULL)` Join multiple strings into a single string.  
`str_c(letters, LETTERS)`



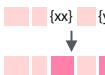
`str_c(..., sep = "", collapse = NULL)` Collapse a vector of strings into a single string.  
`str_c(letters, collapse = "")`



`str_dup(string, times)` Repeat strings times times. `str_dup(fruit, times = 2)`



`str_split_fixed(string, pattern, n)` Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match). Also `str_split` to return a list of substrings.  
`str_split_fixed(fruit, " ", n=2)`



`str_glue(..., .sep = "", .envir = parent.frame())` Create a string from strings and {expressions} to evaluate. `str_glue("Pi is {pi}")`



`str_glue_data(.x, ..., .sep = "", .envir = parent.frame(), .na = "NA")` Use a data frame, list, or environment to create a string from strings and {expressions} to evaluate.  
`str_glue_data(mtcars, "frownames(mtcars)  
has {hp} hp")`

## Order Strings



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



`str_sort(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)1` Sort a character vector.  
`str_sort(x)`

## Helpers

apple  
banana  
pear

apple  
banana  
pear

`str_conv(string, encoding)` Override the encoding of a string. `str_conv(fruit, "ISO-8859-1")`

`str_view(string, pattern, match = NA)` View HTML rendering of first regex match in each string. `str_view(fruit, "[aeiou]")`

`str_view_all(string, pattern, match = NA)` View HTML rendering of all regex matches. `str_view_all(fruit, "[aeiou]")`

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

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

# Need to Know

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

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

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

| Special Character | Represents |
|-------------------|------------|
| \\\               | \          |
| "                 | "          |
| \n                | new line   |

Run `?!!!` to see a complete list

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

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

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

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

## INTERPRETATION

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

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

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

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

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

# Regular Expressions -

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

## MATCH CHARACTERS

| string (type this)     | regexp (to mean this)                     | matches (which matches this) | example              |
|------------------------|-------------------------------------------|------------------------------|----------------------|
| a (etc.)               | a (etc.)                                  | see("a")                     | abc ABC 123 .!?( ){} |
| \.                     | .                                         | see("(.")")                  | abc ABC 123 .!?( ){} |
| \!                     | !                                         | see("(!")")                  | abc ABC 123 .!?( ){} |
| \?                     | ?                                         | see("(!?")")                 | abc ABC 123 .!?( ){} |
| \                      | \                                         | see("(\ ")")                 | abc ABC 123 .!?( ){} |
| \(                     | (                                         | see("(\\"")")                | abc ABC 123 .!?( ){} |
| \)                     | )                                         | see("(\\"")")                | abc ABC 123 .!?( ){} |
| \{                     | {                                         | see("(\{")")                 | abc ABC 123 .!?( ){} |
| \}                     | }                                         | see("(\}")")                 | abc ABC 123 .!?( ){} |
| \n                     | new line                                  | see("\n")                    | abc ABC 123 .!?( ){} |
| \t                     | tab                                       | see("\t")                    | abc ABC 123 .!?( ){} |
| \s                     | any whitespace (S for non-whitespaces)    | see("\s")                    | abc ABC 123 .!?( ){} |
| \d                     | any digit (D for non-digits)              | see("\d")                    | abc ABC 123 .!?( ){} |
| \w                     | any word character (W for non-word chars) | see("\w")                    | abc ABC 123 .!?( ){} |
| \b                     | word boundaries                           | see("\b")                    | abc ABC 123 .!?( ){} |
| [:digit:] <sup>1</sup> | digits                                    | see("[ :digit:]")            | abc ABC 123 .!?( ){} |
| [:alpha:] <sup>1</sup> | letters                                   | see("[ :alpha:]")            | abc ABC 123 .!?( ){} |
| [:lower:] <sup>1</sup> | lowercase letters                         | see("[ :lower:]")            | abc ABC 123 .!?( ){} |
| [:upper:] <sup>1</sup> | uppercase letters                         | see("[ :upper:]")            | abc ABC 123 .!?( ){} |
| [:alnum:] <sup>1</sup> | letters and numbers                       | see("[ :alnum:]")            | abc ABC 123 .!?( ){} |
| [:punct:] <sup>1</sup> | punctuation                               | see("[ :punct:]")            | abc ABC 123 .!?( ){} |
| [:graph:] <sup>1</sup> | letters, numbers, and punctuation         | see("[ :graph:]")            | abc ABC 123 .!?( ){} |
| [:space:] <sup>1</sup> | space characters (i.e. \s)                | see("[ :space:]")            | abc ABC 123 .!?( ){} |
| [:blank:] <sup>1</sup> | space and tab (but not new line)          | see("[ :blank:]")            | abc ABC 123 .!?( ){} |
| .                      | every character except a new line         | see("."")                    | abc ABC 123 .!?( ){} |

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

## ALTERNATES

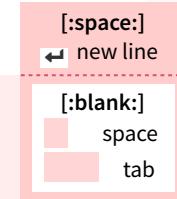
| regexp | matches      | example       |
|--------|--------------|---------------|
| ab d   | or           | alt("ab d")   |
| [abe]  | one of       | alt("[abe]")  |
| [^abe] | anything but | alt("[^abe]") |
| [a-c]  | range        | alt("[a-c]")  |

## ANCHORS

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

## LOOK AROUNDS

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



## [:punct:]

|   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|
| , | : | ; | ? | ! | \ | / | = | * | + | - | ^ |
| - | = | ' | ' | [ | ] | { | } | ( | ) | < | > |

## [:alnum:]

### [:digit:]

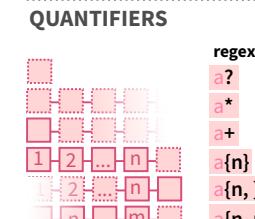
0 1 2 3 4 5 6 7 8 9

## [:alpha:]

### [:lower:]

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

## QUANTIFIERS



| quant | function(rx str_view_all(".aa.aaa", rx)) | matches         | example |
|-------|------------------------------------------|-----------------|---------|
| a?    | quant("a?")                              | zero or one     | .aa.aaa |
| a*    | quant("a*")                              | zero or more    | .aa.aaa |
| a+    | quant("a+")                              | one or more     | .aa.aaa |
| {n}   | quant("{n}")                             | exactly n       | .aa.aaa |
| {n,}  | quant("{n,}")                            | n or more       | .aa.aaa |
| {n,m} | quant("{n,m}")                           | between n and m | .aa.aaa |

## GROUPS

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

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

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

| string   | regexp | matches              | example           |
|----------|--------|----------------------|-------------------|
| (t.)(.t) | (\1)\2 | first () group, etc. | ref("(a)(b)\2\1") |

# 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)





# Math with Date-times

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

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

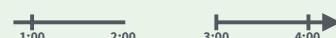
A normal day

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



The start of daylight savings (spring forward)

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



The end of daylight savings (fall back)

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



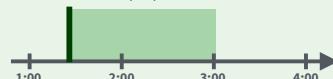
Leap years and leap seconds

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

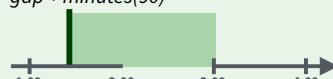


**Periods** track changes in clock times, which ignore time line irregularities.

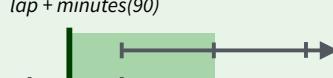
`nor + minutes(90)`



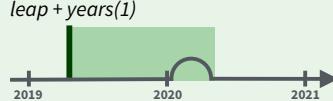
`gap + minutes(90)`



`lap + minutes(90)`



`leap + years(1)`

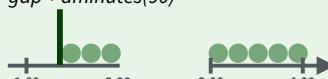


**Durations** track the passage of physical time, which deviates from clock time when irregularities occur.

`nor + dminutes(90)`



`gap + dminutes(90)`



`lap + dminutes(90)`



`leap + dyears(1)`



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

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



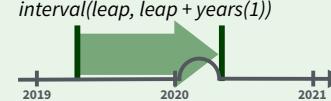
`interval(gap, gap + minutes(90))`



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



`interval(leap, leap + years(1))`



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

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

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

```
jan31 <- ymd("20180131")
jan31 + months(1)
NA
```

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

```
jan31 %m+% months(1)
"2018-02-28"
```

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

```
add_with_rollback(jan31, months(1),
roll_to_first = TRUE)
"2018-03-01"
```

## PERIODS

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

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

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

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

Number of months

Number of days

`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.

**Diftimes** are a class of durations found in base R.

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

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

`dd`

"1209600s (~2 weeks)"

Exact length in seconds

Equivalent in common units

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

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

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

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

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

`dseconds(x = 1) x seconds.`

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

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

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

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

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

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

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

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

## INTERVALS

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

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

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

## 2017-01-01 UTC--2017-11-28 UTC  
j <- d %--% ymd("2017-12-31")

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

a `%within%` b Does interval or date-time a fall within interval b? `now() %within% i`

`int_start(int)` Access/set the start date-time of an interval. Also `int_end()`. `int_start(i) <- now()`; `int_start(i)`

`int_aligns(int1, int2)` Do two intervals share a boundary? Also `int_overlaps()`. `int_aligns(i, j)`

`int_diff(times)` Make the intervals that occur between the date-times in a vector.  
`v <- c(dt, dt + 100, dt + 1000); int_diff(v)`

`int_flip(int)` Reverse the direction of an interval. Also `int_standardize()`. `int_flip(i)`

`int_length(int)` Length in seconds. `int_length(i)`

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

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

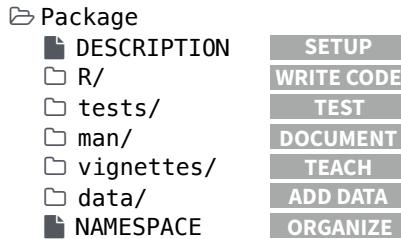
# Package Development: : **CHEAT SHEET**



## Package Structure

A package is a convention for organizing files into directories.

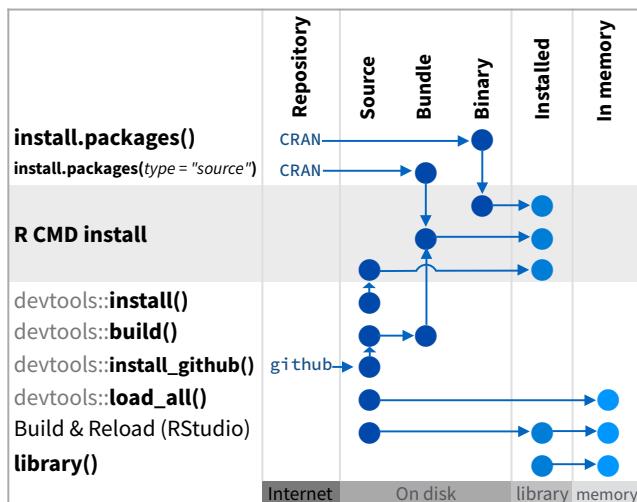
This sheet shows how to work with the 7 most common parts of an R package:



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

```
Package: mypackage
Title: Title of Package
Version: 0.1.0
Authors@R: person("Hadley", "Wickham", email =
 "hadley@r-project.org", 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 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 `src/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

| 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)}  | \tabular{lcr}{                    |
| \eqn{a + b (inline)}  | left \tab centered \tab right \cr |
|                       | cell \tab cell \tab cell \cr}     |

## 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))

### 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 | @seealso |      |
| @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   |

## 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-data/. Includes data-data/ 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

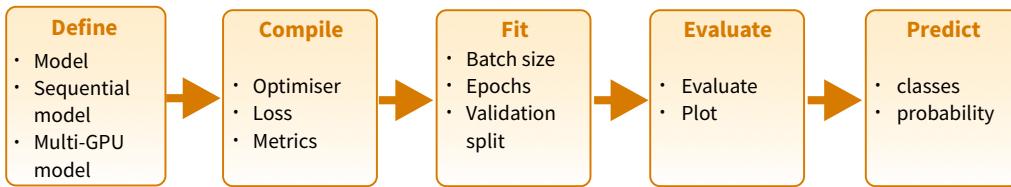
[r-pkgs.had.co.nz/release.html](http://r-pkgs.had.co.nz/release.html)

# 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

See ?install\_keras  
for GPU instructions

## 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()

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

5 0 4 1

```

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)

```

## 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 input activity

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

 `layer_flatten()` Flattens an input

# More layers

## CONVOLUTIONAL LAYERS

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

## POOLING LAYERS

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



Studio

## ACTIVATION LAYERS

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

## DROPOUT LAYERS

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

## RECURRENT LAYERS

|                                                                                    |                                                                                                    |
|------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------|
|   | <code>layer_simple_rnn()</code><br>Fully-connected RNN where the output is to be fed back to input |
|  | <code>layer_gru()</code><br>Gated recurrent unit - Cho et al                                       |

|                                                                                     |                                                                           |
|-------------------------------------------------------------------------------------|---------------------------------------------------------------------------|
|  | <code>layer_cudnn_gru()</code><br>Fast GRU implementation backed by CuDNN |
|-------------------------------------------------------------------------------------|---------------------------------------------------------------------------|

|                                                                                     |                                                                                  |
|-------------------------------------------------------------------------------------|----------------------------------------------------------------------------------|
|  | <code>layer_cudnn_lstm()</code><br>Long-Short Term Memory unit - Hochreiter 1997 |
|-------------------------------------------------------------------------------------|----------------------------------------------------------------------------------|

|                                                                                     |                                                                                                                                        |
|-------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
|  | <code>layer_locally_connected_1d()</code>                                                                                              |
|  | <code>layer_locally_connected_2d()</code><br>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>                               |                                                                      |
| <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

|                                                                                                                                                                                                            |                                                                                                                                                                                               |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>application_xception()</code>                                                                                                                                                                        | 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>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                                                                                                                                                                  |
|                                                                                                                        | <b>ImageNet</b> is a large database of images with labels, extensively used for deep learning                                                                                                 |
| <code>imagenet_preprocess_input()</code>                                                                                                                                                                   |                                                                                                                                                                                               |
| <code>imagenet_decode_predictions()</code>                                                                                                                                                                 |                                                                                                                                                                                               |
| <code>Preprocesses a tensor encoding a batch of images for ImageNet, and decodes predictions</code>                                                                                                        | Preprocesses a tensor encoding a batch of images for ImageNet, and decodes predictions                                                                                                        |
| <code>Callbacks</code>                                                                                                                                                                                     |                                                                                                                                                                                               |
| <code>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> | 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                                                                                                                                                              |

# Data Science in Spark with sparklyr :: CHEAT SHEET

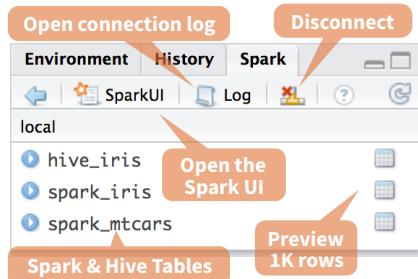


## Intro

**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.

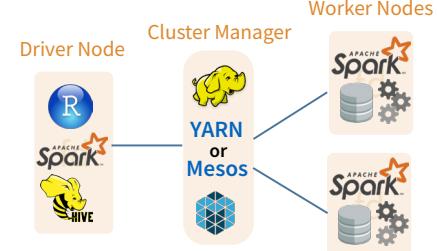
Starting with **version 1.044**, **RStudio Desktop**, **Server** and **Pro** include integrated support for the **sparklyr** package. You can create and manage connections to Spark clusters and local Spark instances from inside the IDE.

### RStudio Integrates with sparklyr

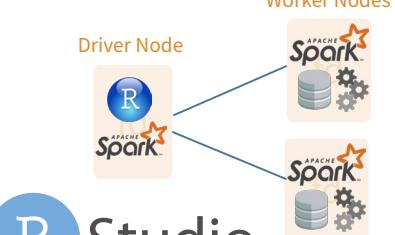


## Cluster Deployment

### MANAGED CLUSTER

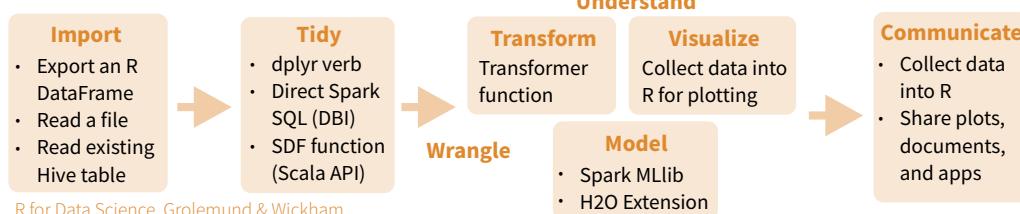


### STAND ALONE CLUSTER



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")
```

• 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

### IMPORTANT TUNING PARAMETERS with defaults

## 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
```

Bring data back into R memory for plotting

```
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()
```

```
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()`

# 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



## 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

# 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

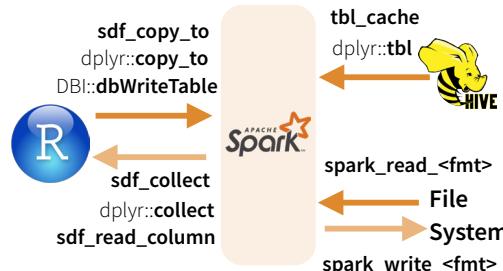
Arguments that apply to all functions: `x, path`

**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_obj()` 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")`

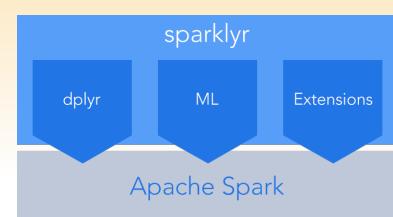
`ml_classification_eval(predicted_tbl_spark, label, predicted_lbl, metric = "f1")`

`ml_tree_feature_importance(sc, model)`

**sparklyr**

is an R interface  
for

APACHE  
SPARK



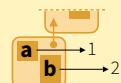
# Tidy evaluation with rlang :: CHEAT SHEET



## 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**

**Constant** - a bare value (i.e. an atomic vector of length 1). `is_bare_atomic(1)`

**abs** ( **1** )

**pi** — code  
3.14 — result

**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))`

**e**

**a + b**

**q**

**a + b, [a, b]**

**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.

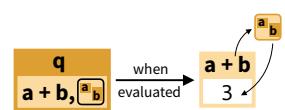
`a + b` `rlang::quo_get_expr(quo)` Return the expression of a quosure.

**Expression Vector** - a list of pieces of quoted code created by base R's `expression` and `parse` functions. Not to be confused with **expression**.

## 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).

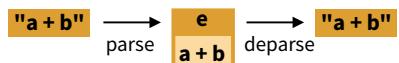
A quosure can be evaluated later in the stored environment to return a predictable result.

`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())`

## Parsing and Deparsing



**Parse** - Convert a string to a saved expression.

• • •

`rlang::parse_expr(x)` Convert a string to an expression. Also **parse\_exps**, **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)`

## 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)`



### EXPRESSION



**Quoted Expression** - An expression that has been saved by itself.

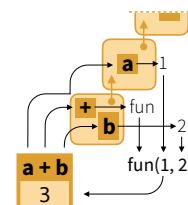
A quoted expression can be evaluated later to return a result that will depend on the environment it is evaluated in

`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 as a quosure. 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(...)`

## Evaluation



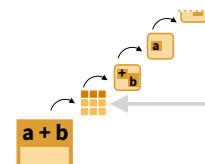
To evaluate an expression, R :

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)`



`a <- 1; b <- 2`  
`p <- quo(.data$a + !!b)`  
`mask <- tibble(a = 5, b = 6)`  
`eval_tidy(p, data = mask)`

### 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.

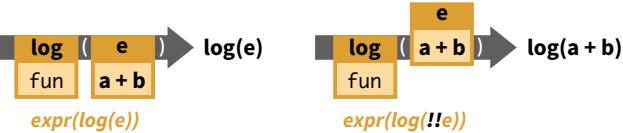
Use the pronoun `.data$` to force a symbol to be matched in data, and `!!` (see back) to force a symbol to be matched in the environments.



# Quasiquotation (!! , !!!, :=)

## QUOTATION

Storing an expression without evaluating it.  
`e <- expr(a + b)`

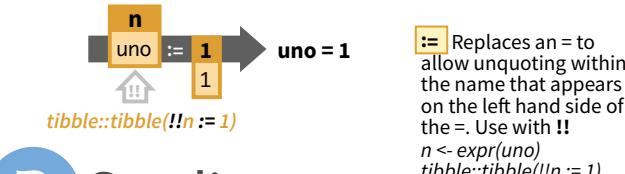
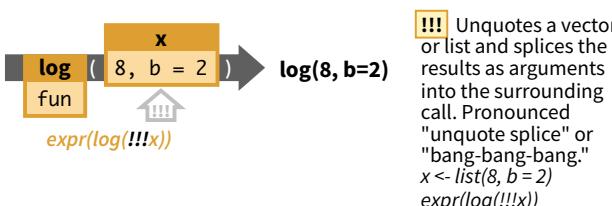
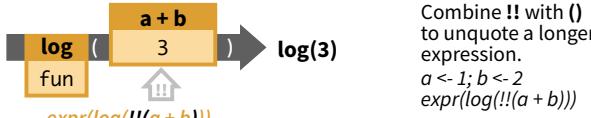
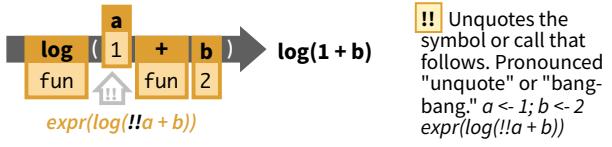


rlang provides !!, !!!, and := for doing quasiquotation.

!!, !!!, 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::enquo`.
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))
 rlang::eval_tidy(todo, df) 3
}

```

1. Capture user arguments with `rlang::enquo`.
2. **Unquote** user arguments into a new expression or quoture to use
3. **Evaluate** the new expression/ quoture instead of the original argument

## APPLY AN ARGUMENT TO A DATA FRAME

```

subset2 <- function(df, rows) {
 rows <- rlang::enquo(rows)
 vals <- rlang::eval_tidy(rows, data = df) 1
 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 QUASIQUOTATION (!! , !!!, :=)**

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
}

```

## PASS TO ARGUMENT NAMES OF A QUOTING FUNCTION

```

named_mean <- function(data, var) {
 require(dplyr)
 var <- rlang::ensym(var) 1
 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)
}

```

- 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

# 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 MacOS 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).
- `"LOO"` 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.

# Notes:



|                          |                                                              |
|--------------------------|--------------------------------------------------------------|
| <b>RStudio Community</b> | <a href="http://rstd.io/community">rstd.io/community</a>     |
| <b>Developer Blog</b>    | <a href="http://rstd.io/dev-blog">rstd.io/dev-blog</a>       |
| <b>R Views Blog</b>      | <a href="http://rstd.io/rviews-blog">rstd.io/rviews-blog</a> |
| <b>Tidyverse Blog</b>    | <a href="http://rstd.io/tidy-blog">rstd.io/tidy-blog</a>     |
| <b>Tensorflow Blog</b>   | <a href="http://rstd.io/tf-blog">rstd.io/tf-blog</a>         |
| <b>Twitter</b>           | <a href="http://rstd.io/twitter">rstd.io/twitter</a>         |
| <b>GitHub</b>            | <a href="http://rstd.io/github">rstd.io/github</a>           |
| <b>LinkedIn</b>          | <a href="http://rstd.io/linkedin">rstd.io/linkedin</a>       |
| <b>YouTube</b>           | <a href="http://rstd.io/youtube">rstd.io/youtube</a>         |
| <b>Facebook</b>          | <a href="http://rstd.io/facebook">rstd.io/facebook</a>       |

