

Base R Cheat Sheet

Getting Help

Accessing the help files

?mean

Get help of a particular function.

help.search('weighted mean')

Search the help files for a word or phrase.

help(package = 'dplyr')

Find help for a package.

More about an object

str(iris)

Get a summary of an object's structure.

class(iris)

Find the class an object belongs to.

Using Libraries

install.packages('dplyr')

Download and install a package from CRAN.

library(dplyr)

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

dplyr::select

Use a particular function from a package.

data(iris)

Load a build-in dataset into the environment.

Working Directory

getwd()

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

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

Change the current working directory.

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

Vectors			Programming					
Creating Vectors			For Loop			While Loop		
c(2, 4, 6)	2 4 6	Join elements into a vector	for (variable in sequence){	Do something	}	while (condition){	Do something	}
2:6	2 3 4 5 6	An integer sequence						
seq(2, 3, by=0.5)	2.0 2.5 3.0	A complex sequence						
rep(1:2, times=3)	1 2 1 2 1 2	Repeat a vector	for (i in 1:4){	j <- i + 10	print(j)	while (i < 5){	print(i)	i <- i + 1
rep(1:2, each=3)	1 1 1 2 2 2	Repeat elements of a vector						
Vector Functions								
sort(x)	rev(x)		if (condition){	Do something		functions_name <- function(var){	Do something	
		Return x sorted.	} else {	Do something different	}	return(new_variable)		
table(x)	unique(x)	See counts of values.						
Selecting Vector Elements								
By Position			If Statements			Functions		
x[4]	The fourth element.		if (i > 3){	print('Yes')		function_name <- function(var){		
x[-4]	All but the fourth.		} else {	print('No')		Do something		
x[2:4]	Elements two to four.					return(new_variable)		
x[!(2:4)]	All elements except two to four.							
x[c(1, 5)]	Elements one and five.		Example			Example		
By Value			if (i > 3){	print('Yes')		square <- function(x){		
x[x == 10]	Elements which are equal to 10.		} else {	print('No')		squared <- x*x		
x[x < 0]	All elements less than zero.					return(squared)		
x[x %in% c(1, 2, 5)]	Elements in the set 1, 2, 5.		Reading and Writing Data			Example		
Named Vectors			df <- read.table('file.txt')	write.table(df, 'file.txt')		Input	Ouput	Description
x['apple']	Element with name 'apple'.					df <- read.csv('file.csv')	write.csv(df, 'file.csv')	Read and write a delimited text file.
Conditions			load('file.RData')	save(df, file = 'file.Rdata')				Read and write a comma separated value file. This is a special case of read.table/write.table.
a == b	Are equal							Read and write an R data file, a file type special for R.
a != b	Not equal		a > b	Greater than	a >= b	a == b	Greater than or equal to	is.na(a)
			a < b	Less than	a <= b	a != b	Less than or equal to	is.null(a)
								Is null

Types

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

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

Maths Functions

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

Variable Assignment

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

The Environment

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

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

Matrixes

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

	<code>m[2,]</code> - Select a row	<code>t(m)</code> Transpose
	<code>m[, 1]</code> - Select a column	<code>m %*% n</code> Matrix Multiplication
	<code>m[2, 3]</code> - Select an element	<code>solve(m, n)</code> Find x in: $m \cdot x = n$

Lists

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

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

Also see the [dplyr](#) library.

Data Frames

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

x	y
1	a
2	b
3	c

Matrix subsetting

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

List subsetting

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

<i>Understanding a data frame</i>
<code>View(df)</code>
See the full data frame.

<i>head(df)</i>
See the first 6 rows.

`nrow(df)` Number of rows.

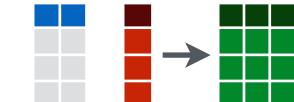
`ncol(df)`

Number of columns.

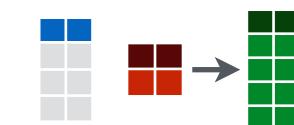
`dim(df)`

Number of columns and rows.

`cbind` - Bind columns.



`rbind` - Bind rows.



Strings

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

Factors

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

Statistics

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

Distributions

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

Plotting

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

Dates

See the [lubridate](#) library.

Data Wrangling with dplyr and tidyr

Cheat Sheet



Syntax - Helpful conventions for wrangling

`dplyr::tbl_df(iris)`

Converts data to `tbl` class. `tbl`'s are easier to examine than data frames. R displays only the data that fits onscreen:

```
Source: local data frame [150 x 5]
  Sepal.Length Sepal.Width Petal.Length
1          5.1        3.5         1.4
2          4.9        3.0         1.4
3          4.7        3.2         1.3
4          4.6        3.1         1.5
5          5.0        3.6         1.4
...
Variables not shown: Petal.Width (dbl), Species (fctr)
```

`dplyr::glimpse(iris)`

Information dense summary of `tbl` data.

`utils::View(iris)`

View data set in spreadsheet-like display (note capital V).

iris x					
<input type="checkbox"/> <input type="radio"/> <input type="button"/> <input type="button"/> Filter <input type="text"/>					
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa

`dplyr::%>%`

Passes object on left hand side as first argument (or . argument) of function on righthand side.

`x %>% f(y)` is the same as `f(x, y)`

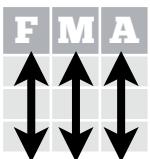
`y %>% f(x, ., z)` is the same as `f(x, y, z)`

"Piping" with `%>%` makes code more readable, e.g.

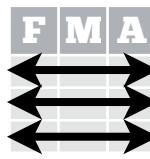
```
iris %>%
  group_by(Species) %>%
  summarise(avg = mean(Sepal.Width)) %>%
  arrange(avg)
```

Tidy Data - A foundation for wrangling in R

In a tidy data set:



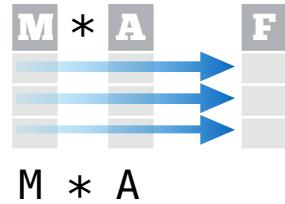
&



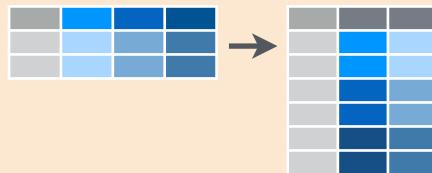
Each **variable** is saved in its own **column**

Each **observation** is saved in its own **row**

Tidy data complements R's **vectorized operations**. R will automatically preserve observations as you manipulate variables. No other format works as intuitively with R.

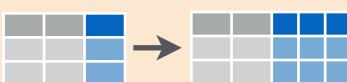


Reshaping Data - Change the layout of a data set



`tidyR::gather(cases, "year", "n", 2:4)`

Gather columns into rows.



`tidyR::spread(pollution, size, amount)`



`tidyR::separate(storms, date, c("y", "m", "d"))`

Separate one column into several.



`tidyR::unite(data, col, ..., sep)`

Unite several columns into one.

`dplyr::data_frame(a = 1:3, b = 4:6)`
Combine vectors into data frame (optimized).

`dplyr::arrange(mtcars, mpg)`
Order rows by values of a column (low to high).

`dplyr::arrange(mtcars, desc(mpg))`
Order rows by values of a column (high to low).

`dplyr::rename(tb, y = year)`
Rename the columns of a data frame.

Subset Observations (Rows)



`dplyr::filter(iris, Sepal.Length > 7)`

Extract rows that meet logical criteria.

`dplyr::distinct(iris)`

Remove duplicate rows.

`dplyr::sample_frac(iris, 0.5, replace = TRUE)`

Randomly select fraction of rows.

`dplyr::sample_n(iris, 10, replace = TRUE)`

Randomly select n rows.

`dplyr::slice(iris, 10:15)`

Select rows by position.

`dplyr::top_n(storms, 2, date)`

Select and order top n entries (by group if grouped data).

Subset Variables (Columns)



`dplyr::select(iris, Sepal.Width, Petal.Length, Species)`

Select columns by name or helper function.

Helper functions for select - ?select

`select(iris, contains("."))`

Select columns whose name contains a character string.

`select(iris, ends_with("Length"))`

Select columns whose name ends with a character string.

`select(iris, everything())`

Select every column.

`select(iris, matches(".t.))`

Select columns whose name matches a regular expression.

`select(iris, num_range("x", 1:5))`

Select columns named x1, x2, x3, x4, x5.

`select(iris, one_of(c("Species", "Genus")))`

Select columns whose names are in a group of names.

`select(iris, starts_with("Sepal"))`

Select columns whose name starts with a character string.

`select(iris, Sepal.Length:Petal.Width)`

Select all columns between Sepal.Length and Petal.Width (inclusive).

`select(iris, -Species)`

Select all columns except Species.

Logic in R - ?Comparison, ?base::Logic

<code><</code>	Less than	<code>!=</code>	Not equal to
<code>></code>	Greater than	<code>%in%</code>	Group membership
<code>==</code>	Equal to	<code>is.na</code>	Is NA
<code><=</code>	Less than or equal to	<code>!is.na</code>	Is not NA
<code>>=</code>	Greater than or equal to	<code>&, , !, xor, any, all</code>	Boolean operators

Summarise Data



dplyr::summarise(iris, avg = mean(Sepal.Length))

Summarise data into single row of values.

dplyr::summarise_each(iris, funs(mean))

Apply summary function to each column.

dplyr::count(iris, Species, wt = Sepal.Length)

Count number of rows with each unique value of variable (with or without weights).



Summarise uses **summary functions**, functions that take a vector of values and return a single value, such as:

dplyr::first

First value of a vector.

dplyr::last

Last value of a vector.

dplyr::nth

Nth value of a vector.

dplyr::n

of values in a vector.

dplyr::n_distinct

of distinct values in a vector.

IQR

IQR of a vector.

min

Minimum value in a vector.

max

Maximum value in a vector.

mean

Mean value of a vector.

median

Median value of a vector.

var

Variance of a vector.

sd

Standard deviation of a vector.

Group Data

dplyr::group_by(iris, Species)

Group data into rows with the same value of Species.

dplyr::ungroup(iris)

Remove grouping information from data frame.

iris %>% group_by(Species) %>% summarise(...)

Compute separate summary row for each group.



Make New Variables



dplyr::mutate(iris, sepal = Sepal.Length + Sepal.Width)

Compute and append one or more new columns.

dplyr::mutate_each(iris, funs(min_rank))

Apply window function to each column.

dplyr::transmute(iris, sepal = Sepal.Length + Sepal.Width)

Compute one or more new columns. Drop original columns.



Mutate uses **window functions**, functions that take a vector of values and return another vector of values, such as:

dplyr::lead

Copy with values shifted by 1.

dplyr::lag

Copy with values lagged by 1.

dplyr::dense_rank

Ranks with no gaps.

dplyr::min_rank

Ranks. Ties get min rank.

dplyr::percent_rank

Ranks rescaled to [0, 1].

dplyr::row_number

Ranks. Ties got to first value.

dplyr::ntile

Bin vector into n buckets.

dplyr::between

Are values between a and b?

dplyr::cume_dist

Cumulative distribution.

dplyr::cumall

Cumulative **all**

dplyr::cumany

Cumulative **any**

dplyr::cummean

Cumulative **mean**

cumsum

Cumulative **sum**

cummax

Cumulative **max**

cummin

Cumulative **min**

cumprod

Cumulative **prod**

pmax

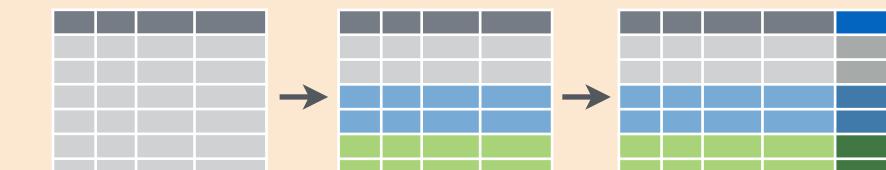
Element-wise **max**

pmin

Element-wise **min**

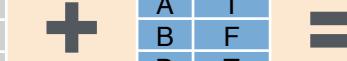
iris %>% group_by(Species) %>% mutate(...)

Compute new variables by group.



Combine Data Sets

a	b		
x1	x2	x1	x3
A	1	A	T
B	2	B	F
C	3	D	T



Mutating Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NA

x1	x3	x2
A	T	1
B	F	2
D	NA	NA

x1	x2	x3
A	1	T
B	2	F
C	3	NA

Filtering Joins

x1	x2
A	1
B	2

x1	x2
C	3

dplyr::semi_join(a, b, by = "x1")

All rows in a that have a match in b.

dplyr::anti_join(a, b, by = "x1")

All rows in a that do not have a match in b.

y	z		
x1	x2	x1	x2
A	1	B	2
B	2	C	3
C	3	D	4

Set Operations

x1	x2
B	2
C	3

x1	x2
A	1
B	2
C	3
D	4

Binding

x1	x2
A	1
B	2
C	3

x1	x2	x1	x2

<tbl_r cells="4" ix="1" maxcspan="1" maxrspan="

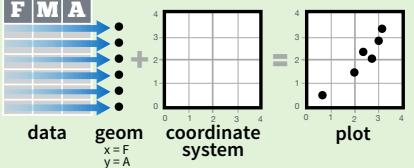
Data Visualization with ggplot2

Cheat Sheet

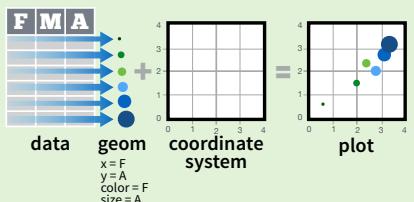


Basics

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



To display data values, map variables in the data set to aesthetic properties of the geom like **size**, **color**, and **x** and **y** locations.



Build a graph with **ggplot()** or **qplot()**

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

Begins a plot that you finish by adding layers to. No defaults, but provides more control than qplot().

```
data
ggplot(mpg, aes(hwy, cty)) +
  geom_point(aes(color = cyl)) +
  geom_smooth(method = "lm") +
  coord_cartesian() +
  scale_color_gradient() +
  theme_bw()
```

add layers, elements with +
layer = geom + default stat + layer specific mappings
additional elements

Add a new layer to a plot with a **geom_***() or **stat_***() function. Each provides a geom, a set of aesthetic mappings, and a default stat and position adjustment.

aesthetic mappings data geom

```
qplot(x = cty, y = hwy, color = cyl, 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.

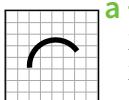
Geoms - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

Graphical Primitives

```
a <- ggplot(seals, aes(x = long, y = lat))
b <- ggplot(economics, aes(date, unemploy))
```



a + geom_blank()
(Useful for expanding limits)



a + geom_curve(aes(yend = lat + delta_lat, xend = long + delta_long, curvature = z))
x, yend, y, yend, alpha, angle, color, curvature, linetype, size



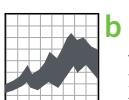
b + geom_path(lineend = "butt", linejoin = "round", linemitre = 1)
x, y, alpha, color, group, linetype, size



b + geom_polygon(aes(group = group))
x, y, alpha, color, fill, group, linetype, size



a + geom_rect(aes(xmin = long, ymin = lat, xmax = long + delta_long, ymax = lat + delta_lat))
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size



b + geom_ribbon(aes(ymin = unemploy - 900, ymax = unemploy + 900))
x, ymax, ymin, alpha, color, fill, group, linetype, size



a + geom_segment(aes(yend = lat + delta_lat, xend = long + delta_long))
x, yend, y, yend, alpha, color, linetype, size



a + geom_spoke(aes(yend = lat + delta_lat, xend = long + delta_long))
x, y, angle, radius, alpha, color, linetype, size

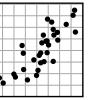
Two Variables

Continuous X, Continuous Y

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



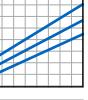
e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE)
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust



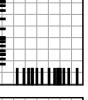
e + geom_jitter(height = 2, width = 2)
x, y, alpha, color, fill, shape, size



e + geom_point()
x, y, alpha, color, fill, shape, size, stroke



e + geom_quantile()
x, y, alpha, color, group, linetype, size, weight



e + geom_rug(sides = "bl")
x, y, alpha, color, linetype, size



e + geom_smooth(method = lm)
x, y, alpha, color, fill, group, linetype, size, weight



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

Continuous Bivariate Distribution

```
h <- ggplot(diamonds, aes(carat, price))
```



h + geom_bin2d(binwidth = c(0.25, 500))
x, y, alpha, color, fill, linetype, size, weight



h + geom_density2d()
x, y, alpha, colour, group, linetype, size



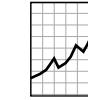
h + geom_hex()
x, y, alpha, colour, fill, size

Continuous Function

```
i <- ggplot(economics, aes(date, unemploy))
```



i + geom_area()
x, y, alpha, color, fill, linetype, size



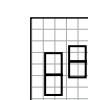
i + geom_line()
x, y, alpha, color, group, linetype, size



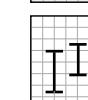
i + geom_step(direction = "hv")
x, y, alpha, color, group, linetype, size

Visualizing error

```
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se))
```



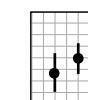
j + geom_crossbar(fatten = 2)
x, y, ymax, ymin, alpha, color, fill, group, linetype, size



j + geom_errorbar()
x, ymax, ymin, alpha, color, group, linetype, size, width (also **geom_errorbarh()**)



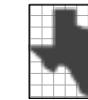
j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size



j + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

Maps

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



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

Three Variables

```
seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
l <- ggplot(seals, aes(long, lat))
```



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



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



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

R Markdown Cheat Sheet

learn more at rmarkdown.rstudio.com



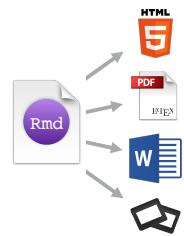
.Rmd files

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.



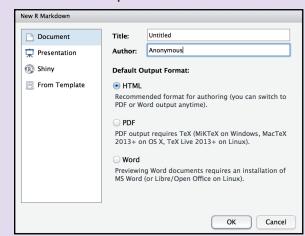
Dynamic Documents

You can choose to export the finished report as a html, pdf, MS Word, ODT, RTF, or markdown document; or as a html or pdf based slide show.

Workflow

1 Open a new .Rmd file

Use the wizard that opens to pre-populate the file with a template



.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**

2 Write document

by editing template

The screenshot shows the RStudio interface with an R Markdown file open. The file contains code chunks, a YAML header, and narrative text. Various menu items and buttons are highlighted, such as 'Knit HTML', 'Set preview location', and 'Run code chunk(s)'. The RStudio sidebar and file browser are also visible.

3 Knit document to create report

Use knit button or `render()` to knit

The screenshot shows the RStudio interface after knitting the R Markdown file. The results are displayed in the browser tab, showing the rendered HTML content with sections and tables. A console window below shows the command used to render the file.

4 Preview Output

in IDE window

The screenshot shows the RStudio interface with the rendered HTML report displayed in the browser window. The browser title is 'report.html'.

5 Publish

(optional) to web or server

Synch publish button to accounts at

- rpubs.com
- shinyapps.io
- RStudio Connect

Reload document

Find in document

File path to output document

6 Examine build log

in R Markdown console

7 Use output file

that is saved alongside .Rmd

render()

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

input - file to render

output_format

output_options - List of render options (as in YAML)

output_file

output_dir

params - list of params to use

envir - environment to evaluate code chunks in

encoding - of input file

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

The screenshot shows the RStudio interface with a YAML header for a shinyAppDir and the corresponding R code to render a table from the 'cars' dataset.

The screenshot shows the RStudio interface with the rendered shinyAppDir output, displaying a table titled 'How many cars?' with data for speed and dist.

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

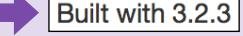
* Your report will be rendered as a Shiny app, which means you must choose an html output format, like `html_document`, and serve it with an active R Session.

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

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 = '##')

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

Code chunks

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



→

getRVersion()  
## [1] '3.2.3'

### Global options

Set with `knitr::opts_chunk$set()`, e.g.

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

Parameters

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

1 Add parameters

Create and set parameters in the header as sub-values of **params**

The screenshot shows the RStudio interface with a YAML header containing parameters: n: 100 and d: ! r Sys.Date().

2 Call parameters

Call parameter values in code as `params$<name>`

The screenshot shows the RStudio interface with a call to a parameter: Today's date is `r params\$d`.

3 Set parameters

Set values with **Knit with parameters** or the `params` argument of `render()`

The screenshot shows the RStudio interface with the 'Knit with Parameters...' option selected in the Knit menu.

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

R For Data Science Cheat Sheet

data.table

Learn R for data science **Interactively** at www.DataCamp.com



data.table

data.table is an R package that provides a high-performance version of base R's `data.frame` with syntax and feature enhancements for ease of use, convenience and programming speed.



Load the package:

```
> library(data.table)
```

Creating A data.table

<pre>> set.seed(45L) > DT <- data.table(V1=c(1L,2L), V2=LETTERS[1:3], V3=round(rnorm(4),4), V4=1:12)</pre>	Create a <code>data.table</code> and call it <code>DT</code>
---	--

Subsetting Rows Using i

<pre>> DT[3:5,]</pre>	Select 3rd to 5th row
<pre>> DT[3:5]</pre>	Select 3rd to 5th row
<pre>> DT[V2=="A"]</pre>	Select all rows that have value A in column v2
<pre>> DT[V2 %in% c("A", "C")]</pre>	Select all rows that have value A or C in column v2

Manipulating on Columns in j

<pre>> DT[,V2] [1] "A" "B" "C" "A" "B" "C" ... > DT[,.(V2,V3)] > DT[,sum(V1)] [1] 18 > DT[,(..(sum(V1), sd(V3))] V1 V2 1: 18 0.4546055 > DT[,(..(Aggregate=sum(V1), Sd.V3=sd(V3))] Aggregate Sd.V3 1: 18 0.4546055 > DT[,(..(V1,Sd.V3=sd(V3))] > DT[,(print(V2), plot(V3), NULL)]</pre>	Return v2 as a vector
	Return v2 and v3 as a <code>data.table</code>
	Return the sum of all elements of v1 in a vector
	Return the sum of all elements of v1 and the std. dev. of v3 in a <code>data.table</code>
	The same as the above, with new names
	Select column v2 and compute std. dev. of v3, which returns a single value and gets recycled
	Print column v2 and plot v3

Doing j by Group

<pre>> DT[,(..(V4.Sum=sum(V4)),by=V1] V1 V4.Sum 1: 1 36 2: 2 42 > DT[,(..(V4.Sum=sum(V4)), by=(V1,V2)] > DT[,(..(V4.Sum=sum(V4)), by=sign(V1-1)] sign V4.Sum 1: 0 36 2: 1 42 > DT[,(..(V4.Sum=sum(V4)), by=(V1.01=sign(V1-1))] > DT[1:5,(..(V4.Sum=sum(V4)), by=V1] > DT[,N,by=V1]</pre>	Calculate sum of v4 for every group in v1
	Calculate sum of v4 for every group in v1 and v2
	Calculate sum of v4 for every group in <code>sign(V1-1)</code>
	The same as the above, with new name for the variable you're grouping by
	Calculate sum of v4 for every group in v1 after subsetting on the first 5 rows
	Count number of rows for every group in v1

General form: DT[i, j, by]

“Take DT, subset rows using i, then calculate j grouped by by”



Adding/Updating Columns By Reference in j Using :=

```
> DT[,V1:=round(exp(V1),2)]
> DT
   V1 V2      V3 V4
1: 2.72 A -0.1107 1
2: 7.39 B -0.1427 2
3: 2.72 C -1.8893 3
4: 7.39 A -0.3571 4
...
> DT[,c("V1","V2"):=list(round(exp(V1),2),
LETTERS[4:6])]
> DT[, ' :=' (V1=round(exp(V1),2),
V2=LETTERS[4:6])][]
   V1 V2      V3 V4
1: 15.18 D -0.1107 1
2: 1619.71 E -0.1427 2
3: 15.18 F -1.8893 3
4: 1619.71 D -0.3571 4
> DT[,V1:=NULL]
> DT[,c("V1","V2"):=NULL]
> Cols.chosen=c("A","B")
> DT[,Cols.Chosen:=NULL]
> DT[,,(Cols.Chosen ):=NULL]
```

v1 is updated by what is after :=
Return the result by calling DT

Columns v1 and v2 are updated by what is after :=
Alternative to the above one. With [], you print the result to the screen

Remove v1
Remove columns v1 and v2

Delete the column with column name Cols.chosen
Delete the columns specified in the variable Cols.chosen

Advanced Data Table Operations

```
> DT[,-N-1]
> DT[,-N]
> DT[,.(V2,V3)]
> DT[,list(V2,V3)]
> DT[,mean(V3),by=.(V1,V2)]
```

V1 V2 V3
1: 1 A 0.4053
2: 1 B 0.4053
3: 1 C 0.4053
4: 2 A -0.6443
5: 2 B -0.6443
6: 2 C -0.6443

Return the penultimate row of the DT
Return the number of rows
Return v2 and v3 as a `data.table`
Return v2 and v3 as a `data.frame`
Return the result of j, grouped by all possible combinations of groups specified in by

.SD & .SDcols

```
> DT[,print(.SD),by=V2]
> DT[, .SD[c(1,.N)],by=V2]
> DT[,lapply(.SD,sum),by=V2]
> DT[,lapply(.SD,sum),by=V2,
.SDcols=c("V3","V4")]
   V2      V3 V4
1: A -0.478 22
2: B -0.478 26
3: C -0.478 30
> DT[,lapply(.SD,sum),by=V2,
.SDcols=paste0("V",3:4)]
```

Look at what .sd contains
Select the first and last row grouped by v2
Calculate sum of columns in .sd grouped by v2
Calculate sum of v3 and v4 in .sd grouped by v2
Calculate sum of v3 and v4 in .sd grouped by v2

Chaining

```
> DT <- DT[,(V4.Sum=sum(V4)),
by=V1]
   V1 V4.Sum
1: 1 36
2: 2 42
> DT[V4.Sum>40]
> DT[,(V4.Sum=sum(V4)),
by=V1][V4.Sum>40]
   V1 V4.Sum
1: 2 42
> DT[,(V4.Sum=sum(V4)),
by=V1][order(-V1)]
   V1 V4.Sum
1: 2 42
2: 1 36
```

Calculate sum of v4, grouped by v1

Select that group of which the sum is >40
Select that group of which the sum is >40 (chaining)

Calculate sum of v4, grouped by v1, ordered on v1

set() -Family

set()

Syntax: `for (i in from:to) set(DT, row, column, new value)`

```
> rows <- list(3:4,5:6)
> cols <- 1:2
> for(i in seq_along(rows))
  {set(DT,
    i=rows[[i]],
    j=cols[i],
    value=NA)}
```

Sequence along the values of rows, and for the values of cols, set the values of those elements equal to NA (invisible)

setnames()

Syntax: `setnames(DT, "old", "new")`

```
> setnames(DT,"V2","Rating")
> setnames(DT,
  c("V2","V3"),
  c("V2.rating","V3.DC"))
```

Set name of v2 to Rating (invisible)
Change 2 column names (invisible)

setnames()

Syntax: `setcolorder(DT, "neworder")`

```
> setcolorder(DT,
  c("V2","V1","V4","V3"))
```

Change column ordering to contents of the specified vector (invisible)

