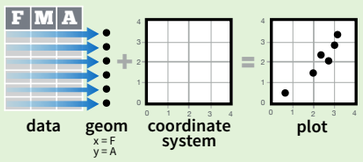
Data Visualization with ggplot2 Cheat Sheet

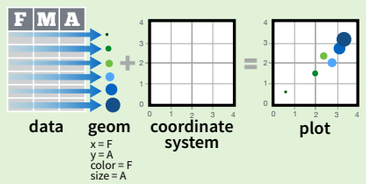
RStudio®

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().

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

data — 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.

One Variable

Continuous

```
a <- ggplot(mpg, aes(hwy))
```

a + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size
b + geom_area(aes(y = ..density..), stat = "bin")

a + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_density(aes(y = ..county..))

a + geom_dotplot()
x, y, alpha, color, fill

a + geom_freqpoly()
x, y, alpha, color, linetype, size
b + geom_freqpoly(aes(y = ..density..))

a + geom_histogram(binwidth = 5**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_histogram(aes(y = ..density..))

Discrete

```
b <- ggplot(mpg, aes(fl))
```

b + geom_bar()
x, alpha, color, fill, linetype, size, weight

Graphical Primitives

```
map <- map_data("state")
c <- ggplot(map, aes(long, lat))
```

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

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

d + geom_path(lineend="butt", linejoin="round', linemitre=1**)**
x, y, alpha, color, linetype, size

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

```
e <- ggplot(seals, aes(x = long, y = lat))
```

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

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

Two Variables

Continuous X, Continuous Y

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

f + geom_blank()
(Useful for expanding limits)

f + geom_jitter()
x, y, alpha, color, fill, shape, size

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

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

f + geom_rug(sides = "bl"**)**
alpha, color, linetype, size

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

f + geom_text(aes(label = cty)**)**
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Discrete X, Continuous Y

```
g <- ggplot(mpg, aes(class, hwy))
```

g + geom_bar(stat = "identity")
x, y, alpha, color, fill, linetype, size, weight

g + geom_boxplot()
lower, middle, upper, x, ymax, ymin, alpha, color, fill, linetype, shape, size, weight

g + geom_dotplot(binaxis = "y", stackdir = "center"**)**
x, y, alpha, color, fill

g + geom_violin(scale = "area"**)**
x, y, alpha, color, fill, linetype, size, weight

Discrete X, Discrete Y

```
h <- ggplot(diamonds, aes(cut, color))
```

h + geom_jitter()
x, y, alpha, color, fill, shape, size

Continuous Bivariate Distribution

```
i <- ggplot(movies, aes(year, rating))
```

i + geom_bin2d(binwidth = c(5, 0.5)**)**
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size, weight

i + geom_density2d()
x, y, alpha, colour, linetype, size

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

Continuous Function

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

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

j + geom_line()
x, y, alpha, color, linetype, size

j + geom_step(direction = "hv"**)**
x, y, alpha, color, linetype, size

Visualizing error

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

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

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

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

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

Maps

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

l + 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))
m <- ggplot(seals, aes(long, lat))
```

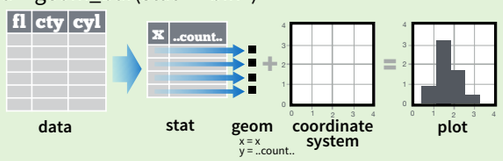
m + geom_contour(aes(z = z)**)**
x, y, z, alpha, colour, linetype, size, weight

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

m + geom_tile(aes(fill = z)**)**
x, y, alpha, color, fill, linetype, size (slow)

Stats - An alternative way to build a layer

Some plots visualize a **transformation** of the original data set. Use a **stat** to choose a common transformation to visualize, e.g. **a + geom_bar(stat = "bin")**



Each stat creates additional variables to map aesthetics to. These variables use a common **..name..** syntax.

stat functions and geom functions both combine a stat with a geom to make a layer, i.e. **stat_bin(geom="bar")** does the same as **geom_bar(stat="bin")**

stat function | layer specific mappings | variable created by transformation

i + stat_density2d(aes(fill = ..level..), geom = "polygon", n = 100**)**

geom for layer | parameters for stat

a + stat_bin(binwidth = 1, origin = 10**)** — 1D distributions
x, y | ..count.., ..ncount.., ..density.., ..ndensity..
a + stat_bindot(binwidth = 1, binaxis = "x"**)**
x, y, | ..count.., ..ncount..
a + stat_density(adjust = 1, kernel = "gaussian"**)**
x, y, | ..count.., ..density.., ..scaled..

f + stat_bin2d(bins = 30, drop = TRUE**)** — 2D distributions
x, y, fill | ..count.., ..density..
f + stat_binhex(bins = 30**)**
x, y, fill | ..count.., ..density..
f + stat_density2d(contour = TRUE, n = 100**)**
x, y, color, size | ..level..

m + stat_contour(aes(z = z)**)** — 3 Variables
x, y, z, order | ..level..
m+ stat_spoke(aes(radius= z, angle = z)**)**
angle, radius, x, xend, y, yend | ..x.., ..xend.., ..y.., ..yend..
m + stat_summary_hex(aes(z = z), bins = 30, fun = mean**)**
x, y, z, fill | ..value..
m + stat_summary2d(aes(z = z), bins = 30, fun = mean**)**
x, y, z, fill | ..value..

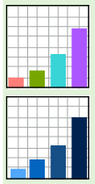
g + stat_boxplot(coef = 1.5**)** — Comparisons
x, y | ..lower.., ..middle.., ..upper.., ..outliers..
g + stat_ydensity(adjust = 1, kernel = "gaussian", scale = "area"**)**
x, y | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

f + stat_ecdf(n = 40**)** — Functions
x, y | ..x.., ..y..
f + stat_quantile(quantiles = c(0.25, 0.5, 0.75), formula = y ~ log(x), method = "rq"**)**
x, y | ..quantile.., ..x.., ..y..
f + stat_smooth(method = "auto", formula = y ~ x, se = TRUE, n = 80, fullrange = FALSE, level = 0.95**)**
x, y | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

ggplot() + stat_function(aes(x = -3:3), fun = dnorm, n = 101, args = list(sd=0.5)**)** — General Purpose
x | ..y..
f + stat_identity()
ggplot() + stat_qq(aes(sample=1:100), distribution = qt, dparams = list(df=5)**)**
sample, x, y | ..x.., ..y..
f + stat_sum()
x, y, size | ..size..
f + stat_summary(fun.data = "mean_cl_boot"**)**
f + stat_unique()

Scales

Scales control how a plot maps data values to the visual values of an aesthetic. To change the mapping, add a custom scale.



n <- b + geom_bar(aes(fill = fl)**)**
n

scale_ | aesthetic to adjust | prepackaged scale to use | scale specific arguments

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

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

General Purpose scales

Use with any aesthetic:
alpha, color, fill, linetype, shape, size

scale_*_continuous() - map cont' values to visual values
scale_*_discrete() - map discrete values to visual values
scale_*_identity() - use data values **as** visual values
scale_*_manual(values = c()**)** - map discrete values to manually chosen visual values

X and Y location scales

Use with x or y aesthetics (x shown here)

scale_x_date(labels = date_format("%m/%d"), breaks = date_breaks("2 weeks")**)** - treat x values as dates. See ?strptime for label formats.
scale_x_datetime() - treat x values as date times. Use same arguments as scale_x_date().
scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale

Color and fill scales

Discrete

n <- b + geom_bar(aes(fill = fl)**)**

n + scale_fill_brewer(palette = "Blues"**)**
For palette choices:
library(RColorBrewer)
display.brewer.all()

n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red"**)**

Continuous

o <- a + geom_dotplot(aes(fill = ..x..)**)**

o + scale_fill_gradient(low = "red", high = "yellow"**)**

o + scale_fill_gradient2(low = "red", high = "blue", mid = "white", midpoint = 25**)**

o + scale_fill_gradientn(colours = terrain.colors(6)**)**
Also: rainbow(), heat.colors(), topo.colors(), cm.colors(), RColorBrewer::brewer.pal()

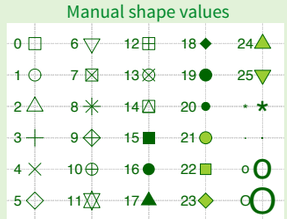
Shape scales

p <- f + geom_point(aes(shape = fl)**)**

p + scale_shape(solid = FALSE**)**

p + scale_shape_manual(values = c(3:7)**)**
Shape values shown in chart on right

Manual shape values



Size scales

q <- f + geom_point(aes(size = cyl)**)**

q + scale_size_area(max = 6**)**
Value mapped to area of circle (not radius)

Coordinate Systems

r <- b + geom_bar()

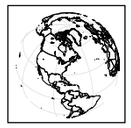
r + coord_cartesian(xlim = c(0, 5)**)**
xlim, ylim
The default cartesian coordinate system

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

r + coord_flip()
xlim, ylim
Flipped Cartesian coordinates

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

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



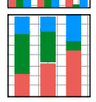
z + coord_map(projection = "ortho", orientation=c(41, -74, 0)**)**
projection, orientation, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)

Position Adjustments

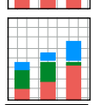
Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

s <- ggplot(mpg, aes(fl, fill = drv))

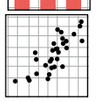
s + geom_bar(position = "dodge")
Arrange elements side by side



s + geom_bar(position = "fill")
Stack elements on top of one another, normalize height



s + geom_bar(position = "stack")
Stack elements on top of one another

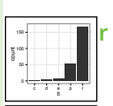


f + geom_point(position = "jitter")
Add random noise to X and Y position of each element to avoid overplotting

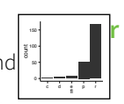
Each position adjustment can be recast as a function with manual **width** and **height** arguments

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

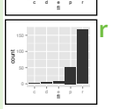
Themes



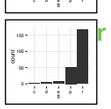
r + theme_bw()
White background with grid lines



r + theme_classic()
White background no gridlines



r + theme_grey()
Grey background (default theme)



r + theme_minimal()
Minimal theme

ggthemes - Package with additional ggplot2 themes

Faceting

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

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

t + facet_grid(. ~ fl)
facet into columns based on fl

t + facet_grid(year ~ .)
facet into rows based on year

t + facet_grid(year ~ fl)
facet into both rows and columns

t + facet_wrap(~ fl)
wrap facets into a rectangular layout

Set **scales** to let axis limits vary across facets

t + facet_grid(y ~ x, scales = "free")
x and y axis limits adjust to individual facets
- **"free_x"** - x axis limits adjust
- **"free_y"** - y axis limits adjust

Set **labeller** to adjust facet labels

t + facet_grid(. ~ fl, labeller = label_both)

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

t + facet_grid(. ~ fl, labeller = label_bquote(alpha ^ .(x)**))**

α^c	α^d	α^e	α^p	α^r

t + facet_grid(. ~ fl, labeller = label_parsed)

c	d	e	p	r

Labels

t + ggtitle("New Plot Title"**)**
Add a main title above the plot
t + xlab("New X label"**)**
Change the label on the X axis
t + ylab("New Y label"**)**
Change the label on the Y axis
t + labs(title =" New title", x = "New x", y = "New y"**)**
All of the above

Use scale functions to update legend labels

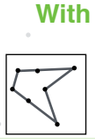
Legends

t + theme(legend.position = "bottom"**)**
Place legend at "bottom", "top", "left", or "right"
t + guides(color = "none"**)**
Set legend type for each aesthetic: colorbar, legend, or none (no legend)
t + scale_fill_discrete(name = "Title", labels = c("A", "B", "C"**)**
Set legend title and labels with a scale function.

Zooming

Without clipping (preferred)
t + coord_cartesian(xlim = c(0, 100), ylim = c(10, 20)**)**

With clipping (removes unseen data points)



t + xlim(0, 100**) + ylim(**10, 20**)**
t + scale_x_continuous(limits = c(0, 100)**) + scale_y_continuous(**limits = c(0, 100)**)**

Data Wrangling
with dplyr and tidyr
Cheat Sheet

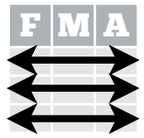
RStudio

Tidy Data - A foundation for wrangling in R

In a tidy data set:

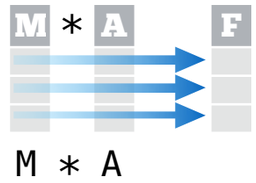
Each **variable** is saved in its own **column**

&



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.



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

Filter

	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.

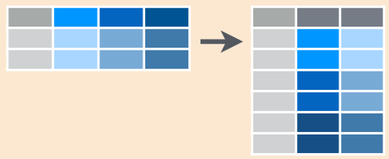
`x %>% f(y)` *is the same as* `f(x, y)`

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

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

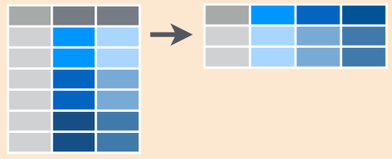
```
iris %>%
  group_by(Species) %>%
  summarise(avg = mean(Sepal.Width)) %>%
  arrange(avg)
```

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)**

Spread rows into columns.

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).

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

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

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.

devtools::install_github("rstudio/EDAWR") for data sets

Learn more with browseVignettes(package = c("dplyr", "tidyr")) • dplyr 0.4.0• tidyr 0.2.0 • Updated: 1/15

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.

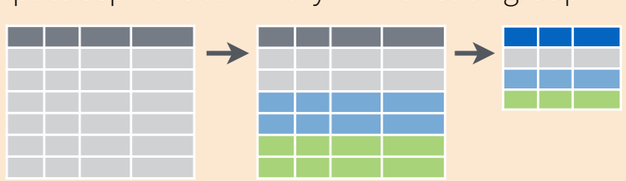
sd
Standard deviation of a vector.

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`

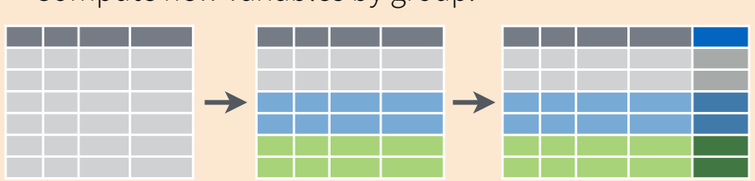
cummin
Cumulative `min`

cumprod
Cumulative `prod`

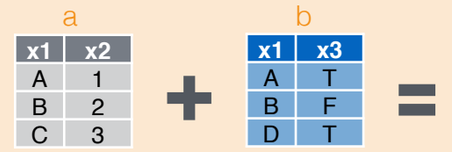
pmax
Element-wise `max`

pmin
Element-wise `min`

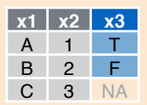
iris %>% group_by(Species) %>% mutate(…)
Compute new variables by group.



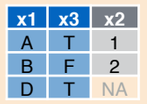
Combine Data Sets



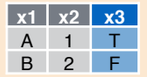
Mutating Joins



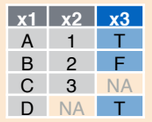
dplyr::**left_join(a, b, by = "x1")**
Join matching rows from b to a.



dplyr::**right_join(a, b, by = "x1")**
Join matching rows from a to b.

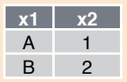


dplyr::**inner_join(a, b, by = "x1")**
Join data. Retain only rows in both sets.

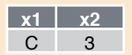


dplyr::**full_join(a, b, by = "x1")**
Join data. Retain all values, all rows.

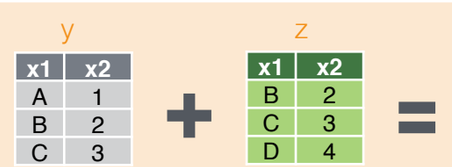
Filtering Joins



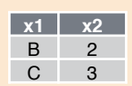
dplyr::**semi_join(a, b, by = "x1")**
All rows in a that have a match in b.



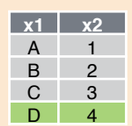
dplyr::**anti_join(a, b, by = "x1")**
All rows in a that do not have a match in b.



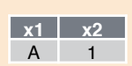
Set Operations



dplyr::**intersect(y, z)**
Rows that appear in both y and z.

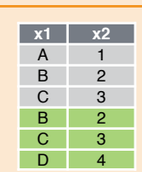


dplyr::**union(y, z)**
Rows that appear in either or both y and z.

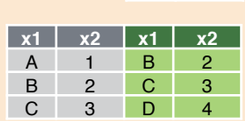


dplyr::**setdiff(y, z)**
Rows that appear in y but not z.

Binding



dplyr::**bind_rows(y, z)**
Append z to y as new rows.



dplyr::**bind_cols(y, z)**
Append z to y as new columns.
Caution: matches rows by position.

devtools::**install_github("rstudio/EDAWR")** for data sets

Learn more with **browseVignettes(package = c("dplyr", "tidyr"))** • dplyr 0.4.0• tidyr 0.2.0 • Updated: 1/15

R Markdown Cheat Sheet

learn more at rmarkdown.rstudio.com

rmarkdown 0.2.50 Updated: 8/14

RStudio®

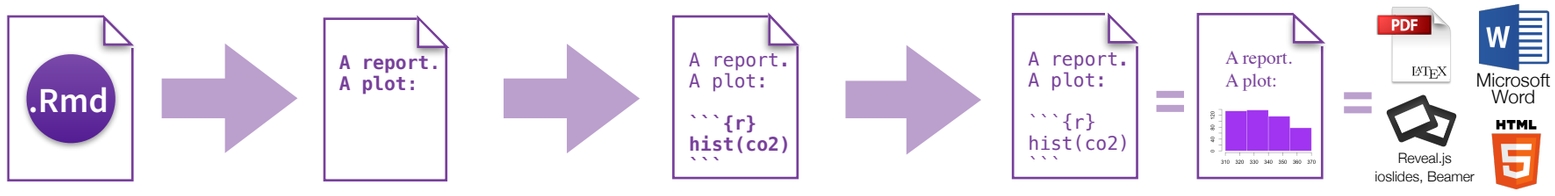
1. Workflow R Markdown is a format for writing reproducible, dynamic reports with R. Use it to embed R code and results into slideshows, pdfs, html documents, Word files and more. To make a report:

i. Open - Open a file that uses the .Rmd extension.

ii. Write - Write content with the easy to use R Markdown syntax

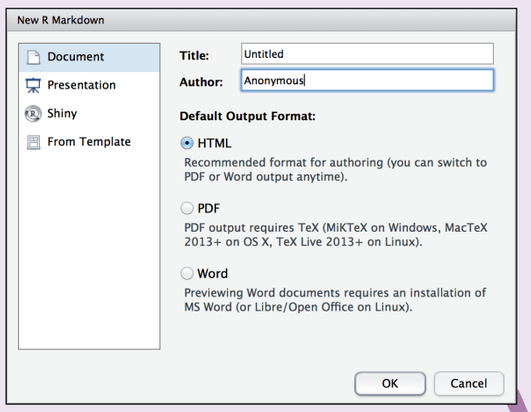
iii. Embed - Embed R code that creates output to include in the report

iv. Render - Replace R code with its output and transform the report into a slideshow, pdf, html or ms Word file.



2. Open File Start by saving a text file with the extension .Rmd, or open an RStudio Rmd template

- In the menu bar, click **File ▶ New File ▶ R Markdown…**
- A window will open. Select the class of output you would like to make with your .Rmd file
- Select the specific type of output to make with the radio buttons (you can change this later)
- Click OK



4. Choose Output Write a YAML header that explains what type of document to build from your R Markdown file.

YAML

A YAML header is a set of key: value pairs at the start of your file. Begin and end the header with a line of three dashes (- - -)

```
---
title: "Untitled"
author: "Anonymous"
output: html_document
---

This is the start of my
report. The above is metadata
saved in a YAML header.
```

The RStudio template writes the YAML header for you

The output value determines which type of file R will build from your .Rmd file (in Step 6)

output: html_document	html file (web page)
output: pdf_document	pdf document
output: word_document	Microsoft Word .docx
output: beamer_presentation	beamer slideshow (pdf)
output: ioslides_presentation	ioslides slideshow (html)

3. Markdown Next, write your report in plain text. Use markdown syntax to describe how to format text in the final report.

syntax

```
Plain text
End a line with two spaces to start a new paragraph.
*italics* and _italics_
**bold** and __bold__
superscript^2^
~~strikethrough~~
[link](www.rstudio.com)

# Header 1

## Header 2

### Header 3

#### Header 4

##### Header 5

###### Header 6

endash: --
emdash: ---
ellipsis: ...
inline equation: $A = \pi*r^{2}$
image: ![](path/to/smallorb.png)

horizontal rule (or slide break):

***

> block quote

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

1. ordered list
2. item 2
    + sub-item 1
    + sub-item 2

Table Header  | Second Header
------------- | -------------
Table Cell    | Cell 2
Cell 3        | Cell 4
```

becomes

Plain text
End a line with two spaces to start a new paragraph.
italics and *italics*
bold and **bold**
superscript^2
~~strikethrough~~
link

Header 1

Header 2

Header 3

Header 4

Header 5

Header 6

endash: –
emdash: —
ellipsis: …
inline equation: $A = \pi * r^2$
image:
horizontal rule (or slide break):

> block quote

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

1. ordered list
2. item 2
    - sub-item 1
    - sub-item 2

Table Header	Second Header
Table Cell	Cell 2
Cell 3	Cell 4

5. Embed Code Use knitr syntax to embed R code into your report. R will run the code and include the results when you render your report.

inline code

Surround code with back ticks and r.
R replaces inline code with its results.

```
Two plus two
equals `r 2 + 2`.
```

▶ Two plus two equals 4.

code chunks

Start a chunk with ```{r}.
End a chunk with ```

```
Here's some code
```{r}
dim(iris)
```
```

▶ Here's some code

```
dim(iris)
```

```
## [1] 150   5
```

display options

Use knitr options to style the output of a chunk.
Place options in brackets above the chunk.

```
Here's some code
```{r eval=FALSE}
dim(iris)
```
```

▶ Here's some code

```
dim(iris)
```

```
Here's some code
```{r echo=FALSE}
dim(iris)
```
```

▶ Here's some code

```
## [1] 150   5
```

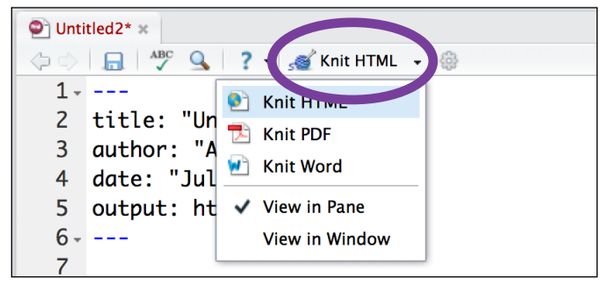
option	default	effect
eval	TRUE	Whether to evaluate the code and include its results
echo	TRUE	Whether to display code along with its results
warning	TRUE	Whether to display warnings
error	FALSE	Whether to display errors
message	TRUE	Whether to display messages
tidy	FALSE	Whether to reformat code in a tidy way when displaying it
results	"markup"	"markup", "asis", "hold", or "hide"
cache	FALSE	Whether to cache results for future renders
comment	"##"	Comment character to preface results with
fig.width	7	Width in inches for plots created in chunk
fig.height	7	Height in inches for plots created in chunk

For more details visit yihui.name/knitr/

6. Render Use your .Rmd file as a blueprint to build a finished report.

Render your report in one of two ways

1. Run **rmarkdown::render("<file path>")**
2. Click the **knit HTML** button at the top of the RStudio scripts pane



When you render, R will

- execute each embedded code chunk and insert the results into your report
- build a new version of your report in the output file type
- open a preview of the output file in the viewer pane
- save the output file in your working directory

7. Interactive Docs Turn your report into an interactive Shiny document in 3 steps

1 Add **runtime: shiny** to the YAML header

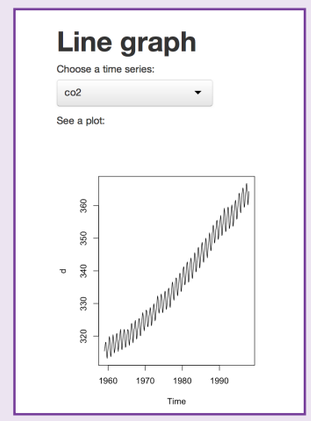
```
---
title: "Line graph"
output: html_document
runtime: shiny
---
```

2 In the code chunks, add Shiny **input** functions to embed widgets. Add Shiny **render** functions to embed reactive output

```
---
title: "Line graph"
output: html_document
runtime: shiny
---

Choose a time series:
```{r echo = FALSE}
selectInput("data", "",
  c("co2", "lh"))
```
See a plot:
```{r echo = FALSE}
renderPlot({
  d <- get(input$data)
  plot(d)
})
```
```

3 Render with **rmarkdown::run** or click **Run Document** in RStudio



* *Note: your report will be a Shiny app, which means you must choose an html output format, like* ***html_document*** *(for an interactive report) or* ***ioslides_presentation*** *(for an interactive slideshow).*

8. Publish Share your report where users can visit it online

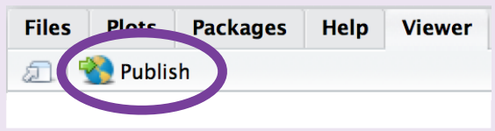
Rpubs.com

Share non-interactive documents on RStudio's free R Markdown publishing site
www.rpubs.com

ShinyApps.io

Host an interactive document on RStudio's server. Free and paid options
www.shinyapps.io

Click the "Publish" button in the RStudio preview window to publish to rpubs.com with one click.



9. Learn More

Documentation and examples - rmarkdown.rstudio.com
Further Articles - shiny.rstudio.com/articles
- blog.rstudio.com
- @rstudio

R Markdown Reference Guide

Learn more about R Markdown at rmarkdown.rstudio.com

Learn more about Interactive Docs at shiny.rstudio.com/articles

Contents:

Syntax

```
Plain text

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

*italics* and _italics_

**bold** and __bold__

superscript^2^

~~strikethrough~~

[link](www.rstudio.com)

# Header 1

## Header 2

### Header 3

#### Header 4

##### Header 5

###### Header 6

endash: --

emdash: ---

ellipsis: ...

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

image: ![](path/to/smallorb.png)

horizontal rule (or slide break):

***

> block quote

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

1. ordered list
2. item 2
    + sub-item 1
    + sub-item 2

Table Header  | Second Header
------------- | -------------
Table Cell    | Cell 2
Cell 3        | Cell 4
```

Becomes

Plain text

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

italics and *italics*

bold and **bold**

superscript^2

~~strikethrough~~

link

Header 1

Header 2

Header 3

Header 4

Header 5

Header 6

endash: –

emdash: —

ellipsis: …

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

image: 

horizontal rule (or slide break):

> block quote

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

1. ordered list
2. item 2
    - sub-item 1
    - sub-item 2

Table Header	Second Header
Table Cell	Cell 2
Cell 3	Cell 4

R Markdown Reference Guide

Learn more about R Markdown at rmarkdown.rstudio.com
Learn more about Interactive Docs at shiny.rstudio.com/articles

Contents:

Syntax	Becomes

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

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

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

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

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

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

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

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

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

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

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

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

Chunk options

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

R Markdown Reference Guide

Learn more about R Markdown at rmarkdown.rstudio.com
Learn more about Interactive Docs at shiny.rstudio.com/articles

Contents:

Chunk options (Continued)

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

R Markdown Reference Guide

Learn more about R Markdown at rmarkdown.rstudio.com
Learn more about Interactive Docs at shiny.rstudio.com/articles

Contents:

Templates

html_document
pdf_document
word_document
md_document
ioslides_presentation
slidy_presentation
beamer_presentation

Basic YAML

```
---
title: "A Web Doc"
author: "John Doe"
date: "May 1, 2015"
output: md_document
---
```

Template options

```
---
title: "Chapters"
output:
  html_document:
    toc: true
    toc_depth: 2
---
```

Latex options

```
---
title: "My PDF"
output: pdf_document
fontsize: 11pt
geometry: margin=1in
---
```

Interactive Docs

```
---
title: "Slides"
output:
  slidy_presentation:
    incremental: true
runtime: shiny
---
```

Syntax for slide formats (ioslides, slidy, beamer)

```
# Dividing slides 1

Pandoc will start a new slide at each first level header

## Header 2

… as well as each second level header

***

You can start a new slide with a horizontal rule`***` if you do not want
a header.

## Bullets

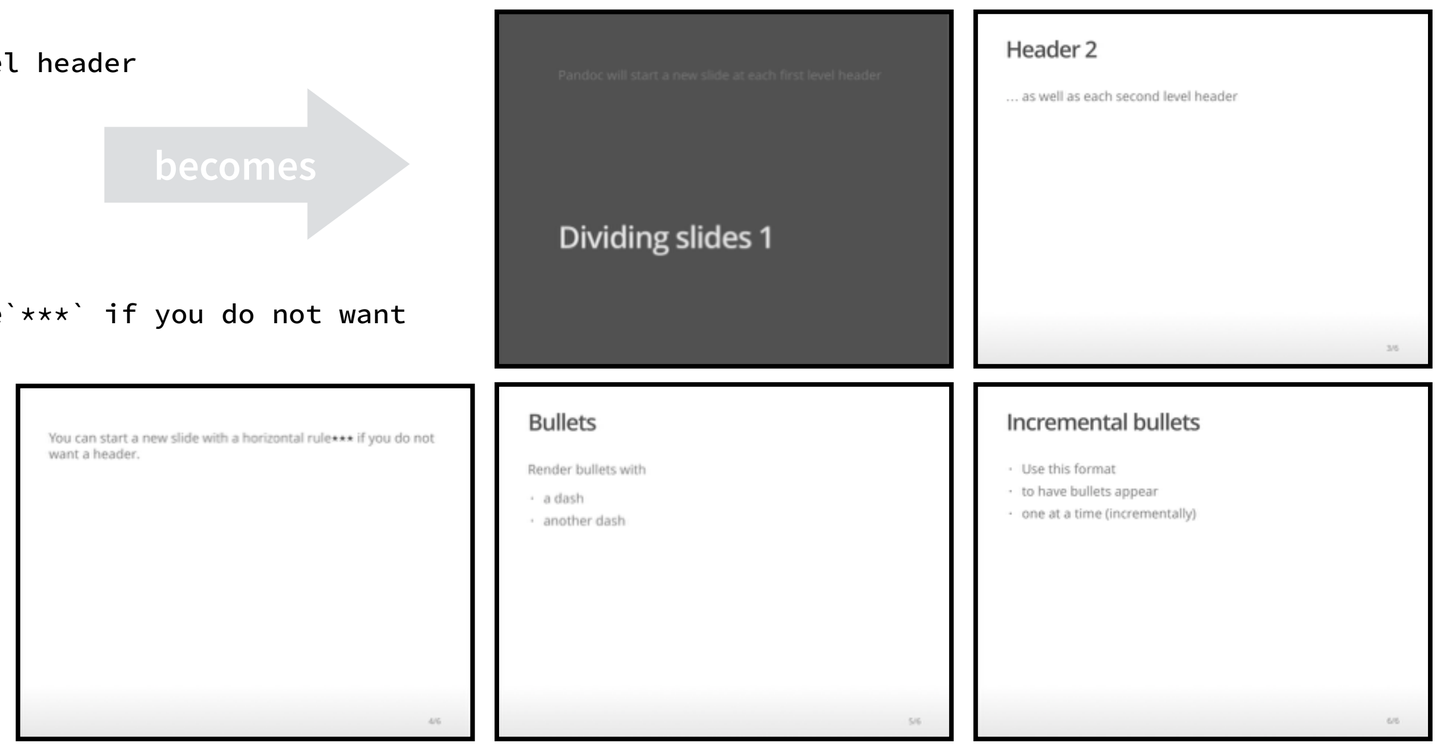
Render bullets with

- a dash
- another dash

## Incremental bullets

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

becomes



Slide display modes

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

ioslides

f - enable fullscreen mode
w - toggle widescreen mode
o - enable overview mode
h - enable code highlight mode
p - show presenter notes

slidy

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

Top level options to customize LaTex (pdf) output

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

R Markdown Reference Guide

Learn more about R Markdown at rmarkdown.rstudio.com
Learn more about Interactive Docs at shiny.rstudio.com/articles

Contents:

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

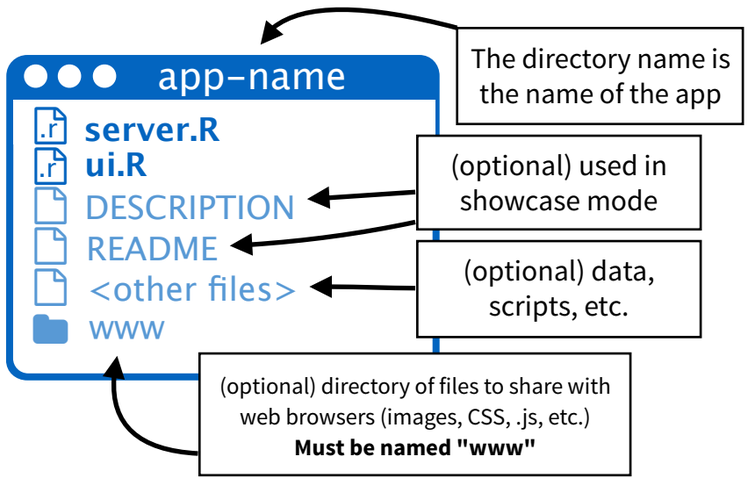
Shiny Cheat Sheet

learn more at shiny.rstudio.com

Shiny 0.10.0 Updated: 6/14

RStudio

1. Structure Each app is a directory that contains a **server.R** file and usually a **ui.R** file (plus optional extra files)



2. server.R A set of instructions that build the R components of your app. To write server.R:

- Ⓐ Provide server.R with the minimum necessary code, **shinyServer(function(input, output) {})**
- Ⓑ Define the R components for your app between the braces that follow **function(input, output)**
- Ⓒ Save each R component in your UI as **output$<component name>**
- Ⓓ Create each output component with a render* function.
- Ⓔ Give each render* function the R code the server needs to build the component. The server will note any reactive values that appear in the code and will rebuild the component whenever these values change.
- Ⓕ Refer to widget values with **input$<widget name>**

server.R

```
  # load libraries, scripts, data

Ⓐ shinyServer(function(input, output) {Ⓑ

  # make user specific variables

    output$text <- renderText({
      input$title
    })

                          Ⓓ
Ⓒ  output$plot <- renderPlot({
      x <- mtcars[ , input$x]Ⓕ
Ⓔ     y <- mtcars[ , input$y]
      plot(x, y, pch = 16)
    })

})
```

render* functions

function	expects	creates
renderDataTable	any table-like object	DataTables.js table
renderImage	list of image attributes	HTML image
renderPlot	plot	plot
renderPrint	any printed output	text
renderTable	any table-like object	plain table
renderText	character string	text
renderUI	Shiny tag object or	UI element (HTML)

input values are reactive.
They must be surrounded with one of:

render* - creates a shiny UI component
reactive - creates a reactive expression
observe - creates a reactive observer
isolate - creates a non-reactive copy of a reactive object

3. Execution Place code where it will be run the minimum necessary number of times

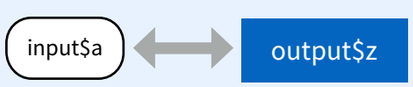
Run once - code placed *outside of* ***shinyServer*** will be run once, when you first launch your app. Use this code to set up the tools that your server will only need one copy of.

Run once per user - code placed *inside* ***shinyServer*** will be run once each time a user visits your app (or refreshes his or her browser). Use this code to set up the tools that your server will need a unique copy of for each user.

Run often - code placed within a render*, reactive, or observe function will be run many times. Place here only the code that the server needs to rebuild a UI component after a widget changes.

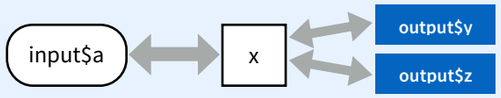
4. Reactivity When an input changes, the server will rebuild each output that depends on it (even if the dependence is indirect). You can control this behavior by shaping the chain of dependence.

render* - An output will automatically update whenever an input in its render* function changes.




```
output$z <- renderText({
  input$a
})
```

Reactive expression - use reactive to create objects that will be used in multiple outputs.




```
x <- reactive({
  input$a
})

output$y <- renderText({
  x()
})
output$z <- renderText({
  x()
})
```

isolate - use use isolate to use an input without depending on it. Shiny will not rebuild the output when the isolated input changes.


```
output$z <- renderText({
  paste(
   isolate(input$a),
   input$b
  )
)
```

observe - use observe to create code that runs when an input changes, but does not create an output object.


```
observe({
  input$a
  # code to run
})
```

ui.R

```
A shinyUI(fluidPage(

    titlePanel("mtcars data"),
  B sidebarLayout(
      sidebarPanel(

    C   textInput("title", "Plot title:",
          value = "x v y"),

        selectInput("x", "Choose an x var:",
          choices = names(mtcars),
          selected = "disp"),

        selectInput("y", "Choose a y var:",
          choices = names(mtcars),
          selected = "mpg")
      ),

      mainPanel(
        h3(textOutput("text")),
        plotOutput("plot")
      )
    )
))
```

C In each panel or column, place…

R components - These are the output objects that you defined in **server.R**. To place a component:

1. Select the ***Output** function that builds the type of object you want to place in the UI.
2. Pass the ***Output** function a character string that corresponds to the name you assigned the object in **server.R**, e.g.

output$plot <- renderPlot({ ... }) ⟷ plotOutput("plot")

*Output functions

dataTableOutput	tableOutput
htmlOutput	textOutput
imageOutput	uiOutput
plotOutput	verbatimTextOutput

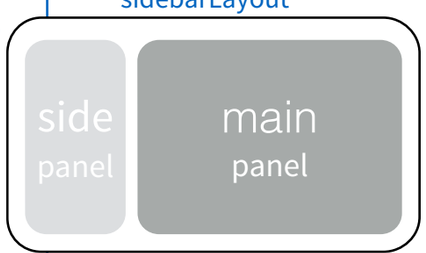
5. ui.R A description of your app's User Interface (UI), the web page that displays your app.

To write ui.R:

A Include the minimum necessary code for ui.R, **shinyUI(fluidPage())**
* note: use `navbarPage` instead of `fluidPage` if you'd like your app to have multiple pages connected by a navbar

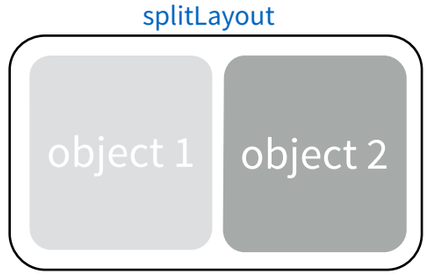
B Build a layout for your UI. **sidebarLayout** provides a default layout when used with **sidebarPanel** and **mainPanel.** **splitLayout**, **flowLayout**, and **inputLayout** divide the page into equally spaced regions. **fluidRow** and **column** work together to create a grid-based layout, which you can use to layout a page or a panel.

sidebarLayout



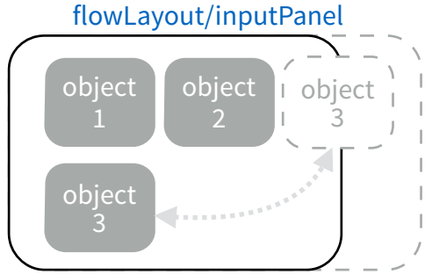

```
shinyUI(fluidPage(
  sidebarLayout(
    sidebarPanel(…),
    mainPanel(…)
  )
))
```

splitLayout



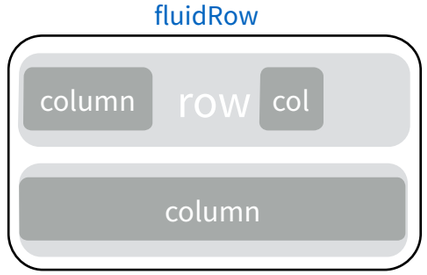

```
shinyUI(fluidPage(
  splitLayout(
    numericInput(…),
    selectInput(…)
  )
))
```

flowLayout/inputPanel

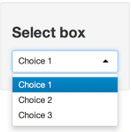



```
shinyUI(fluidPage(
  flowLayout(
    numericInput(…),
    selectInput(…),
    sliderInput(…)
  )
))
```

fluidRow




```
shinyUI(fluidPage(
  fluidRow(
    column(width = 4, …),
    column(width = 2,
      offset = 3, …),
  ),
  fluidRow(
    column(width = 12, …)
  )
))
```



Widgets - The first argument of each widget function is the **<name>** for the widget. You can access a widget's current value in **server.R** with **input$<name>**

widget	function	common arguments
Action button	`actionButton`	inputId, label
checkbox	`checkboxInput`	inputId, label, value
checkbox group	`checkboxGroupInput`	inputId, label, choices, selected
date selector	`dateInput`	inputId, label, value, min, max, format
date range selector	`dateRangeInput`	inputId, label, start, end, min, max, format
file uploader	`fileInput`	inputId, label, multiple
Number field	`numericInput`	inputId, label, value, min, max, step
Radio buttons	`radioButtons`	inputId, label, choices, selected
select box	`selectInput`	inputId, label, choices, selected, multiple
slider	`sliderInput`	inputId, label, min, max, value, step
submit button	`submitButton`	text
text field	`textInput`	inputId, label, value

HTML elements - Add html elements with shiny functions that parallel common HTML tags.

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

6. Run your app

runApp - run from local files
runGitHub - run from files hosted on www.GitHub.com
runGist - run from files saved as a gist (gist.github.com)
runURL - run from files saved at any URL

7. Share your app Launch your app as a live web page that users can visit online.

ShinyApps.io

Host your apps on RStudio's server. Free and paid options
www.shinyapps.io

Shiny Server

Build your own linux server to host apps. Free and open source.
shiny.rstudio.com/deploy

Shiny Server Pro

Build a commercial server with authentication, resource management, and more.
shiny.rstudio.com/deploy

Package Development

with devtools Cheat Sheet

RStudio

Package Structure

A package is a convention for organizing files into directories.

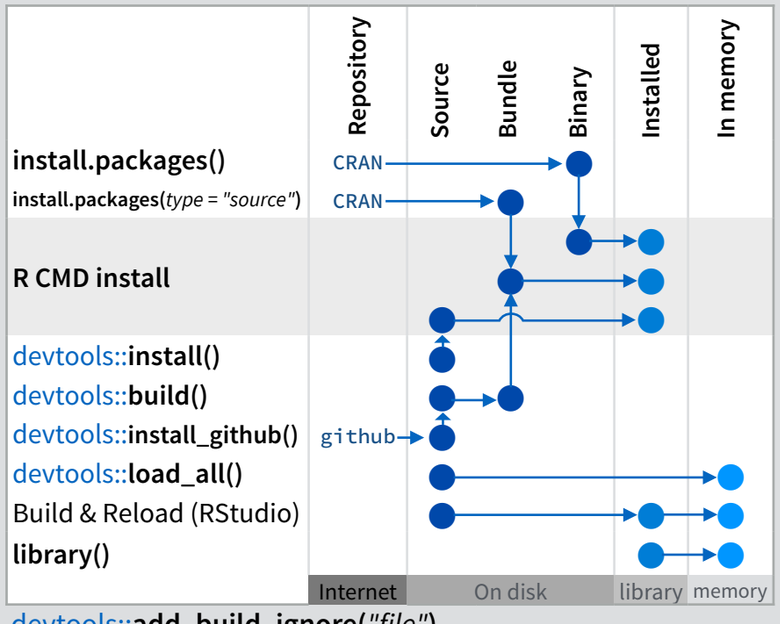
This sheet shows how to work with the 7 most common parts of an R package:

```
Package
  DESCRIPTION   Setup
  R/            Write code
  tests/        Test
  man/          Document
  vignettes/    Teach
  data/         Add data
  NAMESPACE     Organize
```

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::**add_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 and sets up how your package will work with other packages.

- You must have a DESCRIPTION file
- Add the packages that yours relies on with

devtools::**use_package()**

Adds a package to the Imports field (or Suggests field (if second argument is "Suggests").

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

```
Package: mypackage
Title: Title of Package
Version: 0.1.0
Authors@R: person("Hadley", "Wickham", email =
    "hadley@me.com", role = c("aut", "cre"))
Description: What the package does (one paragraph)
Depends: R (>= 3.1.0)
License: GPL-2
LazyData: true
Imports:
    dplyr (>= 0.4.0),
    ggvis (>= 0.2)
Suggests:
    knitr (>= 0.1.0)
```

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

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

Write code (R/)

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

- Create a new package project with

devtools::**create**(*"path/to/name"*)

Create a template to develop into a package.

- Save your code in R/ as scripts (extension .R)

Workflow

1. Edit your code.
2. Load your code with one of

devtools::**load_all()**

Re-loads all *saved* files in R/ into memory.

Ctrl/Cmd + Shift + L (keyboard shortcut)

Saves all open files then calls load_all().

3. Experiment in the console.
4. Repeat.

- Use consistent style with **r-pkgs.had.co.nz/r.html#style**
- Click on a function and press **F2** to open its definition
- Search for a function with **Ctrl + .**

Visit **r-pkgs.had.co.nz** for more

Learn more at http://r-pkgs.had.co.nz • devtools 1.6.1 • Updated: 1/15

info@rstudio.com • 844-448-1212 • rstudio.com

Test (tests/)

Use tests/ to store unit tests that will inform you if your code ever breaks.

- Add a **tests/** directory and import **testthat** with

devtools::**use_testthat()**

Sets up package to use automated tests with testthat

- Write tests with **context()**, **test()**, and expectations
- 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 *saved* in tests/.

Ctrl/Cmd + Shift + T (keyboard shortcut)

3. Repeat until all tests pass

Example test

```
context("Arithmetic")

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

expect_equal()	is equal within small numerical tolerance?
expect_identical()	is exactly equal?
expect_match()	matches specified string or regular expression?
expect_output()	prints specified output?
expect_message()	displays specified message?
expect_warning()	displays specified warning?
expect_error()	throws specified error?
expect_is()	output inherits from certain class?
expect_false()	returns FALSE?
expect_true()	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}
\strong{bold text}
\code{function(args)}
\pkg{package}

\dontrun{code}
\dontshow{code}
\donttest{code}

\deqn{a + b (block)}
\eqn{a + b (inline)}

\email{name@@foo.com}
\href{url}{display}
\url{url}

\link[=dest]{display}
\linkS4class{class}
\code{\link{function}}
\code{\link[package]{function}}

\tabular{lcr}{
  left \tab centered \tab right \cr
  cell \tab cell     \tab cell  \cr
}
```

The **roxygen** package

roxygen lets you write documentation inline in your .R files with a shorthand syntax.

- 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
@describeIn	**@param**	@source data
@examples	@rdname	@include
@export	**@return**	@slot S4
@family	@section	@field RC

Teach (vignettes/)

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

- Create a vignettes/ directory and a template vignette with

  devtools::**use_vignette()**
  Adds template vignette as vignettes/my-vignette.Rmd.

- Append YAML headers to your vignettes (like right)
- Write the body of your vignettes in R Markdown (rmarkdown.rstudio.com)

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

Add data (data/)

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

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

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

devtools::**use_data_raw()**
Adds an R Script used to clean a data set to data-raw/. Includes data-raw/ on .Rbuildignore.

Store data in

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

Organize (NAMESPACE)

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

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

Workflow

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

Submit your package

r-pkgs.had.co.nz/release.html