Data Wrangling
with dplyr and tidyr
Cheat Sheet

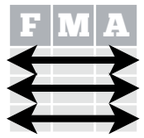
RStudio

Tidy Data - A foundation for wrangling in R

In a tidy data set:

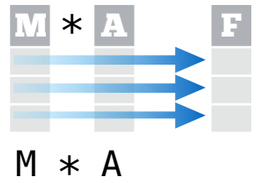
Each **variable** is saved in its own **column**

&



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]

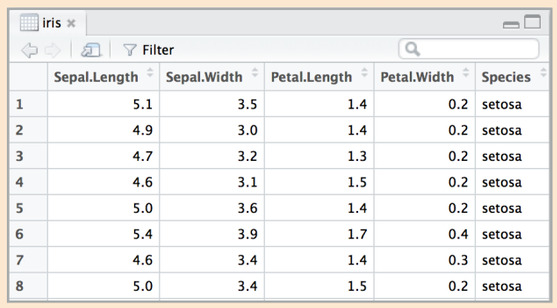
   Sepal.Length Sepal.Width Petal.Length
1           5.1         3.5          1.4
2           4.9         3.0          1.4
3           4.7         3.2          1.3
4           4.6         3.1          1.5
5           5.0         3.6          1.4
..          ...         ...          ...
Variables not shown: 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).



	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.

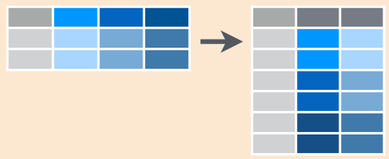
`x %>% f(y)` *is the same as* `f(x, y)`

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

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

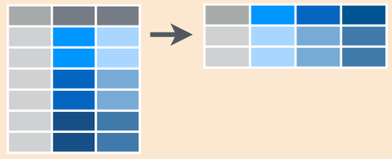
```
iris %>%
  group_by(Species) %>%
  summarise(avg = mean(Sepal.Width)) %>%
  arrange(avg)
```

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.

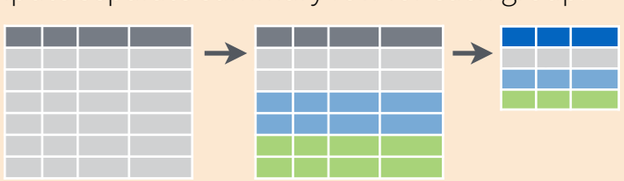
sd
Standard deviation of a vector.

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`

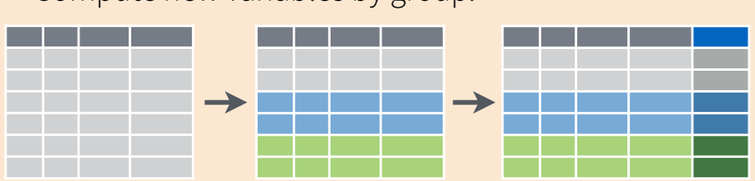
cummin
Cumulative `min`

cumprod
Cumulative `prod`

pmax
Element-wise `max`

pmin
Element-wise `min`

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



Combine Data Sets

a

x1	x2
A	1
B	2
C	3

+

b

x1	x3
A	T
B	F
D	T

=

Mutating Joins

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

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

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

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

x1	x2	x3
A	1	T
B	2	F

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

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

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

Filtering Joins

x1	x2
A	1
B	2

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

x1	x2
C	3

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

y

x1	x2
A	1
B	2
C	3

+

z

x1	x2
B	2
C	3
D	4

=

Set Operations

x1	x2
B	2
C	3

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

x1	x2
A	1
B	2
C	3
D	4

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

x1	x2
A	1

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

Binding

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

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

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

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

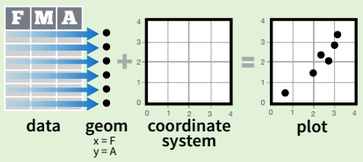
Data Visualization with ggplot2

Cheat Sheet

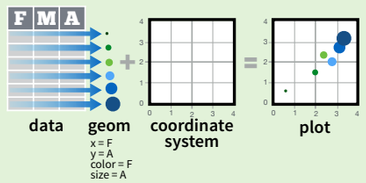
RStudio

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().

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, xend, 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, xend, 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

One Variable

Continuous

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

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

c + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, group, linetype, size, weight

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

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

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

Discrete

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

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

Two Variables

Continuous X, Continuous Y

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

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

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

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

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

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

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

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

Discrete X, Continuous Y

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

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

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

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

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

Discrete X, Discrete Y

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

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

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))
```

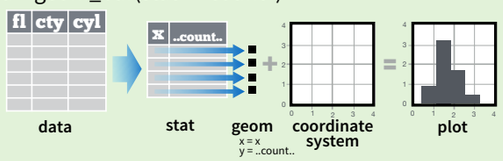
l + geom_contour(aes(z = z)**)**
x, y, z, alpha, colour, group, linetype, size, weight

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

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

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 = "count")**



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

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

stat function · layer mappings

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

variable created by transformation

geom for layer · parameters for stat

1D distributions

c + stat_bin(binwidth = 1, origin = 10**)**
x, y | ..count.., ..ncount.., ..density.., ..ndensity..

c + stat_count(width = 1**)**
x, y, | ..count.., ..prop..

c + stat_density(adjust = 1, kernel = "gaussian"**)**
x, y, | ..count.., ..density.., ..scaled..

2D distributions

e + stat_bin_2d(bins = 30, drop = TRUE**)**
x, y, fill | ..count.., ..density..

e + stat_bin_hex(bins = 30**)**
x, y, fill | ..count.., ..density..

e + stat_density_2d(contour = TRUE, n = 100**)**
x, y, color, size | ..level..

e + stat_ellipse(level = 0.95, segments = 51, type = "t"**)**

3 Variables

l + stat_contour(aes(z = z)**)**
x, y, z, order | ..level..

l + stat_summary_hex(aes(z = z), bins = 30, fun = mean**)**
x, y, z, fill | ..value..

l + stat_summary_2d(aes(z = z), bins = 30, fun = mean**)**
x, y, z, fill | ..value..

Comparisons

f + stat_boxplot(coef = 1.5**)**
x, y | ..lower.., ..middle.., ..upper.., ..width.. , ..ymin.., ..ymax..

f + stat_ydensity(adjust = 1, kernel = "gaussian", scale = "area"**)**
x, y | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

Functions

e + stat_ecdf(n = 40**)**
x, y | ..x.., ..y..

e + stat_quantile(quantiles = c(0.25, 0.5, 0.75), formula = y ~ log(x), method = "rq"**)**
x, y | ..quantile..

e + stat_smooth(method = "auto", formula = y ~ x, se = TRUE, n = 80, fullrange = FALSE, level = 0.95**)**
x, y | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

General Purpose

ggplot() + stat_function(aes(x = -3:3), fun = dnorm, n = 101, args = list(sd=0.5)**)**
x | ..x.., ..y..

e + stat_identity(na.rm = TRUE**)**

ggplot() + stat_qq(aes(sample=1:100), distribution = qt, dparams = list(df=5)**)**
sample, x, y | ..sample.., ..theoretical..

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

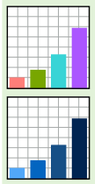
e + stat_summary(fun.data = "mean_cl_boot"**)**

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

e + stat_unique()

Scales

Scales 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(date_labels = "%m/%d"**)**, 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

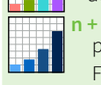
scale_x_reverse() - Reverse direction of x axis

scale_x_sqrt() - Plot x on square root scale

Color and fill scales

Discrete

n <- d + 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 <- c + 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()

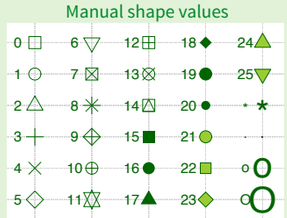
Shape scales

p <- e + geom_point(aes(shape = fl, size = cyl)**)**

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

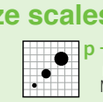
p + scale_shape_manual(values = c(3:7)**)**
Shape values shown in chart on right

Manual shape values



Size scales

p + scale_radius(range=c(1,6)**)**
p + scale_size()



p + scale_size_area(max_scale = 6**)**
Maps to area of circle (not radius)

Coordinate Systems

r <- d + geom_bar()

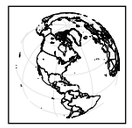
r + coord_cartesian(xlim = c(0, 5)**)**
xlim, ylim
The default cartesian coordinate system

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.



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

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

Set **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 ^ .(fl)**))**

α^c	α^d	α^e	α^p	α^r

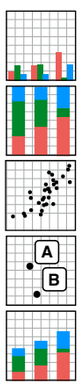
t + facet_grid(. ~ fl, labeller = label_parsed)

c	d	e	p	r

Position Adjustments

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

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



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

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

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

e + geom_label(position = "nudge")
Nudge labels away from points

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

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

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

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

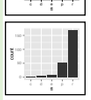
n + theme(legend.position = "bottom"**)**
Place legend at "bottom", "top", "left", or "right"

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

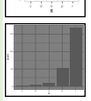
n + scale_fill_discrete(name = "Title", labels = c("A", "B", "C", "D", "E"**)**
Set legend title and labels with a scale function.

Themes

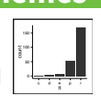
r + theme_bw()
White background with grid lines



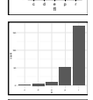
r + theme_gray()
Grey background (default theme)



r + theme_dark()
dark for contrast

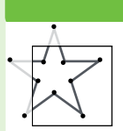


r + theme_classic()
r + theme_light()
r + theme_linedraw()
r + theme_minimal()
Minimal themes



r + theme_void()
Empty theme

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)**)**

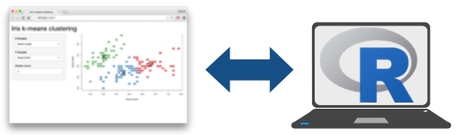
Interactive Web Apps with shiny Cheat Sheet

learn more at shiny.rstudio.com

RStudio

Basics

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



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

App template

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

```
library(shiny)
ui <- fluidPage()
server <- function(input, output){}
shinyApp(ui = ui, server = server)
```

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

Share your app

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

1. Create a free or professional account at **http://shinyapps.io**
2. Click the **Publish** icon in the RStudio IDE (>=0.99) or run:
   rsconnect::deployApp("<path to directory>"**)**

Build or purchase your own Shiny Server

at www.rstudio.com/products/shiny-server/

Building an App - Complete the template by adding arguments to fluidPage() and a body to the server function.

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

Add outputs with ***Output()** functions

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

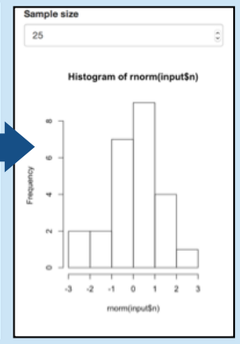
1. Refer to outputs with **output$<id>**
2. Refer to inputs with **input$<id>**
3. Wrap code in a **render*()** function before saving to output

```
library(shiny)

ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)

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

shinyApp(ui = ui, server = server)
```



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

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

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

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

ui.R contains everything you would save to ui.

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

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

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

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

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

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

	works with	
DT::**renderDataTable**(expr, options, callback, escape, env, quoted)		**dataTableOutput**(outputId, icon, …)
renderImage(expr, env, quoted, deleteFile)		**imageOutput**(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)
renderPlot(expr, width, height, res, …, env, quoted, func)		**plotOutput**(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)
renderPrint(expr, env, quoted, func, width)		**verbatimTextOutput**(outputId)
renderTable(expr,…, env, quoted, func)		**tableOutput**(outputId)
renderText(expr, env, quoted, func)		**textOutput**(outputId, container, inline)
renderUI(expr, env, quoted, func)		**uiOutput**(outputId, inline, container, …) & **htmlOutput**(outputId, inline, container, …)

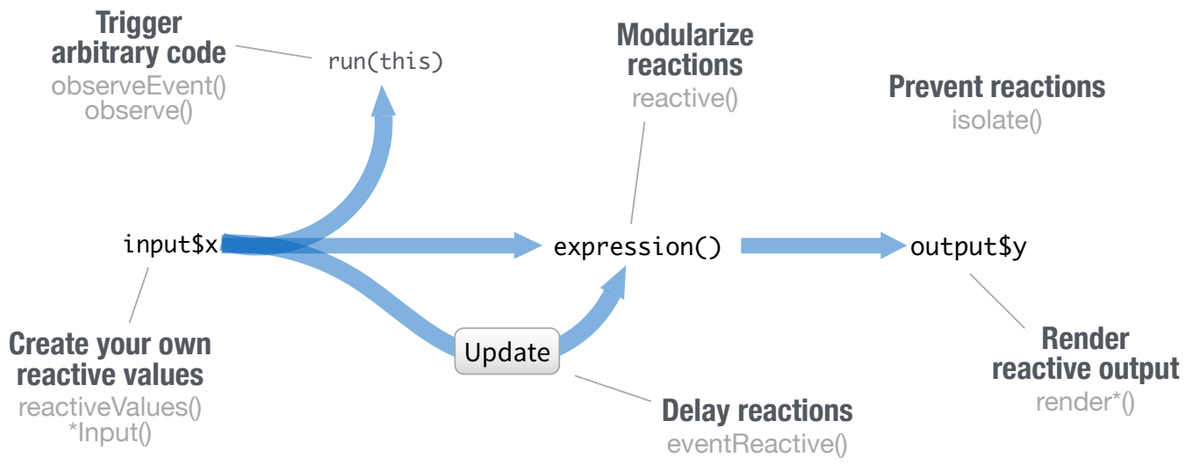
Inputs - collect values from the user

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

Action	**actionButton**(inputId, label, icon, …)
Link	**actionLink**(inputId, label, icon, …)
☑ Choice 1 ☑ Choice 2 ☐ Choice 3	**checkboxGroupInput**(inputId, label, choices, selected, inline)
☑ Check me	**checkboxInput**(inputId, label, value)
	dateInput(inputId, label, value, min, max, format, startview, weekstart, language)
	dateRangeInput(inputId, label, start, end, min, max, format, startview, weekstart, language, separator)
Choose File	**fileInput**(inputId, label, multiple, accept)
1	**numericInput**(inputId, label, value, min, max, step)
•••••••	**passwordInput**(inputId, label, value)
◉ Choice A ○ Choice B ○ Choice C	**radioButtons**(inputId, label, choices, selected, inline)
Choice 1 (Choice 1, Choice 2)	**selectInput**(inputId, label, choices, selected, multiple, selectize, width, size) (also **selectizeInput()**)
0 5 10	**sliderInput**(inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post)
Apply Changes	**submitButton**(text, icon) (Prevents reactions across entire app)
Enter text	**textInput**(inputId, label, value)

Reactivity

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



Create your own reactive values

```
library(shiny)

ui <- fluidPage(
  textInput("a","")
)

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

shinyApp(ui, server)
```

*Input() functions

(see front page)

reactiveValues(...)

Each input function creates a reactive value stored as **input$<inputId>**

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

Render reactive output

```
library(shiny)

ui <- fluidPage(
  textInput("a","")
)

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

shinyApp(ui, server)
```

render*() functions

(see front page)

Builds an object to display. Will rerun code in body to rebuild the object whenever a reactive value in the code changes.

Save the results to **output$<outputId>**

Prevent reactions

```
library(shiny)

ui <- fluidPage(
  textInput("a",""),
  textOutput("b")
)

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

shinyApp(ui, server)
```

isolate(expr)

Runs a code block. Returns a **non-reactive** copy of the results.

Trigger arbitrary code

```
library(shiny)

ui <- fluidPage(
  textInput("a",""),
  actionButton("go", "")
)

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

shinyApp(ui, server)
```

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

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

Modularize reactions

```
library(shiny)

ui <- fluidPage(
  textInput("a",""),
  textInput("z", "")
)

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

shinyApp(ui, server)
```

reactive(x, env, quoted, label, domain)

Creates a **reactive expression** that

- **caches** its value to reduce computation
- can be called by other code
- notifies its dependencies when it ha been invalidated

Call the expression with function syntax, e.g. `re()`

Delay reactions

```
library(shiny)

ui <- fluidPage(
  textInput("a",""),
  actionButton("go", "")
)

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

shinyApp(ui, server)
```

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

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

UI

An app's UI is an HTML document. Use Shiny's functions to assemble this HTML with R.

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

Returns HTML

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

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

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

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

To include a CSS file, use **includeCSS()**, or

1. Place the file in the **www** subdirectory
2. Link to it with

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

To include JavaScript, use **includeScript()** or

1. Place the file in the **www** subdirectory
2. Link to it with

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

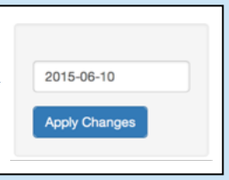
To include an image

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

Layouts

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

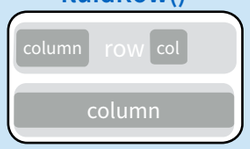
```
wellPanel(
  dateInput("a", ""),
  submitButton()
)
```



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

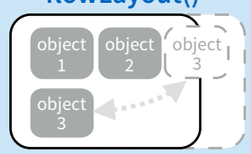
Organize panels and elements into a layout with a layout function. Add elements as arguments of the layout functions.

fluidRow()




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

flowLayout()

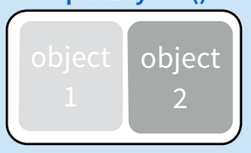



```
ui <- fluidPage(
  flowLayout( # object 1,
              # object 2,
              # object 3
  )
)
```

sidebarLayout()

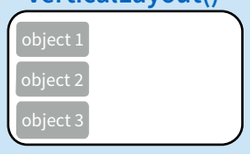

```
ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(),
    mainPanel()
  )
)
```

splitLayout()




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

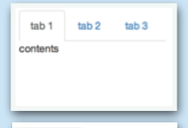
verticalLayout()




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

Layer tabPanels on top of each other, and navigate between them, with:

```
ui <- fluidPage( tabsetPanel(
    tabPanel("tab 1", "contents"),
    tabPanel("tab 2", "contents"),
    tabPanel("tab 3", "contents")))
```

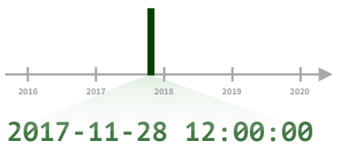



```
ui <- fluidPage( navlistPanel(
    tabPanel("tab 1", "contents"),
    tabPanel("tab 2", "contents"),
    tabPanel("tab 3", "contents")))
```


```
ui <- navbarPage(title = "Page",
    tabPanel("tab 1", "contents"),
    tabPanel("tab 2", "contents"),
    tabPanel("tab 3", "contents"))
```

Dates and times with lubridate : : **CHEAT SHEET**

Date-times



2017-11-28 12:00:00
A **date-time** is a point on the timeline, stored as the number of seconds since 1970-01-01 00:00:00 UTC

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

2017-11-28
A **date** is a day stored as the number of days since 1970-01-01

d <- ***as_date****(17498)*
"2017-11-28"

12:00:00
An hms is a **time** stored as the number of seconds since 00:00:00

*t <- hms::****as.hms****(85)*
00:01:25

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

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

2017-11-28T14:02:00	**ymd_hms**(), **ymd_hm**(), **ymd_h**(). *ymd_hms("2017-11-28T14:02:00")*
2017-22-12 10:00:00	**ydm_hms**(), **ydm_hm**(), **ydm_h**(). *ydm_hms("2017-22-12 10:00:00")*
11/28/2017 1:02:03	**mdy_hms**(), **mdy_hm**(), **mdy_h**(). *mdy_hms("11/28/2017 1:02:03")*
1 Jan 2017 23:59:59	**dmy_hms**(), **dmy_hm**(), **dmy_h**(). *dmy_hms("1 Jan 2017 23:59:59")*
20170131	**ymd**(), **ydm**(). *ymd(20170131)*
July 4th, 2000	**mdy**(), **myd**(). *mdy("July 4th, 2000")*
4th of July '99	**dmy**(), **dym**(). *dmy("4th of July '99")*
2001: Q3	**yq**() Q for quarter. *yq("2001: Q3")*
2:01	hms::**hms**() Also lubridate::**hms**(), **hm**() and **ms**(), which return periods.* *hms::hms(sec = 0, min= 1, hours = 2)*
2017.5	**date_decimal**(decimal, tz = "UTC") *date_decimal(2017.5)*
	now(tzone = "") Current time in tz (defaults to system tz). *now()*
	today(tzone = "") Current date in a tz (defaults to system tz). *today()*
	fast_strptime() Faster strptime. *fast_strptime('9/1/01', '%y/%m/%d')*
	parse_date_time() Easier strptime. *parse_date_time("9/1/01", "ymd")*

GET AND SET COMPONENTS

Use an accessor function to get a component.

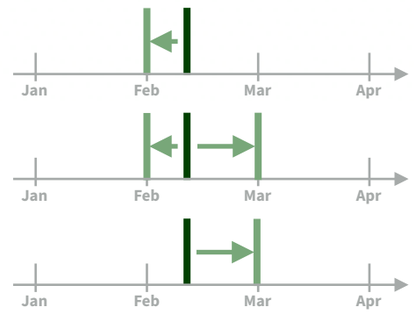
d ## "2017-11-28"
day(d) ## 28

Assign into an accessor function to change a component in place.

day(d) <- 1
d ## "2017-11-01"

2018-01-31 11:59:59	**date**(x) Date component. *date(dt)*
2018-01-31 11:59:59	**year**(x) Year. *year(dt)* **isoyear**(x) The ISO 8601 year. **epiyear**(x) Epidemiological year.
2018-01-31 11:59:59	**month**(x, label, abbr) Month. *month(dt)*
2018-01-31 11:59:59	**day**(x) Day of month. *day(dt)* **wday**(x,label,abbr) Day of week. **qday**(x) Day of quarter.
2018-01-31 11:59:59	**hour**(x) Hour. *hour(dt)*
2018-01-31 11:59:59	**minute**(x) Minutes. *minute(dt)*
2018-01-31 11:59:59	**second**(x) Seconds. *second(dt)*
	week(x) Week of the year. *week(dt)* **isoweek**() ISO 8601 week. **epiweek**() Epidemiological week.
	quarter(x, with_year = FALSE) Quarter. *quarter(dt)*
	semester(x, with_year = FALSE) Semester. *semester(dt)*
	am(x) Is it in the am? *am(dt)* **pm**(x) Is it in the pm? *pm(dt)*
	dst(x) Is it daylight savings? *dst(d)*
	leap_year(x) Is it a leap year? *leap_year(d)*
	update(object, ..., simple = FALSE) *update(dt, mday = 2, hour = 1)*

Round Date-times



floor_date(x, unit = "second") Round down to nearest unit. *floor_date(dt, unit = "month")*

round_date(x, unit = "second") Round to nearest unit. *round_date(dt, unit = "month")*

ceiling_date(x, unit = "second", change_on_boundary = NULL) Round up to nearest unit. *ceiling_date(dt, unit = "month")*

rollback(dates, roll_to_first = FALSE, preserve_hms = TRUE) Roll back to last day of previous month. *rollback(dt)*

Stamp Date-times

stamp() Derive a template from an example string and return a new function that will apply the template to date-times. Also **stamp_date**() and **stamp_time**().

1. Derive a template, create a function
   sf <- stamp("Created Sunday, Jan 17, 1999 3:34")
2. Apply the template to dates
   sf(ymd("2010-04-05"))
   ## [1] "Created Monday, Apr 05, 2010 00:00"

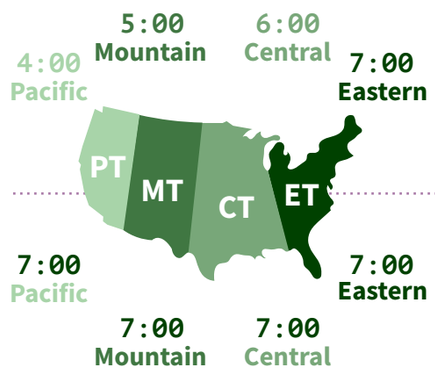
Tip: use a date with day > 12

Time Zones

R recognizes ~600 time zones. Each encodes the time zone, Daylight Savings Time, and historical calendar variations for an area. R assigns *one* time zone per vector.

Use the **UTC** time zone to avoid Daylight Savings.

OlsonNames() Returns a list of valid time zone names. *OlsonNames()*



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

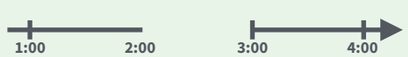
force_tz(time, tzone = "") Get the **same clock time** in a new time zone (a new date-time). *force_tz(dt, "US/Pacific")*

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

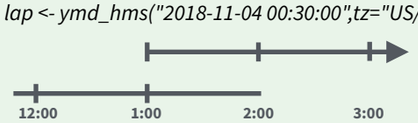
Math with date-times relies on the **timeline**, which behaves inconsistently. Consider how the timeline behaves during:

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

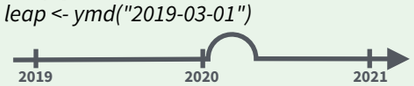
The start of daylight savings (spring forward)
gap <- ymd_hms("2018-03-11 01:30:00",tz="US/Eastern")



The end of daylight savings (fall back)
lap <- ymd_hms("2018-11-04 00:30:00",tz="US/Eastern")

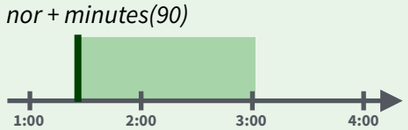


Leap years and leap seconds
leap <- ymd("2019-03-01")

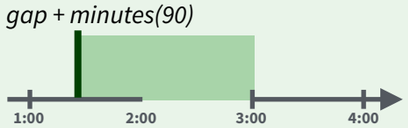


Periods track changes in clock times, which ignore time line irregularities.

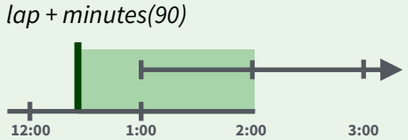
nor + minutes(90)



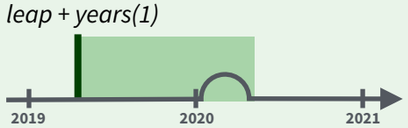
gap + minutes(90)



lap + minutes(90)

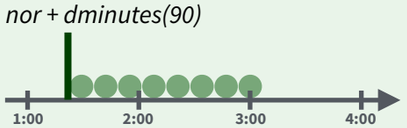


leap + years(1)

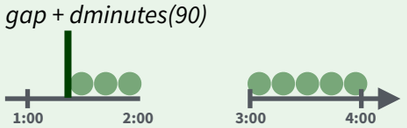


Durations track the passage of physical time, which deviates from clock time when irregularities occur.

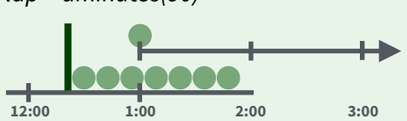
nor + dminutes(90)



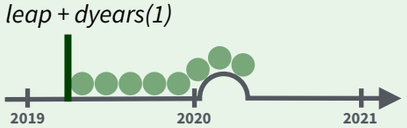
gap + dminutes(90)



lap + dminutes(90)

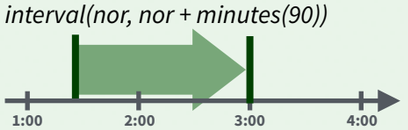


leap + dyears(1)

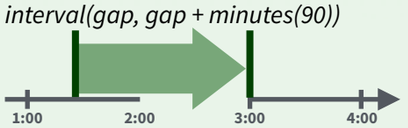


Intervals represent specific intervals of the timeline, bounded by start and end date-times.

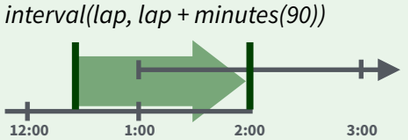
interval(nor, nor + minutes(90))



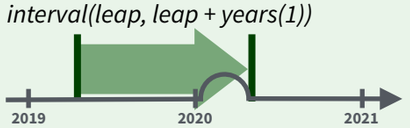
interval(gap, gap + minutes(90))



interval(lap, lap + minutes(90))



interval(leap, leap + years(1))



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

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

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

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

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

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

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

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

PERIODS

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

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

```
p <- months(3) + days(12)
p
"3m 12d 0H 0M 0S"
```

Number of months | Number of days | etc.

years(x = 1) x years.
months(x) x months.
weeks(x = 1) x weeks.
days(x = 1) x days.
hours(x = 1) x hours.
minutes(x = 1) x minutes.
seconds(x = 1) x seconds.
milliseconds(x = 1) x milliseconds.
microseconds(x = 1) x microseconds
nanoseconds(x = 1) x nanoseconds.
picoseconds(x = 1) x picoseconds.

period(num = NULL, units = "second", ...)
An automation friendly period constructor.
period(5, unit = "years")

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

period_to_seconds(x) Convert a period to the "standard" number of seconds implied by the period. Also **seconds_to_period**(). *period_to_seconds(p)*

DURATIONS

Add or subtract durations to model physical processes, like battery life. Durations are stored as seconds, the only time unit with a consistent length. **Difftimes** are a class of durations found in base R.

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

```
dd <- ddays(14)
dd
"1209600s (~2 weeks)"
```

Exact length in seconds | Equivalent in common units

dyears(x = 1) 31536000x seconds.
dweeks(x = 1) 604800x seconds.
ddays(x = 1) 86400x seconds.
dhours(x = 1) 3600x seconds.
dminutes(x = 1) 60x seconds.
dseconds(x = 1) x seconds.
dmilliseconds(x = 1) x $\times 10^{-3}$ seconds.
dmicroseconds(x = 1) x $\times 10^{-6}$ seconds.
dnanoseconds(x = 1) x $\times 10^{-9}$ seconds.
dpicoseconds(x = 1) x $\times 10^{-12}$ seconds.

duration(num = NULL, units = "second", ...)
An automation friendly duration constructor. *duration(5, unit = "years")*

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

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

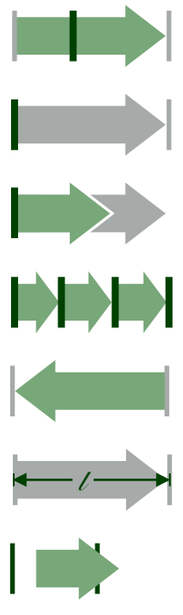
INTERVALS

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

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

Start Date | End Date

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



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

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

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

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

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

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

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

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

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`

```
> set.seed(45L)
> DT <- data.table(V1=c(1L,2L),
                   V2=LETTERS[1:3],
                   V3=round(rnorm(4),4),
                   V4=1:12)
```

Create a `data.table` and call it `DT`

Subsetting Rows Using `i`

```
> DT[3:5,]
> DT[3:5]
> DT[V2=="A"]
> DT[V2 %in% c("A","C")]
```

Select 3rd to 5th row
Select 3rd to 5th row
Select all rows that have value `A` in column `V2`
Select all rows that have value `A` or `C` in column `V2`

Manipulating on Columns in `j`

```
> 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)]
```

Return `V2` as a vector

Return `V2` and `V3` as a `data.table`
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 `data.table`

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

```
> 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]
```

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 `sign(V1-1)`

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`

Indexing And Keys

```
> setkey(DT,V2)
> DT["A"]
   V1 V2      V3 V4
1:  1  A -0.2392  1
2:  2  A -1.6148  4
3:  1  A  1.0498  7
4:  2  A  0.3262 10
> DT[c("A","C")]
> DT["A",mult="first"]

> DT["A",mult="last"]

> DT[c("A","D")]
   V1 V2      V3 V4
1:  1  A -0.2392  1
2:  2  A -1.6148  4
3:  1  A  1.0498  7
4:  2  A  0.3262 10
5: NA  D      NA NA
> DT[c("A","D"),nomatch=0]
   V1 V2      V3 V4
1:  1  A -0.2392  1
2:  2  A -1.6148  4
3:  1  A  1.0498  7
4:  2  A  0.3262 10
> DT[c("A","C"),sum(V4)]

> DT[c("A","C"),
     sum(V4),
     by=.EACHI]
   V2 V1
1:  A 22
2:  C 30
> setkey(DT,V1,V2)
> DT[.(2,"C")]
   V1 V2      V3 V4
1:  2  C  0.3262  6
2:  2  C -1.6148 12
> DT[.(2,c("A","C"))]
   V1 V2      V3 V4
1:  2  A -1.6148  4
2:  2  A  0.3262 10
3:  2  C  0.3262  6
4:  2  C -1.6148 12
```

A key is set on `V2`; output is returned invisibly
Return all rows where the key column (set to `V2`) has the value `A`

Return all rows where the key column (`V2`) has value `A` or `C`
Return first row of all rows that match value `A` in key column `V2`
Return last row of all rows that match value `A` in key column `V2`
Return all rows where key column `V2` has value `A` or `D`

Return all rows where key column `V2` has value `A` or `D`

Return total sum of `V4`, for rows of key column `V2` that have values `A` or `C`
Return sum of column `V4` for rows of `V2` that have value `A`, and anohter sum for rows of `V2` that have value `C`

Sort by `V1` and then by `V2` within each group of `V1` (invisible)
Select rows that have value 2 for the first key (`V1`) and the value `C` for the second key (`V2`)

Select rows that have value 2 for the first key (`V1`) and within those rows the value `A` or `C` for the second key (`V2`)

Advanced Data Table Operations

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

R Markdown Cheat Sheet

learn more at rmarkdown.rstudio.com

RStudio

.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

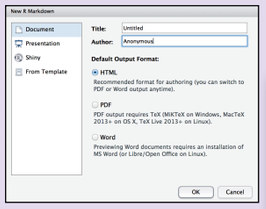
At the click of a button, or the type of a command, you can rerun the code in an R Markdown file to reproduce your work and export the results as a finished report.

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 at File ▸ New File ▸ R Markdown. Use the wizard that opens to pre-populate the file with a template



2 Write document by editing template

3 Knit document to create report Use knit button or **render()** to knit

4 Preview Output in IDE window

5 Publish (optional) to web or server

Synch publish button to accounts at
- rpubs.com,
- shinyapps.io
- RStudio Connect

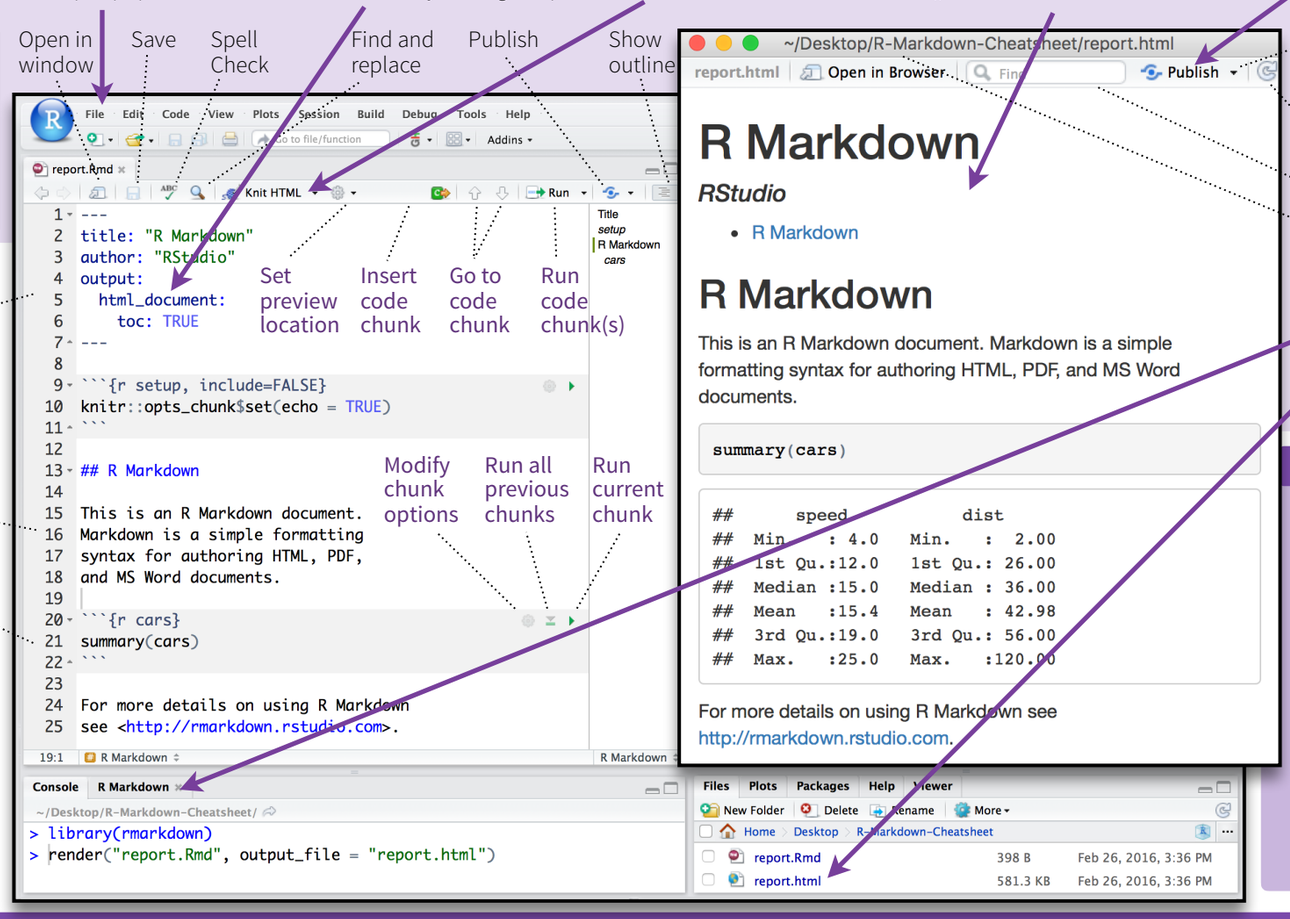
Reload document

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



.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**

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

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

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

renderTable({
  head(cars, input$n)
})
```
```

How many cars? 5

	speed	dist
1	4.00	2.00
2	4.00	10.00
3	7.00	4.00
4	7.00	22.00
5	8.00	16.00

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

** Your report will 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

Code chunks

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

```
```{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)
```
```

Important chunk options

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

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

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

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

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

dependson - chunk dependencies for caching (default = NULL)

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

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

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

eval - Run code in chunk (default = TRUE)

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

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

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

highlight - highlight source code (default = TRUE)

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

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

results (default = 'markup')
'asis' - passthrough results
'hide' - do not display results
'hold' - put all results below all code

tidy - tidy code for display (default = FALSE)

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

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

Parameters

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

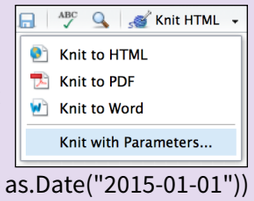
1 Add parameters Create and set parameters in the header as sub-values of **params**

```
---
params:
  n: 100
  d: !r Sys.Date()
---
```
Indent 2 spaces

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

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

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



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

Pandoc's Markdown

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

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

$$E = mc^{2}$$

> block quote

# Header1  {#anchor}

## Header 2  {#css_id}

### Header 3  {.css_class}

#### Header 4

##### Header 5

###### Header 6

<!--Text comment-->

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

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

![Caption](smallorb.png)

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

* item 2

    Continued (indent 4 spaces)

1. ordered list
2. item 2
        i) sub-item 1
          A.  sub-sub-item 1

(@)  A list whose numbering
continues after
(@)  an interruption
Term 1

:   Definition 1

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

- slide bullet 1
- slide bullet 2

(>- to have bullets appear on click)
horizontal rule/slide break:

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

Plain text
End a line with two spaces to start a new paragraph.
italics and **bold**
`verbatim code`
sub/superscript$^2_2$
~~strikethrough~~
escaped: * _ \
endash: –, emdash: —
equation: $A = \pi * r^2$
equation block:

$$E = mc^2$$

> block quote

Header1

Header 2

Header 3

Header 4

Header 5

Header 6

HTML ignored in pdfs

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

Caption

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

    Continued (indent 4 spaces)

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

1. A list whose numbering

continues after

2. an interruption

Term 1

Definition 1

Right	Left	Default	Center
12	12	12	12
123	123	123	123
1	1	1	1

- slide bullet 1
- slide bullet 2

(>- to have bullets appear on click)

horizontal rule/slide break:

A footnote [1]

1. Here is the footnote.

Set render options with YAML

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

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

```
---
output: html_document
---

# Body
```

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

Customize output with sub-options (listed at right):

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

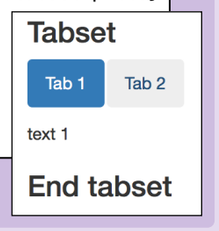
# Body
```

Indent 2 spaces · Indent 4 spaces

html tabsets

Use .tabset css class to place sub-headers into tabs

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



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

Options not listed: extra_dependencies, fig_crop, fig_retina, font_adjustment, font_theme, footer, logo, html_preview, reference_odt, transition, variant, widescreen

Create a Reusable template

1. **Create a new package** with a inst/rmarkdown/templates directory
2. In the directory, **Place a folder** that contains:
    - **template.yaml** (see below)
    - **skeleton.Rmd** (contents of the template)
    - any supporting files
3. **Install the package**
4. **Access template** in wizard at File ▶ New File ▶ R Markdown

template.yaml

```
---
name: My Template
---
```

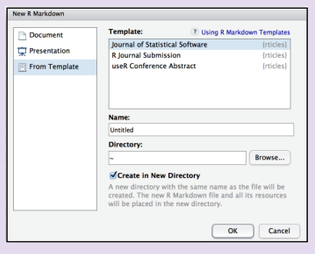


Table suggestions

Several functions format R data into tables

Table with kable

eruptions	waiting
3.600	79
1.800	54
3.333	74
2.283	62

	eruptions	waiting
1	3.60	79.00
2	1.80	54.00
3	3.33	74.00
4	2.28	62.00

Table with xtable

Table with stargazer

	eruptions	waiting
1	3.600	79
2	1.800	54
3	3.333	74
4	2.283	62

data <- faithful[1:4,]

```
```{r results = 'asis'}
knitr::kable(data, caption = "Table with kable")
```
```

```
```{r results = "asis"}
print(xtable::xtable(data, caption = "Table with xtable"),
  type = "html", html.table.attributes = "border=0"))
```
```

```
```{r results = "asis"}
stargazer::stargazer(data, type = "html",
  title = "Table with stargazer")
```
```

Learn more in the **stargazer, xtable,** and **knitr** packages.

Citations and Bibliographies

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

1. **Set bibliography file** and CSL 1.0 Style file (optional) in the YAML header

```
---
bibliography: refs.bib
csl: style.csl
---
```

2. **Use citation keys in text**

```
Smith cited [@smith04].
Smith cited without author [-@smith04].
@smith04 cited in line.
```

3. **Render.** Bibliography will be added to end of document

Smith cited (Joe Smith 2004).
Smith cited without author (2004).
Joe Smith (2004) cited in line.

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

Creating Vectors

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

Vector Functions

sort(x)
Return x sorted.

rev(x)
Return x reversed.

table(x)
See counts of values.

unique(x)
See unique values.

Selecting Vector Elements

By Position

`x[4]`	The fourth element.
`x[-4]`	All but the fourth.
`x[2:4]`	Elements two to four.
`x[-(2:4)]`	All elements except two to four.
`x[c(1, 5)]`	Elements one and five.

By Value

`x[x == 10]`	Elements which are equal to 10.
`x[x < 0]`	All elements less than zero.
`x[x %in% c(1, 2, 5)]`	Elements in the set 1, 2, 5.

Named Vectors

`x['apple']`	Element with name 'apple'.

Programming

For Loop

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

Example

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

While Loop

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

Example

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

If Statements

```
if (condition){
   Do something
} else {
   Do something different
}
```

Example

```
if (i > 3){
   print('Yes')
} else {
   print('No')
}
```

Functions

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

Example

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

Reading and Writing Data

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

Conditions

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

Types

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

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

Maths Functions

`log(x)`	Natural log.	`sum(x)`	Sum.
`exp(x)`	Exponential.	`mean(x)`	Mean.
`max(x)`	Largest element.	`median(x)`	Median.
`min(x)`	Smallest element.	`quantile(x)`	Percentage quantiles.
`round(x, n)`	Round to n decimal places.	`rank(x)`	Rank of elements.
`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.

`m[2, ]` - Select a row

`m[ , 1]` - Select a column

`m[2, 3]` - Select an element

`t(m)`
Transpose

`m %*% n`
Matrix Multiplication

`solve(m, n)`
Find x in: m * x = n

Lists

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

`l[[2]]`	`l[1]`	`l$x`	`l['y']`
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

List subsetting

`df$x`

`df[[2]]`

Understanding a data frame

`View(df)`	See the full data frame.
`head(df)`	See the first 6 rows.

Matrix subsetting

`df[ , 2]`

`df[2, ]`

`df[2, 2]`

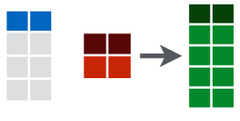
`nrow(df)`
Number of rows.

`ncol(df)`
Number of columns.

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

`cbind` - Bind columns.

`rbind` - Bind rows.



Strings

Also see the **stringr** library.

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

Factors

`factor(x)`
Turn a vector into a factor. Can set the levels of the factor and the order.

`cut(x, breaks = 4)`
Turn a numeric vector into a factor but 'cutting' into sections.

Statistics

`lm(x ~ y, data=df)`
Linear model.

`glm(x ~ y, data=df)`
Generalised linear model.

`summary`
Get more detailed information out a model.

`t.test(x, y)`
Preform a t-test for difference between means.

`pairwise.t.test`
Preform a t-test for paired data.

`prop.test`
Test for a difference between proportions.

`aov`
Analysis of variance.

Distributions

	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	`rnorm`	`dnorm`	`pnorm`	`qnorm`
Poison	`rpois`	`dpois`	`ppois`	`qpois`
Binomial	`rbinom`	`dbinom`	`pbinom`	`qbinom`
Uniform	`runif`	`dunif`	`punif`	`qunif`

Plotting

Also see the **ggplot2** library.

`plot(x)`
Values of x in order.

`plot(x, y)`
Values of x against y.

`hist(x)`
Histogram of x.

Dates

See the **lubridate** library.