Data visualization with ggplot2 : : **CHEATSHEET**
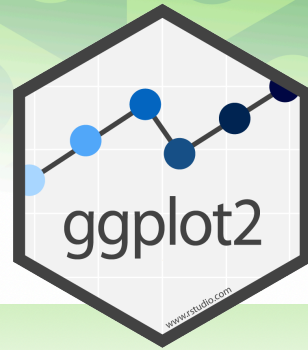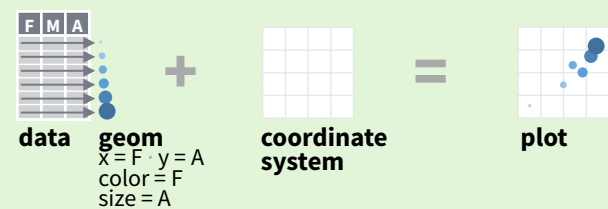
ggplot2

Basics

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

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.

Complete the template below to build a graph.

ggplot (data = <DATA>) +
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),
  stat = <STAT>, position = <POSITION>) +
<COORDINATE_FUNCTION> +
<FACET_FUNCTION> +
<SCALE_FUNCTION> +
<THEME_FUNCTION>

required

Not required, sensible defaults supplied

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

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.

Aes   Common aesthetic values.

color and **fill** - string ("red", "#RRGGBB")

linetype - integer or string (0 = "blank", 1 = "solid", 2 = "dashed", 3 = "dotted", 4 = "dotdash", 5 = "longdash", 6 = "twodash")

size - integer (in mm for size of points and text)

linewidth - integer (in mm for widths of lines)

shape - integer/shape name or a single character ("a")

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

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

a + geom_blank() and **a + expand_limits()** Ensure limits include values across all plots.

b + geom_curve(aes(yend = lat + 1, xend = long + 1), curvature = 1**)** - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

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

a + geom_polygon(aes(alpha = 50)**)** - x, y, alpha, color, fill, group, subgroup, linetype, size

b + geom_rect(aes(xmin = long, ymin = lat, xmax = long + 1, ymax = lat + 1)**)** - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

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

LINE SEGMENTS
common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept = 0, slope = 1)**)**
b + geom_hline(aes(yintercept = lat)**)**
b + geom_vline(aes(xintercept = long)**)**
b + geom_segment(aes(yend = lat + 1, xend = long + 1)**)**
b + geom_spoke(aes(angle = 1:1155, radius = 1)**)**

ONE VARIABLE   continuous
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

c + geom_area(stat = "bin"**)** x, y, alpha, color, fill, linetype, size

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

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

c + geom_freqpoly() x, y, alpha, color, group, linetype, size

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

c2 + geom_qq(aes(sample = hwy)**)** x, y, alpha, color, fill, linetype, size, weight

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

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

TWO VARIABLES
both continuous
e <- ggplot(mpg, aes(cty, hwy))

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

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**)** - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

one discrete, one continuous
f <- ggplot(mpg, aes(class, hwy))

f + geom_col() x, y, alpha, color, fill, group, linetype, size

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

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

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

both discrete
g <- ggplot(diamonds, aes(cut, color))

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

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

l + geom_contour_filled(aes(fill = z)**)** x, y, alpha, color, fill, group, linetype, size, subgroup

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_density_2d() x, y, alpha, color, group, linetype, size

h + geom_hex() x, y, alpha, color, 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
Draw the appropriate geometric object depending on the simple features present in the data. aes() arguments: map_id, alpha, color, fill, linetype, linewidth.
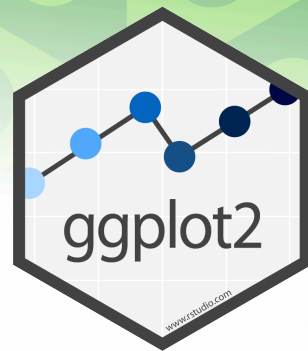
nc <- **sf::st_read(**system.file("shape/nc.shp", package = "sf")**)**

ggplot(nc) +
  geom_sf(aes(fill = AREA)**)**

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

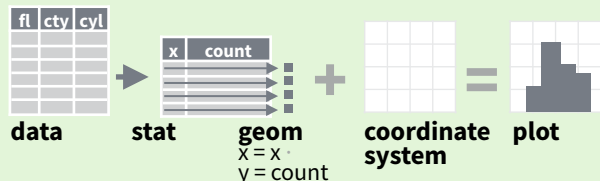**l + geom_tile(**aes(fill = z)**)** x, y, alpha, color, fill, linetype, size, width

posit

Stats An alternative way to build a layer.

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

data → **stat** → **geom** + **coordinate system** = **plot**
x = x · y = count

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

geom to use | **stat function** | **geommappings**
i + stat_density_2d(aes(fill = after_stat(level)**),
geom = "polygon"**)
variable created by stat

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

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

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

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

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

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

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

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

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

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

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

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

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

e + stat_quantile(quantiles = c(0.1, 0.9),
formula = y ~ log(x), method = "rq"**) x, y** | quantile

e + stat_smooth(method = "lm", formula = y ~ x, se = T,
level = 0.95**) x, y** | se, x, y, ymin, ymax

ggplot() + xlim(-5, 5**) + stat_function(**fun = dnorm,
n = 20, geom = "point"**) x** | x, y

ggplot() + stat_qq(aes(sample = 1:100)**)
x, y, sample** | sample, theoretical

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

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

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

**e + stat_identity()

**e + stat_unique()

Scales Override defaults with **scales** package.

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

n <- d + geom_bar(aes(fill = fl))

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 | **title to use in legend/axis** | **labels to use in legend/axis** | **breaks to use in legend/axis**

GENERAL PURPOSE SCALES

Use with most aesthetics

scale_*_continuous() - Map cont' values to visual ones.
scale_*_discrete() - Map discrete values to visual ones.
scale_*_binned() - Map continuous values to discrete bins.
scale_*_identity() - Use data values as visual ones.
scale_*_manual(values = c()**)** - Map discrete values to manually chosen visual ones.
scale_*_date(date_labels = "%m/%d"),
date_breaks = "2 weeks"**)** - Treat data values as dates.
scale_*_datetime() - Treat data values as date times.
Same as scale_*_date(). See ?strptime for label formats.

X & Y LOCATION SCALES

Use with x or y aesthetics (x shown here)

scale_x_log10() - Plot x on log10 scale.
scale_x_reverse() - Reverse the direction of the x axis.
scale_x_sqrt() - Plot x on square root scale.

COLOR AND FILL SCALES (DISCRETE)

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

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

COLOR AND FILL SCALES (CONTINUOUS)

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

o + scale_fill_distiller(palette = "Blues"**)

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

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

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

SHAPE AND SIZE SCALES

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

**p + scale_shape() + scale_size()
p + scale_shape_manual(values = c(3:7)**)

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
□ ○ △ + × ◇ ▽ ⊠ ✳ ⊕ ⊞ ⊠ ⊠ ⊡ △ ○ ○ ○ □ ◇ ▽

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

Coordinate Systems

r <- d + geom_bar()

r + coord_cartesian(xlim = c(0, 5)**)** - 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()
Flip cartesian coordinates by switching x and y aesthetic mappings.

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

r + coord_trans(y = "sqrt"**)** - x, y, xlim, ylim
Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.

π + coord_sf() - xlim, ylim, crs. Ensures all layers use a common Coordinate Reference System.

Position Adjustments

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

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

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

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

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

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

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

Each position adjustment can be recast as a function with manual **width** and **height** arguments:
s + geom_bar(position = position_dodge(width = 1))

Themes

**r + theme_bw()
White background with grid lines.

**r + theme_classic()

**r + theme_light()

**r + theme_gray()
Grey background (default theme).

**r + theme_linedraw()

**r + theme_minimal()
Minimal theme.

**r + theme_dark()
Dark for contrast.

**r + theme_void()
Empty theme.

r + theme() Customize aspects of the theme such as axis, legend, panel, and facet properties.
r + ggtitle("Title") + theme(plot.title.postion = "plot")
r + theme(panel.background = element_rect(fill = "blue"))

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 label:

**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

Labels and Legends

Use **labs()** to label the elements of your plot.

t + labs(x = "New x axis label", **y** = "New y axis label",
title ="Add a title above the plot",
subtitle = "Add a subtitle below title",
caption = "Add a caption below plot",
alt = "Add alt text to the plot",
<AES> = "New **<AES>** legend title")

t + annotate(geom = "text", x = 8, y = 9, label = "A"**)
Places a geom with manually selected aesthetics.

p + guides(x = guide_axis(n.dodge = 2)**)** Avoid crowded or overlapping labels with guide_axis(n.dodge or angle).

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

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

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

Zooming

Without clipping (preferred):

t + coord_cartesian(xlim = c(0, 100), ylim = c(10, 20)**)

With clipping (removes unseen data points):

t + xlim(0, 100**) + ylim(**10, 20**)

t + scale_x_continuous(limits = c(0, 100)**) +
scale_y_continuous(**limits = c(0, 100)**)

posit