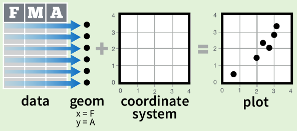
Data Visualization
with ggplot2
Cheat Sheet

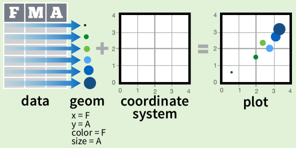
RStudio®

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 **qplot()** or **ggplot()**

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.

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.

last_plot()
Returns the last plot

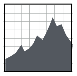
ggsave("plot.png", width = 5, height = 5)
Saves last plot as 5’ x 5’ file named "plot.png" in working directory. Matches file type to file extension.

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

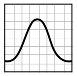
One Variable

Continuous

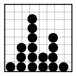
a <- ggplot(mpg, aes(hwy))



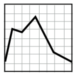
a + **geom_area(stat = "bin")**
x, y, alpha, color, fill, linetype, size
b + geom_area(aes(y = ..density..), stat = "bin")



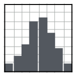
a + **geom_density**(kernel = "gaussian")
x, y, alpha, color, fill, linetype, size, weight
b + geom_density(aes(y = ..county..))



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



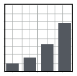
a + **geom_freqpoly()**
x, y, alpha, color, linetype, size
b + geom_freqpoly(aes(y = ..density..))



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

Discrete

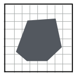
b <- ggplot(mpg, aes(fl))



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

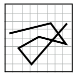
Graphical Primitives

c <- ggplot(map, aes(long, lat))

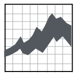


c + **geom_polygon**(aes(group = group))
x, y, alpha, color, fill, linetype, size

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

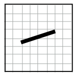


d + **geom_path**(lineend="butt", linejoin="round’, linemitre=1)
x, y, alpha, color, linetype, size

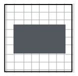


d + **geom_ribbon**(aes(ymin=unemploy - 900, ymax=unemploy + 900))
x, ymax, ymin, alpha, color, fill, linetype, size

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



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

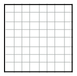


e + **geom_rect**(aes(xmin = long, ymin = lat,
xmax= long + delta_long,
ymax = lat + delta_lat))
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

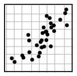
Two Variables

Continuous X, Continuous Y

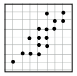
f <- ggplot(mpg, aes(cty, hwy))



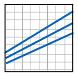
f + **geom_blank()**



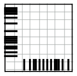
f + **geom_jitter()**
x, y, alpha, color, fill, shape, size



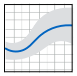
f + **geom_point()**
x, y, alpha, color, fill, shape, size



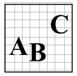
f + **geom_quantile()**
x, y, alpha, color, linetype, size, weight



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



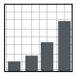
f + **geom_smooth**(model = lm)
x, y, alpha, color, fill, linetype, size, weight



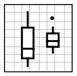
f + **geom_text**(aes(label = cty))
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Discrete X, Continuous Y

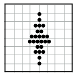
g <- ggplot(mpg, aes(class, hwy))



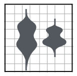
g + **geom_bar(stat = "identity")**
x, y, alpha, color, fill, linetype, size, weight



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



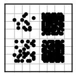
g + **geom_dotplot**(binaxis = "y", stackdir = "center")
x, y, alpha, color, fill



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

Discrete X, Discrete Y

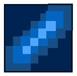
h <- ggplot(diamonds, aes(cut, color))



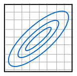
h + **geom_jitter()**
x, y, alpha, color, fill, shape, size

Continuous Bivariate Distribution

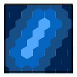
i <- ggplot(movies, aes(year, rating))



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



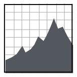
i + **geom_density2d()**
x, y, alpha, colour, linetype, size



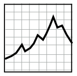
i + **geom_hex()**
x, y, alpha, colour, fill size

Continuous Function

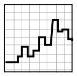
j <- ggplot(economics, aes(date, unemploy))



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



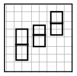
j + **geom_line()**
x, y, alpha, color, linetype, size



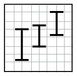
j + **geom_step**(direction = "hv")
x, y, alpha, color, linetype, size

Visualizing error

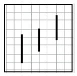
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
k <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))



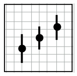
k + **geom_crossbar**(fatten = 2)
x, y, ymax, ymin, alpha, color, fill, linetype, size



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



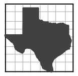
k + **geom_linerange()**
x, ymin, ymax, alpha, color, linetype, size



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

Maps

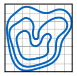
data <- data.frame(murder = USArrests$Murder,
state = tolower(rownames(USArrests)))
map <- map_data("state")
l <- ggplot(data, aes(fill = murder))



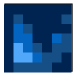
l + **geom_map**(aes(map_id = state), map = map) +
expand_limits(x = map$long, y = map$lat)
map_id, alpha, color, fill, linetype, size

Three Variables

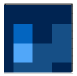
seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
m <- ggplot(seals, aes(long, lat))



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



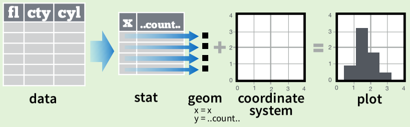
m + **geom_raster**(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE)
x, y, alpha, fill



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

Stats - An alternative way to build a layer

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



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

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

stat function | layer specific mappings | variable created by transformation

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

geom for layer | parameters for stat

1D distributions

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

2D distributions

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

3 Variables

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

Comparisons

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

Functions

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

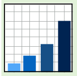
General Purpose

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

Scales

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

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



n

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

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

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

General Purpose scales

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

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

X and Y location scales

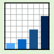
Use with x or y aesthetics (x shown here)

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

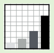
Color and fill scales

Discrete

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



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

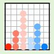


n + **scale_fill_grey**(start = 0.2, end = 0.8, na.value = "red")

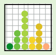
Continuous

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

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

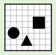


o + **scale_fill_gradient2**(low = "red", hight = "blue", mid = "white", midpoint = 25)



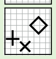
o + **scale_fill_gradientn**(colours = terrain.colors(6))
Also: rainbow(), heat.colors(), topo.colors(), cm.colors(), RColorBrewer::brewer.pal()

Shape scales



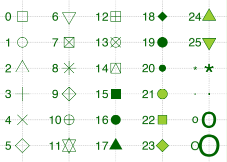
p <- f + **geom_point**(aes(shape = fl))

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

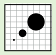


p + **scale_shape_manual**(values = c(3:7))
Shape values shown in chart on right

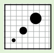
Manual shape values



Size scales



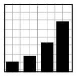
q <- f + **geom_point**(aes(size = cyl))



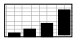
q + **scale_size_area**(max = 6)
Value mapped to area of circle (not radius)

Coordinate Systems

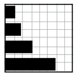
r <- b + geom_bar()



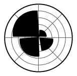
r + **coord_cartesian**(xlim = c(0, 5))
xlim, ylim
The default cartesian coordinate system



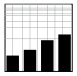
r + **coord_fixed**(ratio = 1/2)
ratio, xlim, ylim
Cartesian coordinates with fixed aspect ratio between x and y units



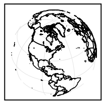
r + **coord_flip**()
xlim, ylim
Flipped Cartesian coordinates



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



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

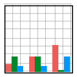


z + **coord_map**(projection = "ortho", orientation=c(41, -74, 0))
projection, orientation, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)

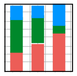
Position Adjustments

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

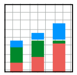
s <- ggplot(mpg, aes(fl, fill = drv))



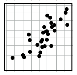
s + geom_bar(**position = "dodge"**)
Arrange elements side by side



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



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

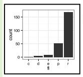


f + geom_point(**position = "jitter"**)
Add random noise to X and Y position of each element to avoid overplotting

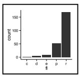
Each position adjustment can be recast as a function with manual **width** and **height** arguments

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

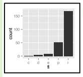
Themes



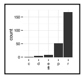
r + **theme_bw()**
White background with grid lines



r + **theme_classic()**
White background no gridlines



r + **theme_grey()**
Grey background (default theme)



r + **theme_minimal()**
Minimal theme

ggthemes - Package with additional ggplot2 themes

Faceting

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

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

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

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

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

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

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

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

Set **labeller** to adjust facet labels

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

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

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

α^c α^d α^e α^p α^r

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

c   d   e   p   r

Labels

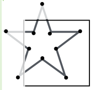
t + **ggtitle**("New Plot Title")
Add a main title above the plot
t + **xlab**("New X label")
Change the label on the X axis
t + **ylab**("New Y label")
Change the label on the Y axis
t + **labs**(title =" New title", x = "New x", y = "New y")
All of the above

Use scale functions to update legend labels

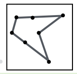
Legends

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

Zooming



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



With clipping (removes unseen data points)
t + **xlim**(0, 100) + **ylim**(10, 20)
t + **scale_x_continuous**(limits = c(0, 100)) + **scale_y_continuous**(limits = c(0, 100))