

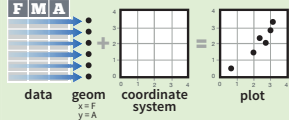
Data Visualization with ggplot2

Cheat Sheet

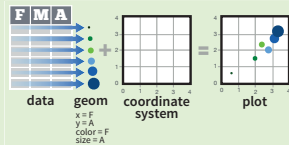


Basics

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



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



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

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

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

```
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, unemployment))



a + geom_blank()
(Useful for expanding limits)



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



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



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



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



b + geom_ribbon(aes(ymin = unemployment - 900, ymax = unemployment + 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, unemployment))



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



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



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

Visualizing error

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



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



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



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



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

Maps

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



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

Three Variables

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



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



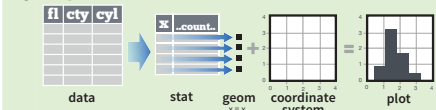
l + geom_contour(aes(z = z))
x, y, z, alpha, colour, group, linetype, size, weight



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

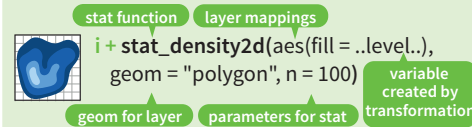
Stats - An alternative way to build a layer

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")`



`c + stat_bin(binwidth = 1, origin = 10)` 1D distributions
`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 = TRUE)` 2D distributions
`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))` 3 Variables
`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..`

`f + stat_boxplot(coef = 1.5)` Comparisons
`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..`

`e + stat_ecdf(n = 40)` Functions
`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..`

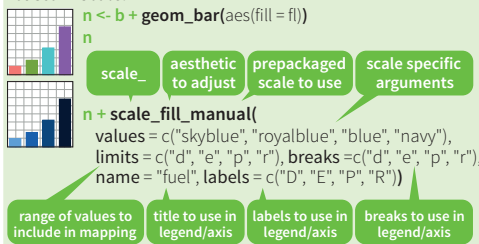
`ggplot() + stat_function(aes(x = -3:3), fun = dnorm, n = 101, args = list(sd=0.5))` General Purpose
`x | ..x.., ..y..`

`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")`
`e + 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.



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 Continuous

`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")`
`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
Manual shape values

`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

Size scales
`p + scale_radius(range=c(1,6))`
`p + scale_size_area(max_size=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(ytrans = "sqrt")`
xtrans, ytrans, limx, limy
Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.
`pi + 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
`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_linedraw()`
`r + theme_minimal()`
Minimal themes
`r + theme_void()`
Empty theme
`r + theme_dark()`
dark for contrast

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

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.

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))`