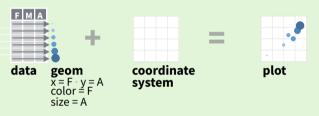
# Data Visualization with ggplot2:: CHEAT SHEET

### Basics

**ggplot2** is based on the **grammar of graphics**, the idea that you can build every graph from the same 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.

required ggplot (data = <DATA>) + <GEOM\_FUNCTION> (mapping = aes( <MAPPINGS> stat = <STAT>, position = <POSITION>) + required, <COORDINATE FUNCTION>+ defaults <FACET FUNCTION> supplied <SCALE FUNCTION> > <THEME\_FUNCTION>

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

### aesthetic mappings | data | geom

**qplot(**x = cty, y = hwy, 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

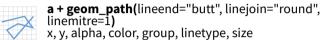
Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

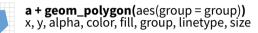
#### **GRAPHICAL PRIMITIVES**

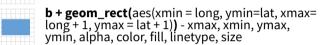
a <- ggplot(economics, aes(date, unemploy)) b <- ggplot(seals, aes(x = long, y = lat))

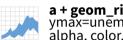
> a + geom\_blank() (Useful for expanding limits)











**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



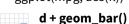
c + geom\_freqpoly() x, y, alpha, color, group, linetype, size



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))



x, alpha, color, fill, linetype, size, weight

#### **TWO VARIABLES**

### continuous x, continuous y

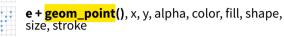
e <- ggplot(mpg, aes(cty, hwy))

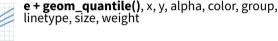


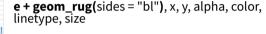
**e + geom\_label(**aes(label = cty), nudge\_x = 1, nudge\_y = 1, check\_overlap = TRUE) x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust



e + geom\_jitter(height = 2, width = 2) x, y, alpha, color, fill, shape, size









e + geom\_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\_col(), x, y, alpha, color, fill, group,



**f** + **geom\_boxplo**t(), 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

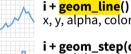
#### continuous function

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



i + geom area()

x, y, alpha, color, fill, linetype, size



x, y, alpha, color, group, linetype, size

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

### visualizing error

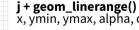
 $df \leftarrow data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)$ i <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))</pre>



j + geom\_crossbar(fatten = 2) x, y, ymax, ymin, alpha, color, fill, group, linetype,



**j + geom\_errorbar()**, x, ymax, ymin, alpha, color, group, linetype, size, width (also geom\_errorbarh())



x, ymin, ymax, alpha, color, group, linetype, size



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

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

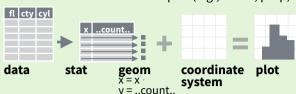


l + geom\_tile(aes(fill = z)), x, y, alpha, color, fill, linetype, size, width



# Stats An alternative way to build a layer

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



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 ..name.. syntax to map stat variables to aesthetics.



geom to use stat function geommappings

i + stat\_density2d(aes(fill = ..level..), geom = "polygon")

variable created by stat

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..

**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 1...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 =  $y \sim log(x)$ , 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() + stat\_function(**aes(x = -3:3), n = 99, fun = dnorm, args = list(sd=0.5)) x | ..x.., ..y..

e + stat\_identity(na.rm = TRUE)

 $\label{eq:ggplot() + stat_qq(aes(sample=1:100), dist = qt, dparam=list(df=5)) sample, x, y \mid ...sample..., ...theoretical..}$ 

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



### **GENERAL PURPOSE SCALES**

Use with most aesthetics

scale\_\*\_continuous() - map cont' values to visual ones

scale \* discrete() - map discrete values to visual ones

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.

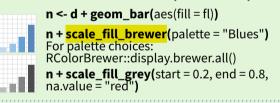
scale\_\*\_datetime() - treat data x values as date times. Use same arguments as scale x date(). See ?strptime for label formats.

#### **X & Y LOCATION SCALES**

Use with x or y aesthetics (x shown here)

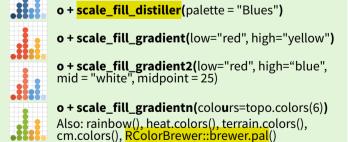
scale\_x\_log10() - Plot x on log10 scale scale\_x\_reverse() - Reverse direction of x axis scale\_x\_sqrt() - Plot x on square root scale

### **COLOR AND FILL SCALES (DISCRETE)**



### **COLOR AND FILL SCALES (CONTINUOUS)**

o <- c + geom\_dotplot(aes(fill = ..x..))</pre>



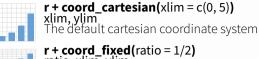
### **SHAPE AND SIZE SCALES**

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

p <- e + geom\_point(aes(shape = fl, size = cyl))

## **Coordinate Systems**

### r <- d + geom\_bar()



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

r + coord\_polar(theta = "x", direction=1)

# r + coord\_flip()

xlim, ylim Flipped Cartesian coordinates



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.

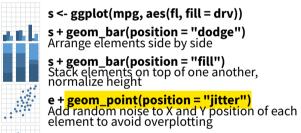


 $\pi$  + coord\_quickmap()

π + coord map(projection = "ortho", orientation=c(41, -74, 0))projection, 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.



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

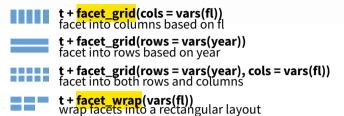


# **Faceting**

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



t <- ggplot(mpg, aes(cty, hwy)) + geom\_point()



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

t + facet\_grid(rows = vars(drv), cols = vars(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(cols = vars(fl), labeller = label_both)						
fl: c	fl: d	fl: e	fl: p	fl: r		
t + facet_grid(rows = vars(fl),						
labeller = label_bquote(alpha ^ .(fl)))						
$\alpha^c$	$\alpha^d$	$\alpha^e$	$\alpha^p$	$\alpha^r$		

### Labels

t + labs( x = "New x axis label", y = "New y axis label", title ="Add a title above the plot", Use scale functions subtitle = "Add a subtitle below title", to update legend caption = "Add a caption below plot", <AES> = "New <AES> legend title")

**t + annotate(**geom = "text", x = 8, y = 9, label = "A")

geom to place manual values for geom's aesthetics

## 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)

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

