GGPLOT

Installation and Load

```
if(!("ggplot2" %in% installed.packages())) install.packages("ggplot2")
library(ggplot2)
```

ggplot foundations

"...supply a dataset and **aesthetic mapping** (with aes()). You then add on **layers** (like geom_point() or geom_histogram()), **scales** (like scale_colour_brewer()), **faceting** specifications (like facet_wrap()) and **coordinate systems** (like coord_flip())..."

(from https://ggplot2.tidyverse.org)

```
ggplot(data = <DATA>) + <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

Data

```
v1 <- c(1,2,3,4,5)
v2<- v1*2
v3 <- v1*10
v4 <- v1 * 20
df1 <-data.frame(v1,v2,v3, v4); df1
rm(v1,v2,v3,v4)
```

Use of layered approach

```
# using layers to develop plot with arbitrary complexity
ggplot(df1,mapping=aes(x=v1,y=v2))
ggplot(df1,mapping=aes(x=v1,y=v2)) + geom_point()
```

Specifying Aesthetics manually

```
# specifying aesthetics
ggplot(df1,mapping=aes(x=v1,y=v2)) + geom_point(aes(v1,v3)) +
  geom_point(aes(v2,v3),color='red')


ggplot(df1,mapping=aes(x=v1,y=v2)) + geom_point(aes(v1,v3)) +
  geom_point(shape=24)


# direct aesthetics mapping
ggplot(df1,mapping=aes(x=v1,y=v2)) + geom_point(aes(y=v3))+
  geom_abline(intercept = 20) + geom_point(aes(y=v4, size=v4),shape=22,
fill='red')
ggplot(df1,mapping=aes(x=v1,y=v2)) + geom_point(aes(color="red"))+
  geom_point(aes(y=v3,color="blue"))
```

Specifying Aesthetics - Scaling

```
ggplot(df1,mapping=aes(x=v1,y=v2)) + geom_point(aes(v1,v3)) +
  geom_point(aes(shape=factor(v1)))

# aesthetics mapping with variables – scaling
ggplot(data=mpg)+geom_point(mapping=aes(x=displ,y= hwy))
ggplot(data = mpg) + geom_point(mapping = aes(x = displ, y = hwy, color =
class))
ggplot(data = mpg) + geom_point(mapping=aes(x=displ, y = hwy, size = class))
ggplot(data = mpg) + geom_point(mapping=aes(x=displ, y = hwy, shape = class))
ggplot(data = mpg) + geom_point(mapping=aes(x=displ, y = hwy, alpha = class))
ggplot(data = mpg) +
  geom_point(mapping=aes(x=displ, y = hwy, size = cty, color = class))
```

Plot limits

```
# use of plot limits
ggplot(df1,mapping=aes(x=v1,y=v2)) + geom_point(aes(y=v3)) +
  geom_abline(intercept = 0, slope = 1) + xlim(0,5) + ylim(0,50)
```

Mixing variable

```
# mixing variables mapping - overriding earlier specs
ggplot(df1,mapping=aes(x=v1,y=v3)) + geom_point(mapping=aes(x=v1,y=v2)) +
  geom_line(aes(y=v2))
ggplot(df1,mapping=aes(x=v1,y=v3)) + geom_point(mapping=aes(x=v1,y=v2)) +
  geom_line()
ggplot(df1,mapping=aes(x=v1,y=v3)) + geom_point(mapping=aes(x=v1,y=v2)) +
  geom_line(aes(y=v4))
```

Use of themes

```
# use of themes
ggplot(df1,mapping=aes(x=v1,y=v3)) + geom_point(mapping=aes(x=v1,y=v2)) +
  geom_line(aes(y=v4)) + theme_dark()
ggplot(df1,mapping=aes(x=v1,y=v3)) + geom_point(mapping=aes(x=v1,y=v2)) +
  geom_line(aes(y=v4)) + theme_classic()

# use of themes
mynamestheme <- theme(plot.title = element_text(family = "Helvetica", face = "bold", size = (15)),
                      legend.title = element_text(colour = "steelblue",  face = "bold.italic", 
family = "Helvetica"),
                      legend.text = element_text(face = "italic", colour="steelblue4",family = 
"Helvetica"),
                      axis.title = element_text(family = "Helvetica", size = (10), colour = 
"steelblue4"),
                      axis.text = element_text(family = "Courier", colour = "cornflowerblue", size = 
(10)))

ggplot(df1,mapping=aes(x=v1,y=v2)) + geom_point(aes(v1,v3)) +
  geom_point(aes(shape=factor(v1))) +
  labs(title = "Arbitrary plot", subtitle = "multiple scatter plots") +
  xlab('v1') + ylab('v2 or v3') + mynamestheme
```

Labeling

```
# use of labs for formatting plot labeling
ggplot(df1,mapping=aes(x=v1,y=v2)) + geom_point(aes(v1,v3)) +
  geom_point(aes(shape=factor(v1))) +
  labs(title = "Arbitrary plot", subtitle = "multiple scatter plots") +
  xlab('v1') + ylab('v2 or v3')
ggplot(df1,mapping=aes(x=v1,y=v2)) + geom_point(aes(v1,v3)) +
  geom_point(aes(shape=factor(v1))) +
  labs(title = "Arbitrary plot", subtitle = "multiple scatter plots",
  x = 'v1', y = 'v2 or v3')
```

Facets and Grids

```
# facets
ggplot(data = mpg) + geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_wrap(~ class)
ggplot(data = mpg) + geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_wrap(~ class, nrow = 2)

ggplot(data = mpg) + geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_wrap(~ class + drv)
ggplot(data = mpg) + geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_grid(class ~ drv)
```

Aesthetics description

```
# check for details of aesthetics
vignette("ggplot2–specs")
```

Multiple geoms - continuous and discrete data

```
# geoms - continuous and discrete data
ggplot(data = mpg) + geom_smooth(mapping=aes(x=displ, y = hwy))
ggplot(data = mpg) + geom_point(mapping=aes(x=displ, y = hwy)) +
geom_smooth(aes(x=displ, y = hwy))
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = color, y = ..count..))   #mark use of
computed variables in plot
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = color, y = ..count..),
           position = position_dodge())
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = color, y = ..prop..))
ggplot(data = diamonds,mapping = aes(x = cut, fill = clarity)) +
  geom_bar(alpha=1/2,position="dodge")
ggplot(data = diamonds) +
  geom_histogram(mapping = aes(x = carat), binwidth = 0.5)
ggplot(data = diamonds, mapping = aes(x = carat, color = cut)) +
  geom_freqpoly(binwidth = 0.1) + scale_color_hue(h=c(20,100))
ggplot(ToothGrowth, aes(x = factor(dose), y = len, fill = dose)) +  geom_boxplot()
```

Multiple geoms - continuous and discrete data

```
library(ggplot2)
head(BOD)
ggplot(BOD, aes(x = Time, y = demand)) + geom_col(fill="black", color =
"black")
ggplot(BOD, aes(x = factor(Time), y = demand)) +
  geom_col()
```

Breaking into parts

```
# breaking into multiple parts
base_plot <- ggplot(data=mpg,mapping=aes(x=displ,y=hwy))
base_plot + geom_point() + geom_smooth()
```

dplyr and ggplot

```
unusual <- diamonds %>%
  filter(y < 3 | y > 20) %>%  ##Same as where clause, hopeless diamonds
  arrange(y)
ggplot(data = unusual,mapping=aes(x= x,y = y)) +
  geom_point()
library(nycflights13)
nycflights13::flights %>%
  mutate( cancelled = is.na(dep_time),
          sched_hour = sched_dep_time %/% 100,
          sched_min = sched_dep_time %% 100,
          sched_dep_time = sched_hour + sched_min/60) %>%
  ggplot(mapping = aes(sched_dep_time)) +
  geom_freqpoly(mapping = aes(color = cancelled),binwidth =1/4)
```

Use of reorder

```
my_mpg %>%
  ggplot() +
geom_bar(aes(reorder(mpg$cyl,mpg$cyl,NROW),y=after_stat(prop),group=1)) +
  xlab(levels(reorder(mpg$cyl,mpg$cyl,NROW)))
```

Scales

```
ggplot(data = diamonds, mapping = aes(x = carat, color = cut)) +
  geom_freqpoly(binwidth = 0.1) + scale_color_hue(h=c(20,100))

tg<-ToothGrowth
tg$dose <- as.factor(ToothGrowth$dose)
my_plot <- ggplot(tg, aes(x = dose, y = len, fill = dose)) +  geom_boxplot()
my_plot
my_plot + scale_fill_manual(values = c("#004f71", "#465a01", "#981d97"))
my_plot + scale_fill_manual(values = c("red", "blue","green"))

#scale_fill_manual() for box plot, bar plot, violin plot,
#scale_color_manual() for lines and points
```

Stats

Many graphs, like scatterplots, plot the raw values of your dataset. Other graphs, like bar charts, calculate new values to plot:

- bar charts, histograms, and frequency polygons bin your data and then plot bin counts, the number of points that fall in each bin.
- smoothers fit a model to your data and then plot predictions from the model.
- boxplots compute a robust summary of the distribution and then display a specially formatted box.

The algorithm used to calculate new values for a graph is called a stat, short for statistical transformation.

You can learn which stat a geom uses by inspecting the default value for the stat argument.

geom_bar - stat_count() # prop is another computed variable

You can generally use geoms and stats interchangeably.

Stats

```
ggplot(data = diamonds) + geom_bar(mapping = aes(x = cut))
ggplot(data = diamonds) + stat_count(mapping = aes(x = cut))   #same as above

mpg_tab <- as.data.frame(table(mpg$cyl))
mpg_tab
ggplot(mpg_tab) + geom_bar(aes(Var1,Freq))   #gives error
ggplot(mpg_tab) + geom_bar(aes(Var1,Freq),stat = "identity")
ggplot(mpg) + geom_bar(aes(cyl))
ggplot(mpg) + stat_count(aes(cyl))
ggplot(mpg) + stat_count(aes(factor(cyl)))
ggplot(mpg) + stat_count(aes(cyl,y=stat(prop)))
ggplot(mpg) + geom_bar(aes(cyl,..prop..))
ggplot(mpg) + stat_count(aes(cyl,..prop..))
```

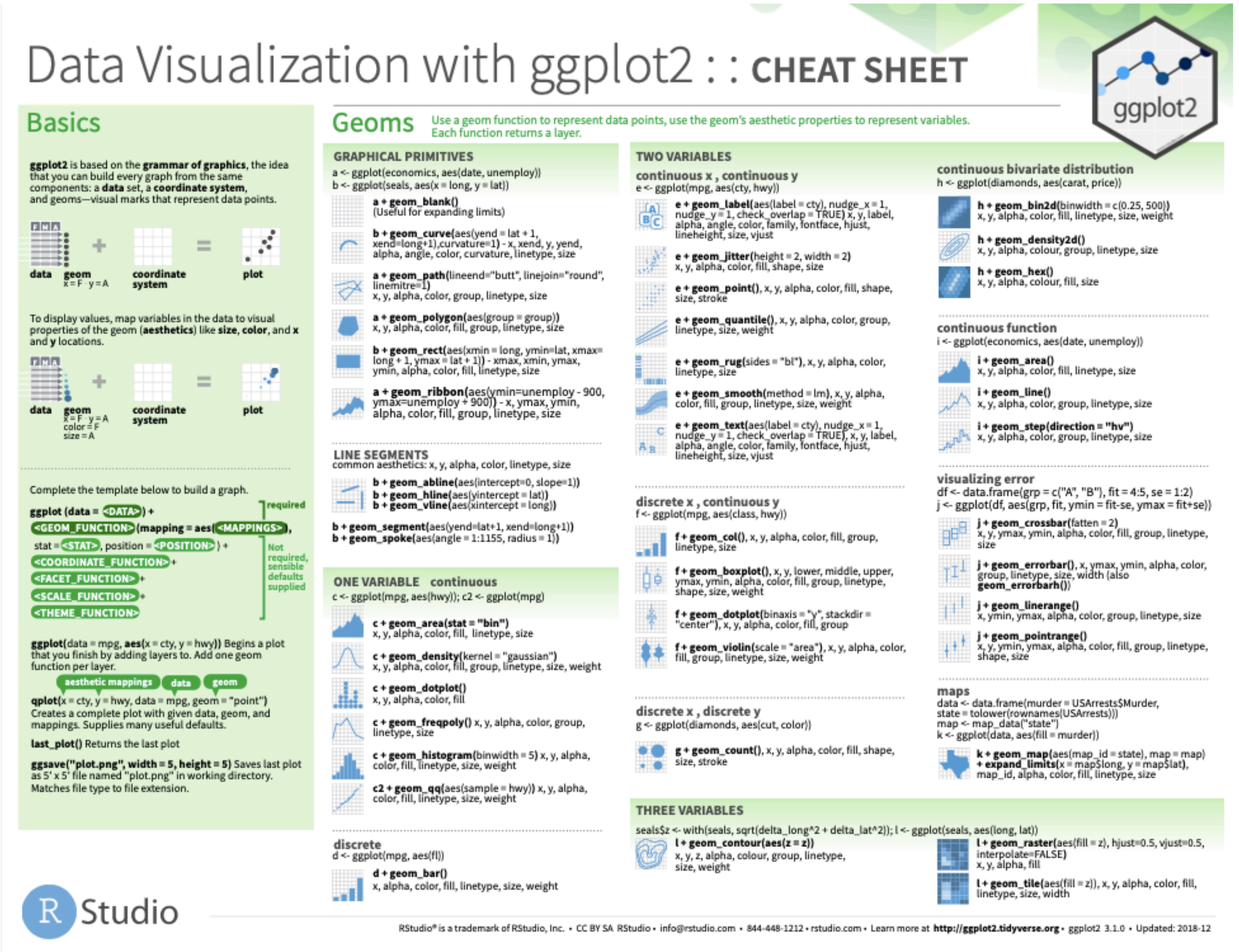
Stats

```
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)
c + stat_bin(mapping=aes(hwy,..ncount..),binwidth = 5)
c + stat_bin(mapping=aes(hwy,..density..),binwidth = 5)
c + stat_density(adjust = 1, kernel = 'gaussian')
c + stat_density(adjust = 1, kernel = 'triangular')
e <- ggplot(mpg, aes(cty, hwy))
e + stat_ellipse(level = 0.95, segments = 51, type = "t")
```

Types of plots

- One variable - continuous
- one variable - discrete
- two variables - both continuous
- two variables - both discrete
- two variables - one discrete, one continuous
- three variables (contour plots, etc.)
- faceted / gridded plots

ggplot reference sheet



Data Visualization with ggplot2 : : CHEAT SHEET

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.

data geom x = F · y = A coordinate system plot

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

data geom x = F · y = A color = F size = A coordinate system plot

Complete the template below to build a graph.

```
ggplot (data = <DATA>) +                                   required
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),
 stat = <STAT>, position = <POSITION>) +                   Not
<COORDINATE_FUNCTION> +                                    required,
<FACET_FUNCTION> +                                         sensible
<SCALE_FUNCTION> +                                         defaults
<THEME_FUNCTION>                                           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.

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)

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(group = group))
x, y, alpha, color, fill, group, 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

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_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

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))

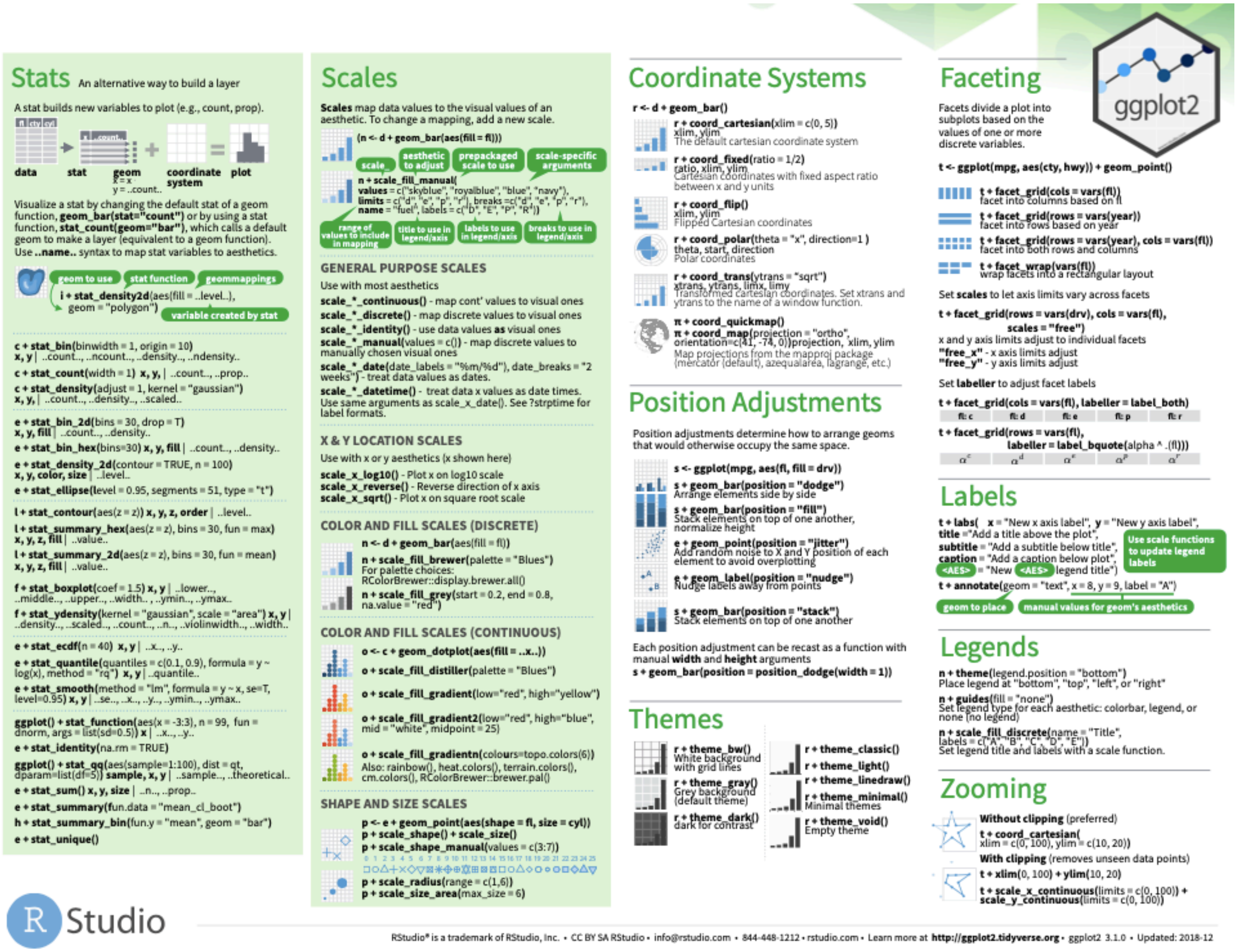
l + geom_contour(aes(z = z))
x, y, z, alpha, colour, group, linetype, size, weight

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

RStudio

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **http://ggplot2.tidyverse.org** • ggplot2 3.1.0 • Updated: 2018-12



Stats An alternative way to build a layer

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

data stat geom x = x · y = ..count.. coordinate system plot

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 | ..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() + stat_function(aes(x = -3:3), n = 99, fun = dnorm, args = list(sd=0.5)) **x** | ..x.., ..y..

e + stat_identity(na.rm = TRUE)

ggplot() + stat_qq(aes(sample=1:100), dist = qt, dparam=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 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 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 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)

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

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(colours=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()
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.

π + 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.

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_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

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(cols = vars(fl))
facet into columns based on fl

t + facet_grid(rows = vars(year))
facet into rows based on year

t + facet_grid(rows = vars(year), cols = vars(fl))
facet into both rows and columns

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

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)))

α^c	α^d	α^e	α^p	α^r

Labels

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",
<AES> = "New <AES> legend title")

Use scale functions to update legend labels

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))

RStudio

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **http://ggplot2.tidyverse.org** • ggplot2 3.1.0 • Updated: 2018-12