What is grafify?

`grafify` is a new R package for making great-looking `ggplot2` graphs quickly in R. It has 19 plotting functions that simplify common ggplot graphs and provide color-blind friendly themes.

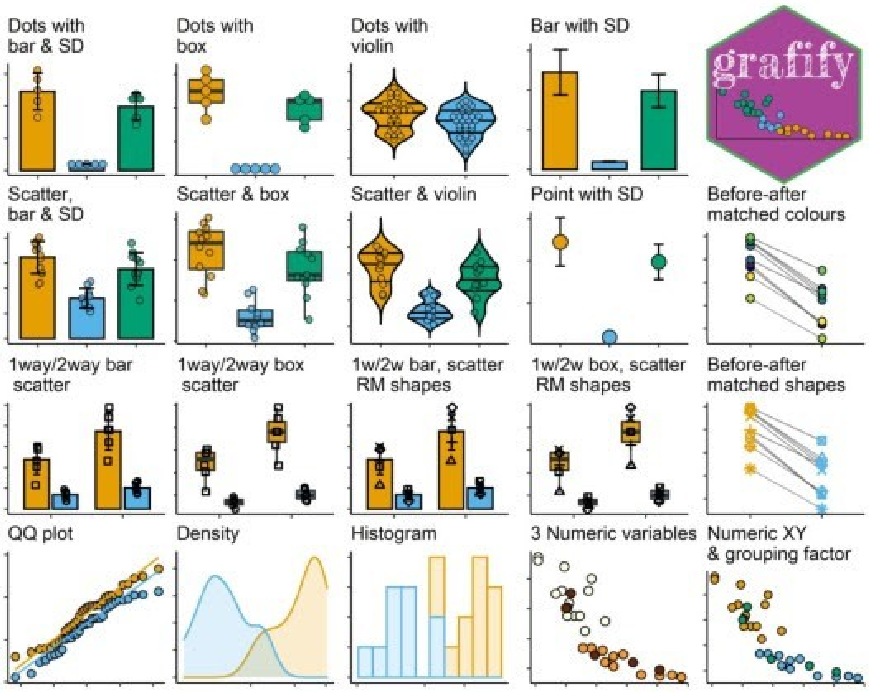


Image Credit: grafify package

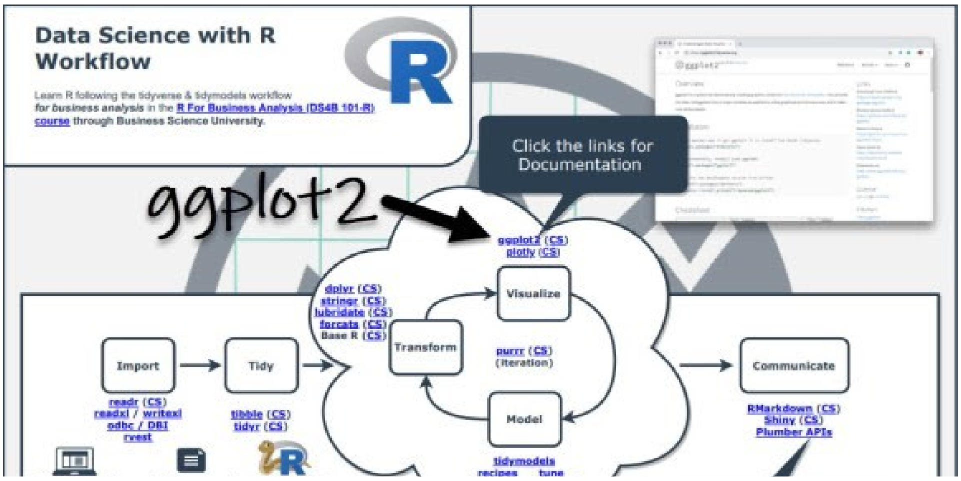
We'll go through a short tutorial to get you up and running with `grafify`.

Before we get started, get the R Cheat Sheet

`grafify` is great for making quick `ggplot2` plots. But, you'll still need to learn how to wrangle data with `dplyr` and visualize data with `ggplot2`. For those topics, I'll use the Ultimate R Cheat Sheet to refer to `dplyr` and `ggplot2` code in my workflow.

Quick Example:

Download the Ultimate R Cheat Sheet. **Then Click the "CS" next to "ggplot2"** opens the Data Visualization with ggplot2 Cheat Sheet.



Data Science with R Workflow

Learn R following the tidyverse & tidymodels workflow for business analysis in the R For Business Analysis (DS4B 101-R) course through Business Science University.

RStudio IDE (CS) fs (file system) reticulate (CS)

recipes tune
rsample yardstick
parsnip dials

CS = Cheat Sheet

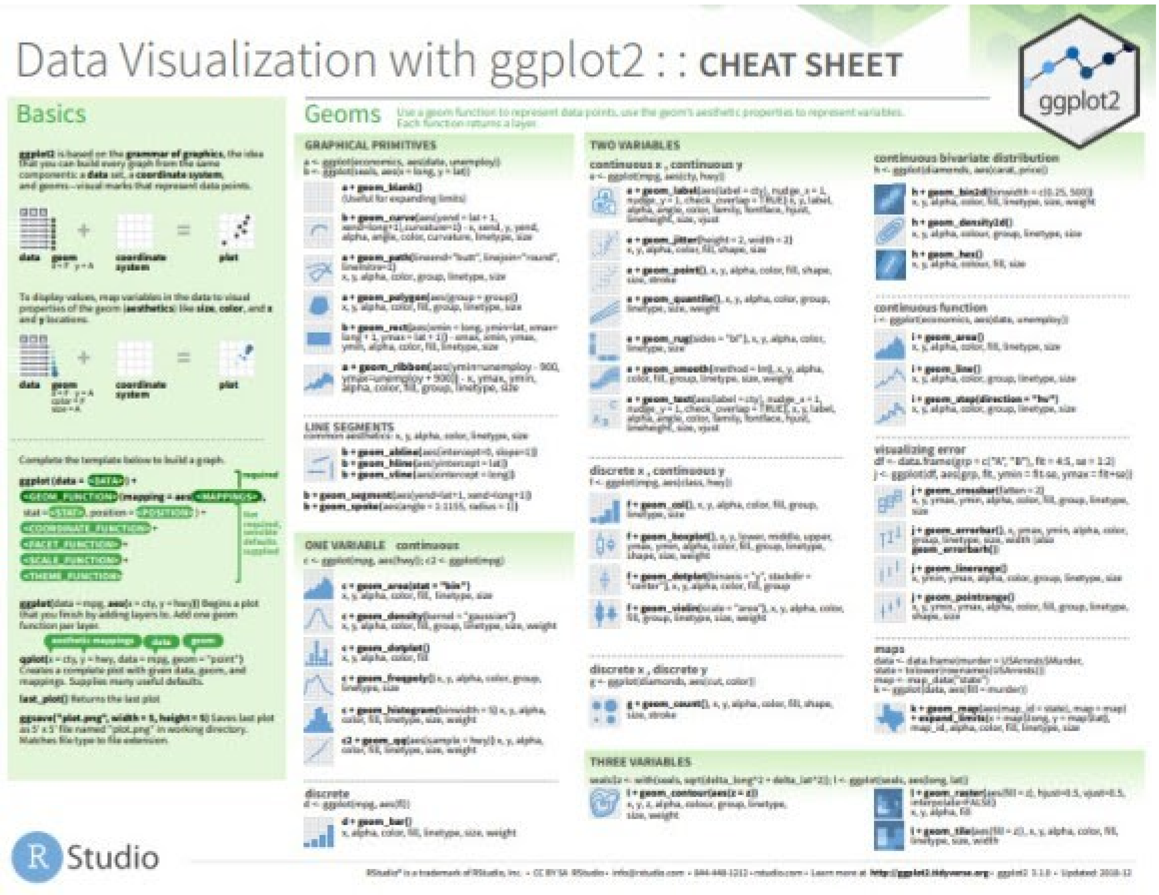
Important Resources

- R For Data Science Book: http://r4ds.had.co.nz/
- Rmarkdown: Book and Cookbook
- More Cheatsheets: https://www.rstudio.com/resources/cheatsheets/
- tidyverse packages: https://www.tidyverse.org/
- Connecting to databases: https://db.rstudio.com/
- Reproducible Environments: https://environments.rstudio.com/

Learn the Tidyverse & Tidymodels

Business Science University
university.business-science.io

Now you're ready to quickly reference `ggplot2` functions.



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.

data geom 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.

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.

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

Onto the tutorial.

How grafify works

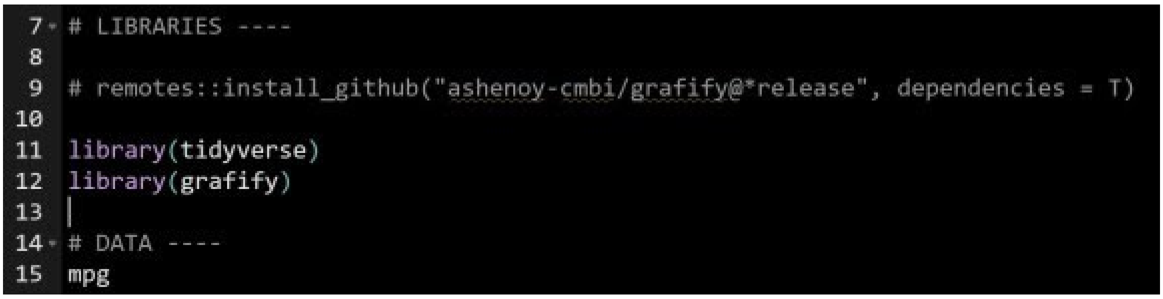
The `grafify` package extends `ggplot2` by adding several simplified plotting functions. In this tutorial, we'll cover:

- **2-Variable Functions:** `plot_scatterbar_sd()`, `plot_scatterbox()`, and `plot_dotviolin()`
- **3-Variable Functions:** `plot_3d_scatterbox()`
- **Before-After Functions:** `plot_befafter_colors()`

Load the Libraries and Data

First, run this code to:

1. **Load Libraries:** Load `grafify` and `tidyverse`.
2. **Import Data:** We're using the `mpg` dataset that comes with `ggplot2`.

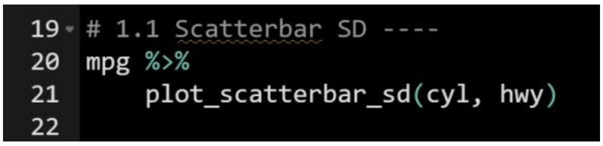


```
# LIBRARIES ----

# remotes::install_github("ashenoy-cmbi/grafify@*release", dependencies = T)

library(tidyverse)
library(grafify)

# DATA ----
mpg
```

Get the code.

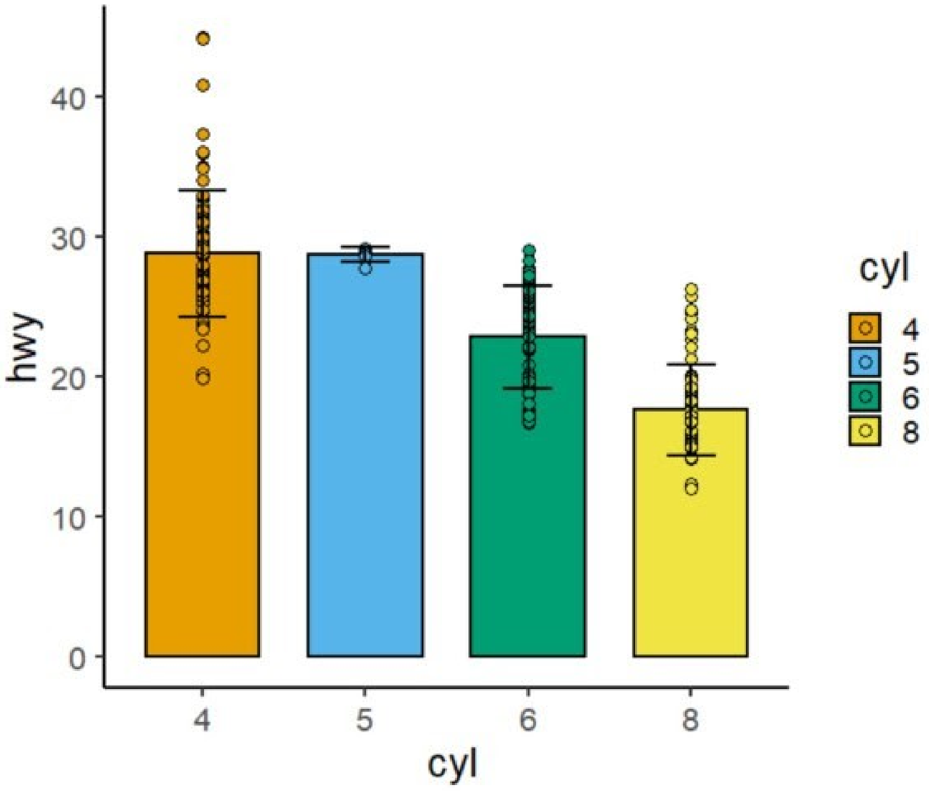
Scatterbar SD Plot

First, we can make a Scatterbar Plot that shows the data points along with error bars at a standard deviation. Simply use `plot_scatterbar_sd()`.



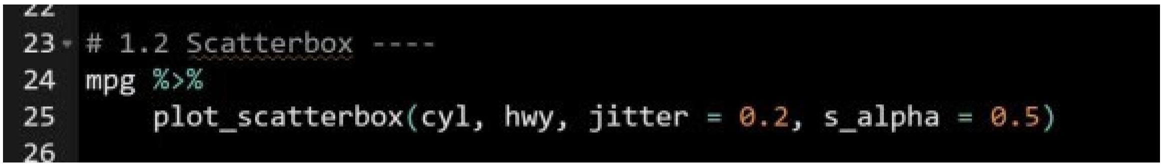
```
# 1.1 Scatterbar SD ----
mpg %>%
    plot_scatterbar_sd(cyl, hwy)

```

Get the code.



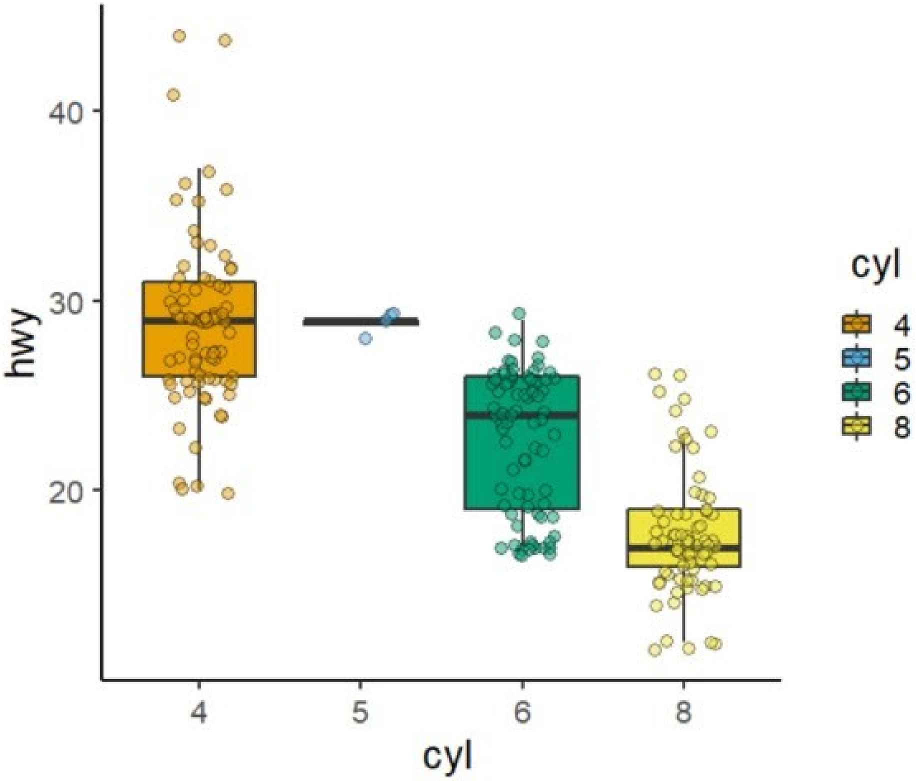
Scatterbox Plot

Next, we can make a Scatterbox Plot that shows a custom boxplot / jitter plot combination. I've added a jitter point to show the distribution. Simply use `plot_scatterbox()`.



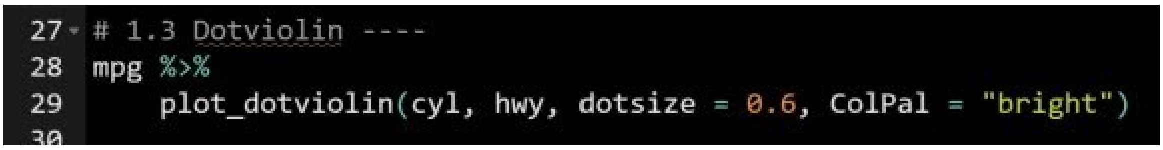
```
# 1.2 Scatterbox ----
mpg %>%
    plot_scatterbox(cyl, hwy, jitter = 0.2, s_alpha = 0.5)
```

Get the code.



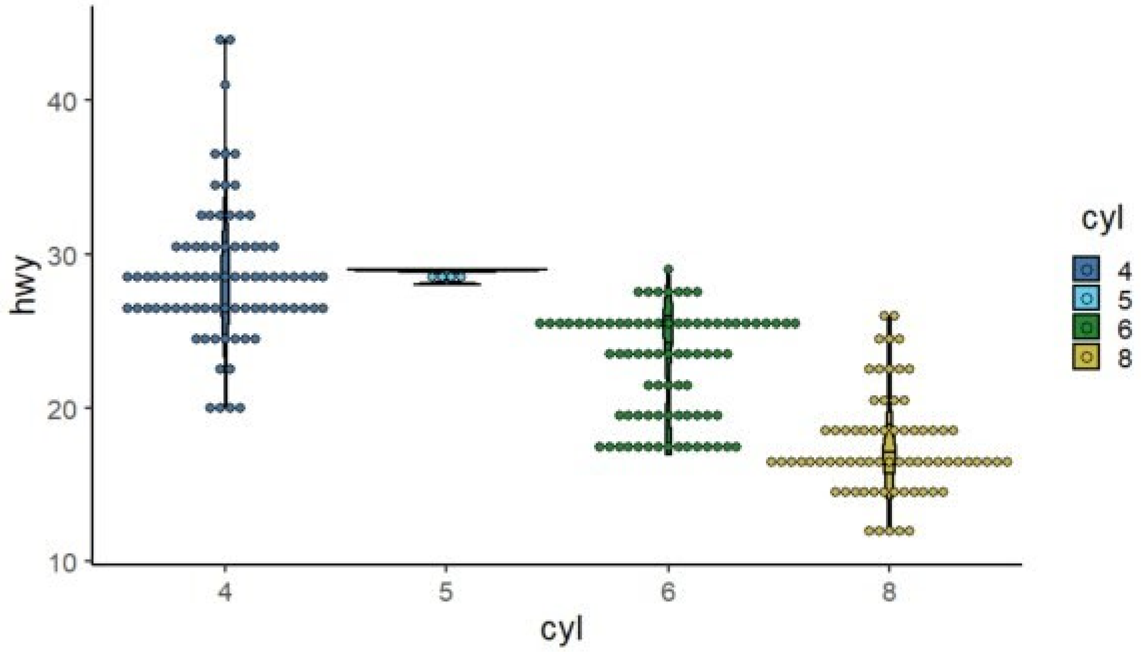
Dotviolin Plot

Next, we can make a Dotviolin Plot that shows a custom violin plot / dotplot combination. Simply use `plot_dotviolin()`.



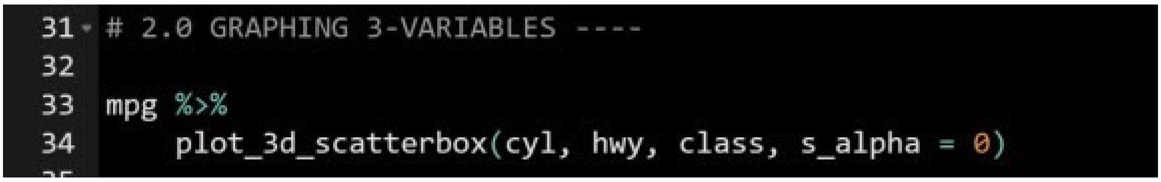
```
# 1.3 Dotviolin ----
mpg %>%
    plot_dotviolin(cyl, hwy, dotsize = 0.6, ColPal = "bright")
```

Get the code.



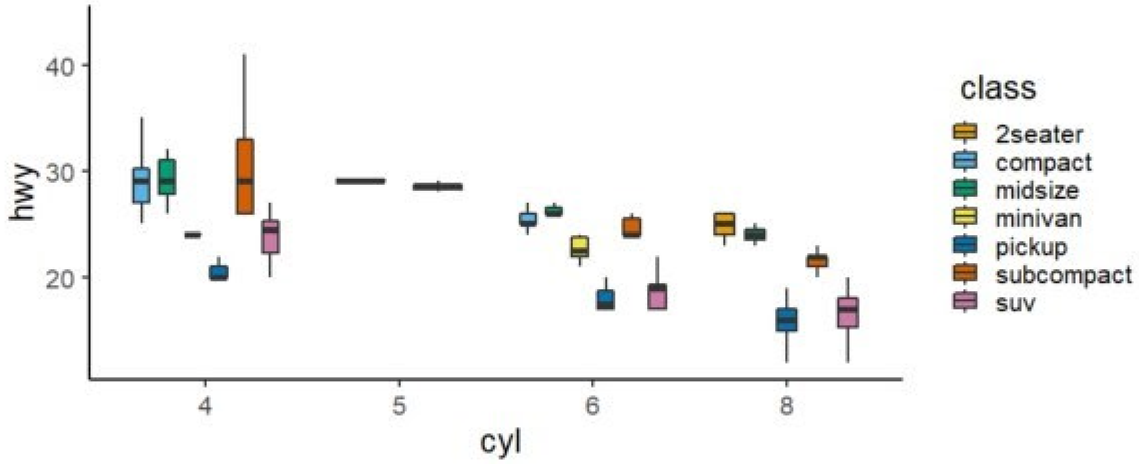
Scatterbox 3D Plot

Next, we can make a 3D Scatterbox Plot that shows three variables using boxplot / jitter plot combination. This is great for drilling into multiple categories. Simply use `plot_3d_scatterbox()`.



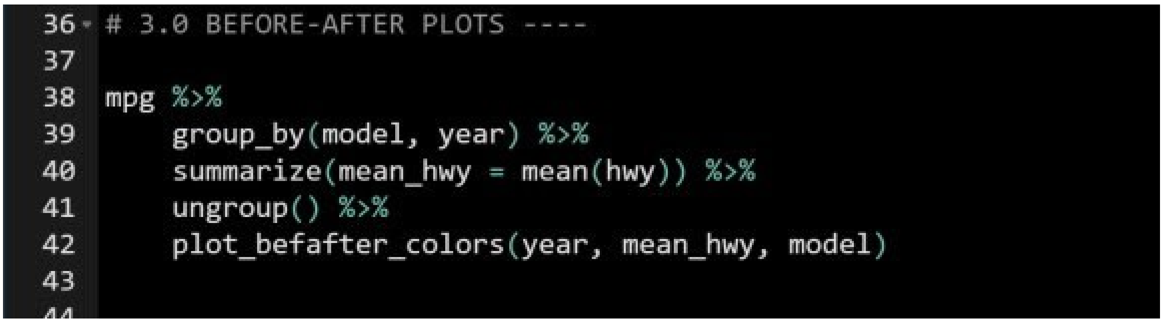
```
# 2.0 GRAPHING 3-VARIABLES ----

mpg %>%
    plot_3d_scatterbox(cyl, hwy, class, s_alpha = 0)
```

Get the code.



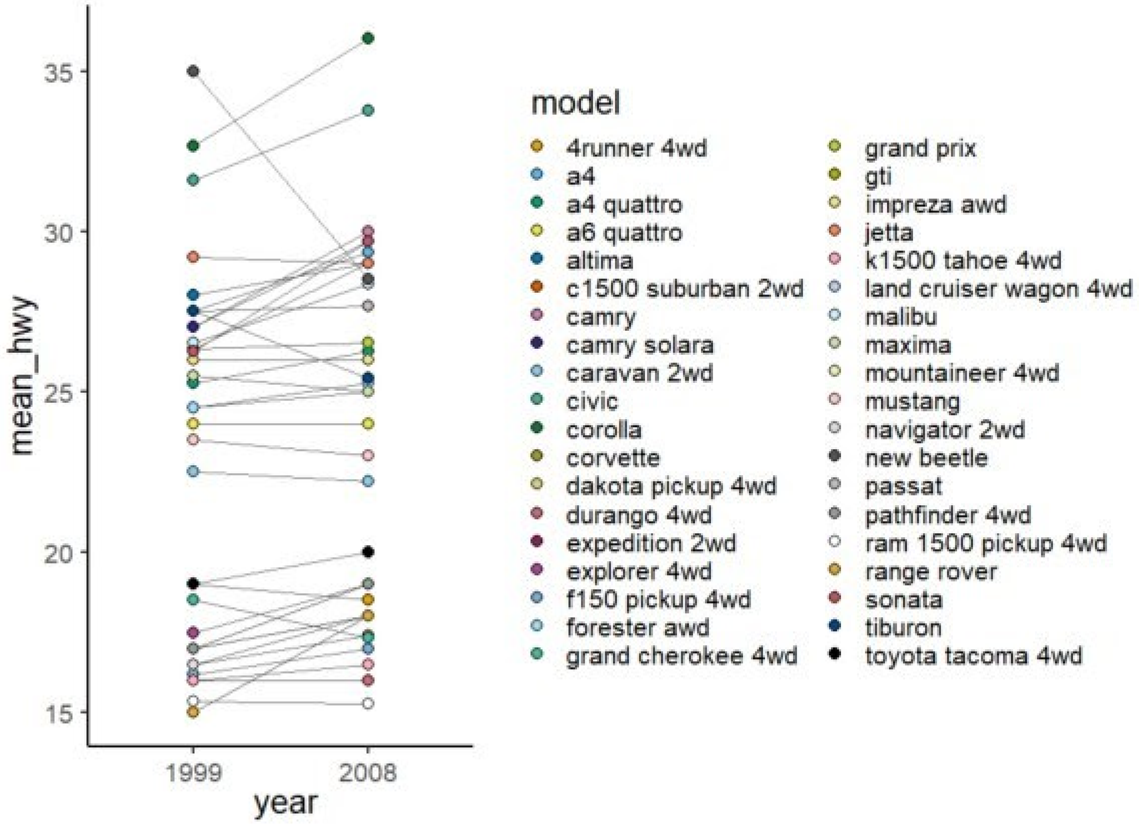
Before-After Plot

Finally, we can make a Before-After Plot that shows changes between two states (in this case how various models changed in MPG Fuel Efficiency from 1999 to 2008). This is great for comparing two states. Simply use `plot_befafter_colors()`.



```
# 3.0 BEFORE-AFTER PLOTS ----

mpg %>%
    group_by(model, year) %>%
    summarize(mean_hwy = mean(hwy)) %>%
    ungroup() %>%
    plot_befafter_colors(year, mean_hwy, model)

```

Get the code.



Summary

With 19 plotting functions, the grafify package makes it quick and easy to make custom ggplot2 visualizations that are easy to visualize and explore data. With that said, it’s critical to learn ggplot2 for plots beyond what grafify offers.