

# Data Visualization in R

download from [github.com/rjake/data-viz-talk](https://github.com/rjake/data-viz-talk)

Jake Riley

# Today's talk

- Intro
- What is ggplot
- Tips & Tricks
- Best Practices
- Try it out

# An intro

- Jake Riley
- Clinical Data Analyst at CHOP
- Avid package developer



- #dogdad
- @yake\_84



# Before we get started

- this talk is aimed at intermediate **ggplot2** users
- everything is within the **tidyverse** framework & R for Data Science (R4DS)
- the pipe **%>%** is used in many places and allows us to create a sequence of manipulations

```
iris %>% head(20)      head(iris, 20)
```

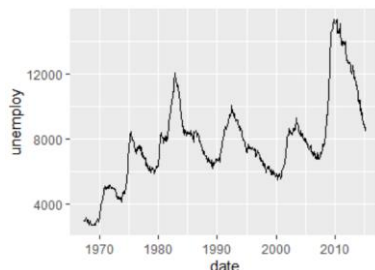
- the **+** used with **ggplot()** is another type of pipe
- you can pipe from a **dplyr** sequence into a **ggplot()** sequence

# What is ggplot?

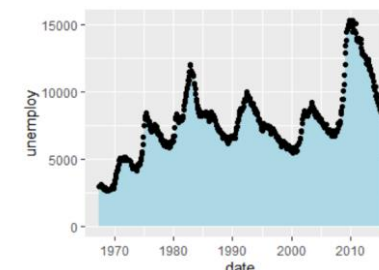
- **g**rammar of **g**raphics
- just like every sentence has a **subject, verb, and noun**, every chart has a
  - **coordinate system** (cartesian, polar)
  - **geometry** (bar, point, boxplot)
  - **aesthetics** (x, y, fill, alpha)
- the hope is that we will invent new types of charts

```
library(tidyverse)
econ <- ggplot(economics, aes(date, unemploy))
```

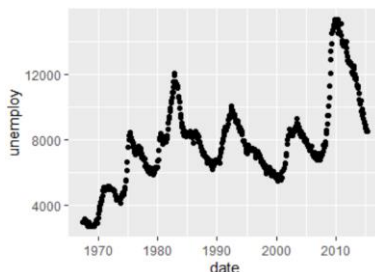
```
econ +
  geom_line()
```



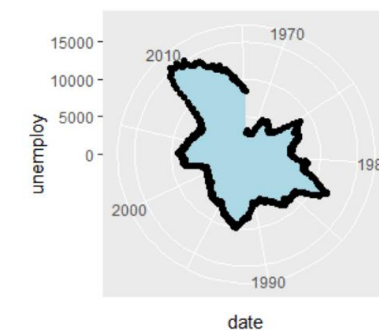
```
econ +
  geom_area(fill = "lightblue") +
  geom_point()
```



```
econ +
  geom_point()
```



```
econ +
  geom_area(fill = "lightblue") +
  geom_point() +
  coord_polar()
```



# Demystifying aes ()

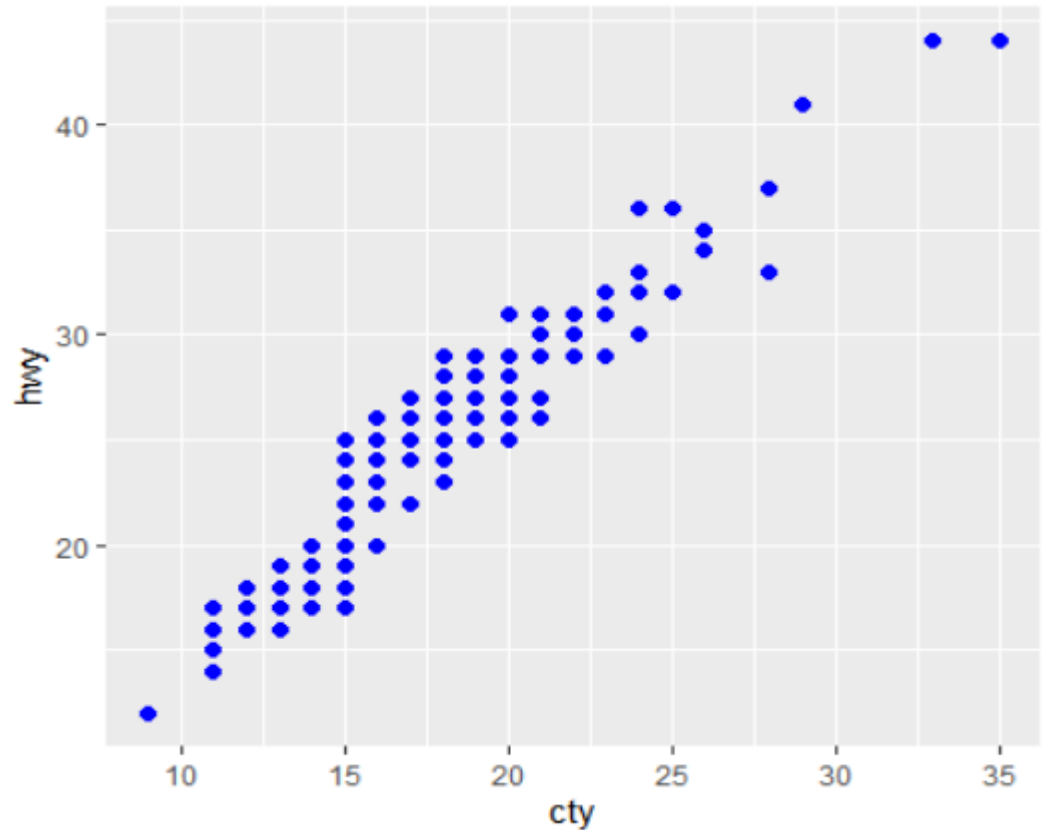
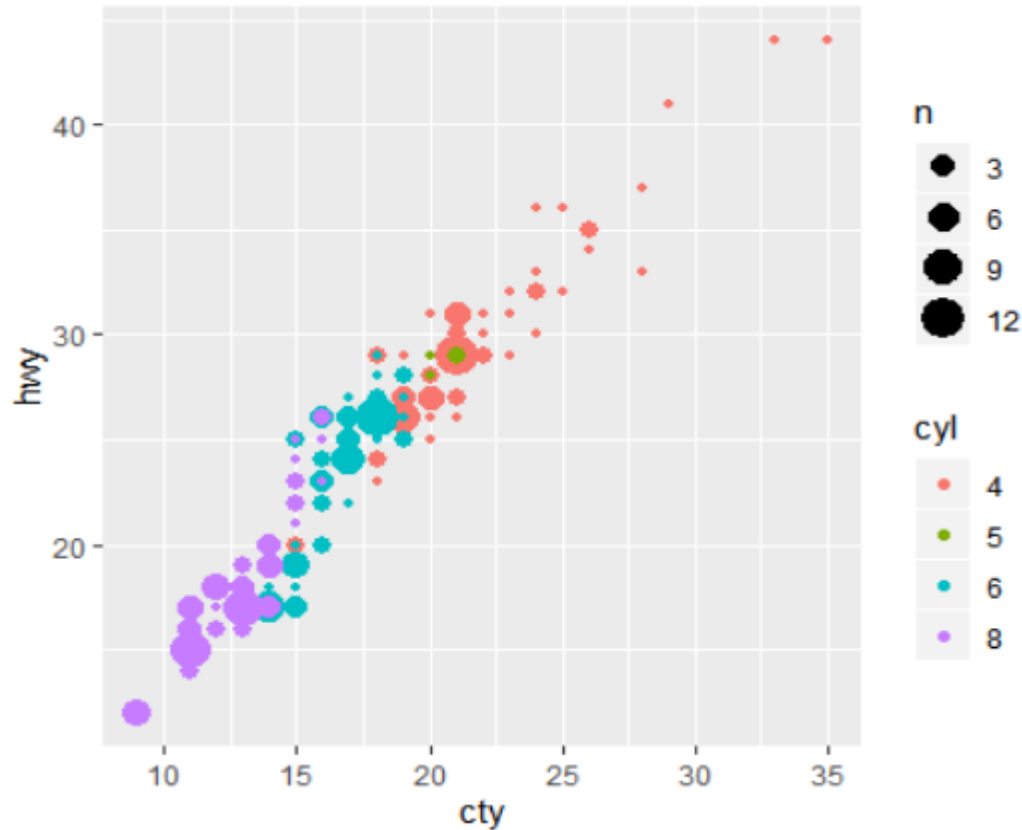
```
geom_point(aes(color = class, size = n), ...)  
geom_point(aes(...), color = "blue", size = 2)
```

- **aes ()** = aesthetics
- dynamic, data driven **variables** go inside the **aes ()**
- constant, static **values** go outside
- the first 2 arguments of **aes ()** are **x** and **y** and I will mostly omit naming these

# Note the difference

```
geom_point(aes(color = class, size = n), ...)
```

```
geom_point(aes(...), color = "blue", size = 2)
```



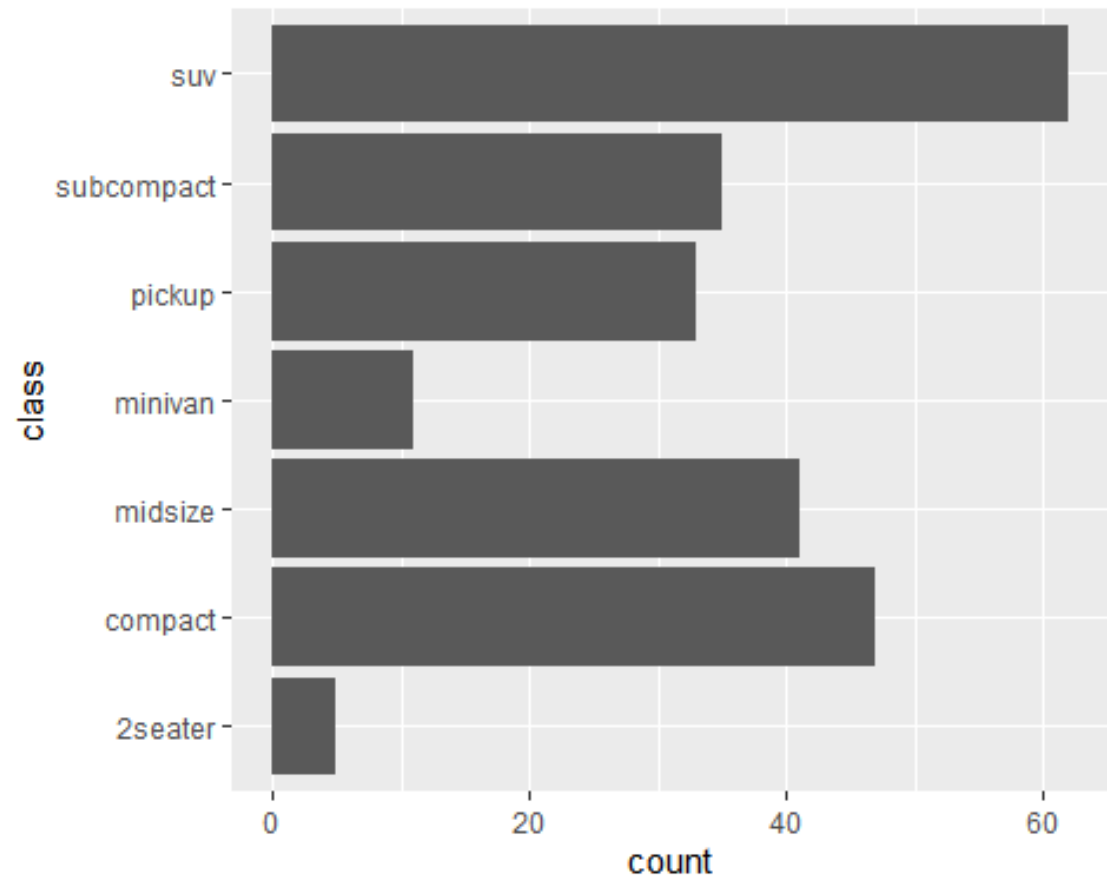


# Tips & Tricks

# Descending bar charts

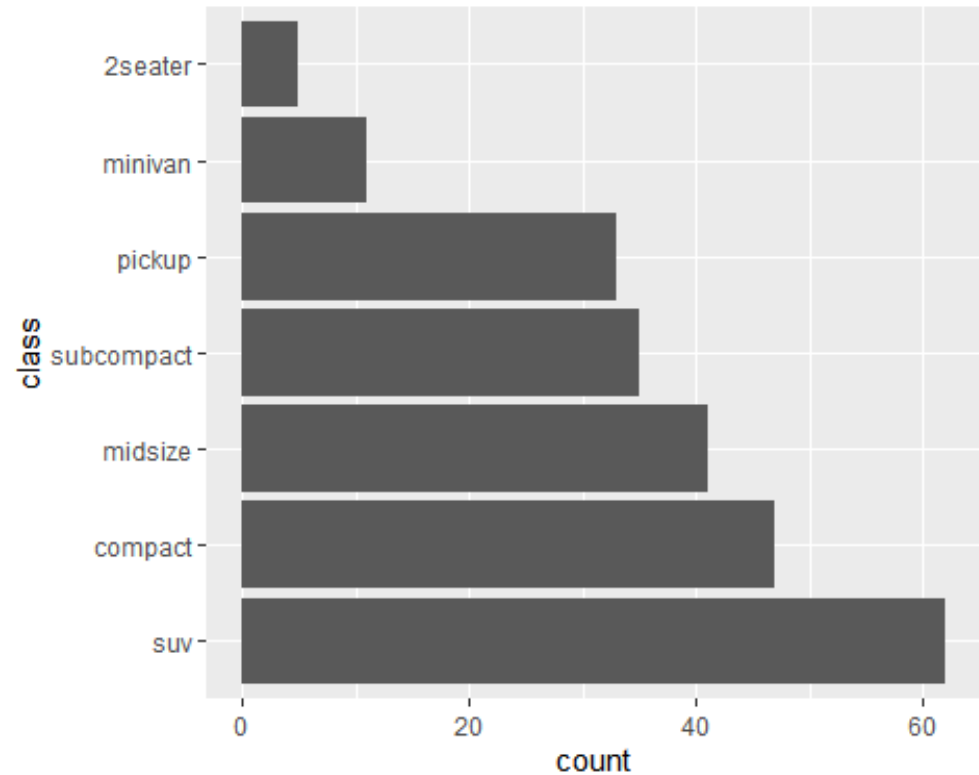
The number one thing I get asked is how to make a barchart go in descending order.

```
ggplot(mpg, aes(y = class)) +  
  geom_bar()
```



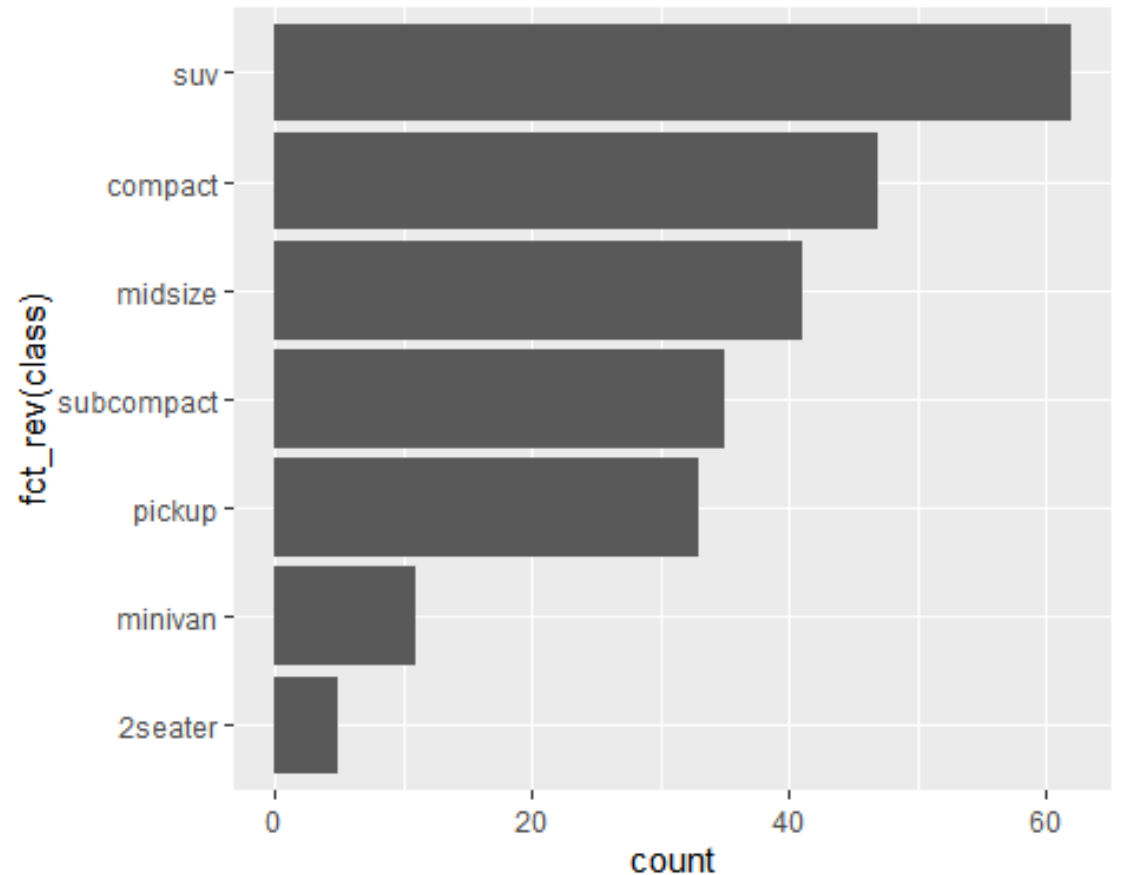
# Arrange by volume: `fct_infreq()`

```
mpg %>%  
  mutate(class = fct_infreq(class)) %>%  
  ggplot(aes(y = class)) +  
  geom_bar()
```



# Arrange in descending order: `fct_rev()`

```
mpg %>%  
  mutate(class = fct_infreq(class)) %>%  
  ggplot(aes(y = fct_rev(class))) +  
  geom_bar()
```



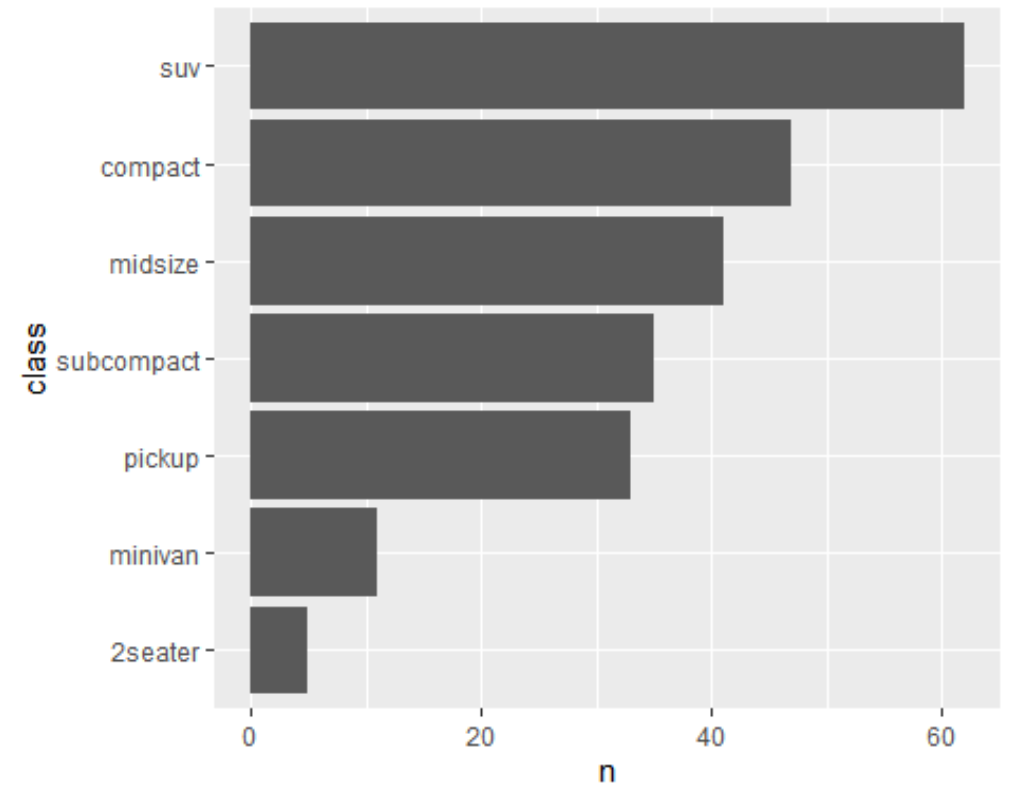
# Aggregated data: `fct_reorder()`

```
mpg %>%  
  count(class) %>%  
  mutate(class = fct_reorder(class, n, sum))
```

```
## # A tibble: 7 x 2  
##   class      n  
##   <fct>    <int>  
## 1 2seater      5  
## 2 compact    47  
## 3 midsize    41  
## 4 minivan    11  
## 5 pickup     33  
## 6 subcompact  35  
## 7 suv        62
```

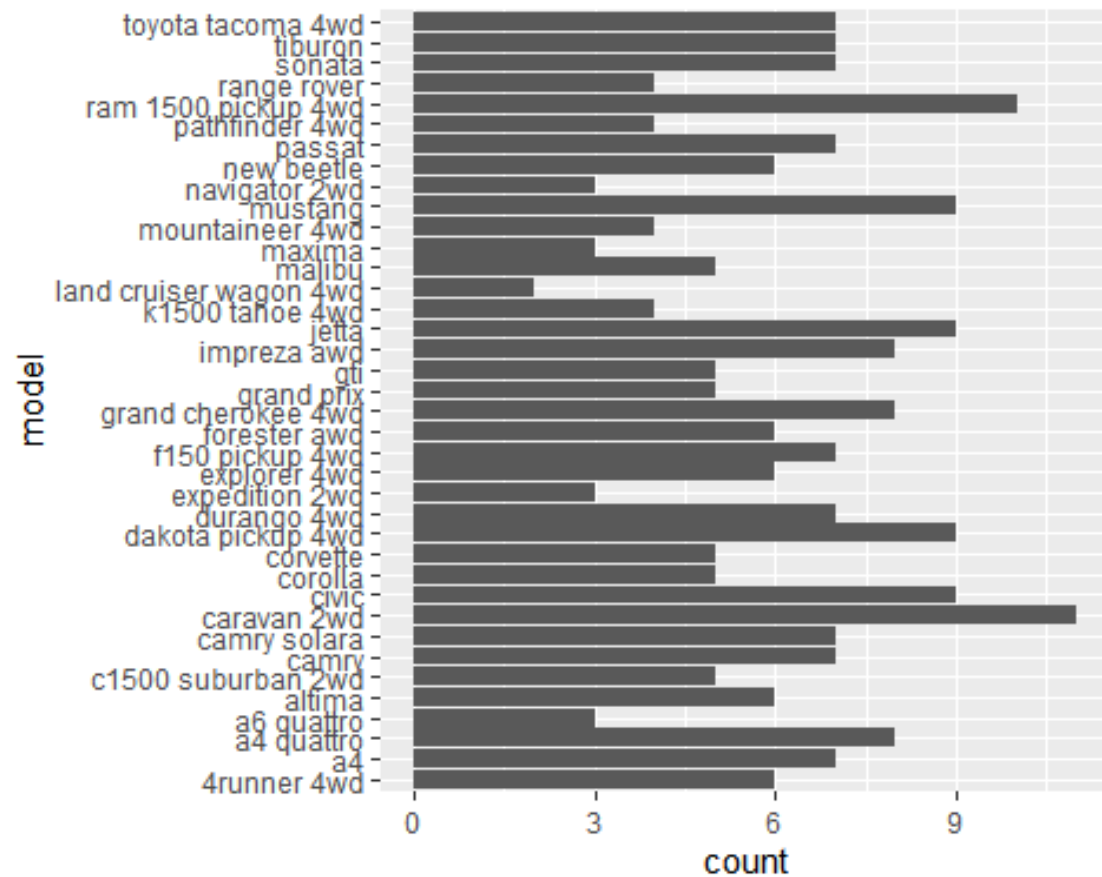
# Aggregated data: geom\_col()

```
mpg %>%  
  count(class) %>%  
  mutate(class = fct_reorder(class, n, sum)) %>%  
  ggplot(aes(n, class)) +  
  geom_col()
```



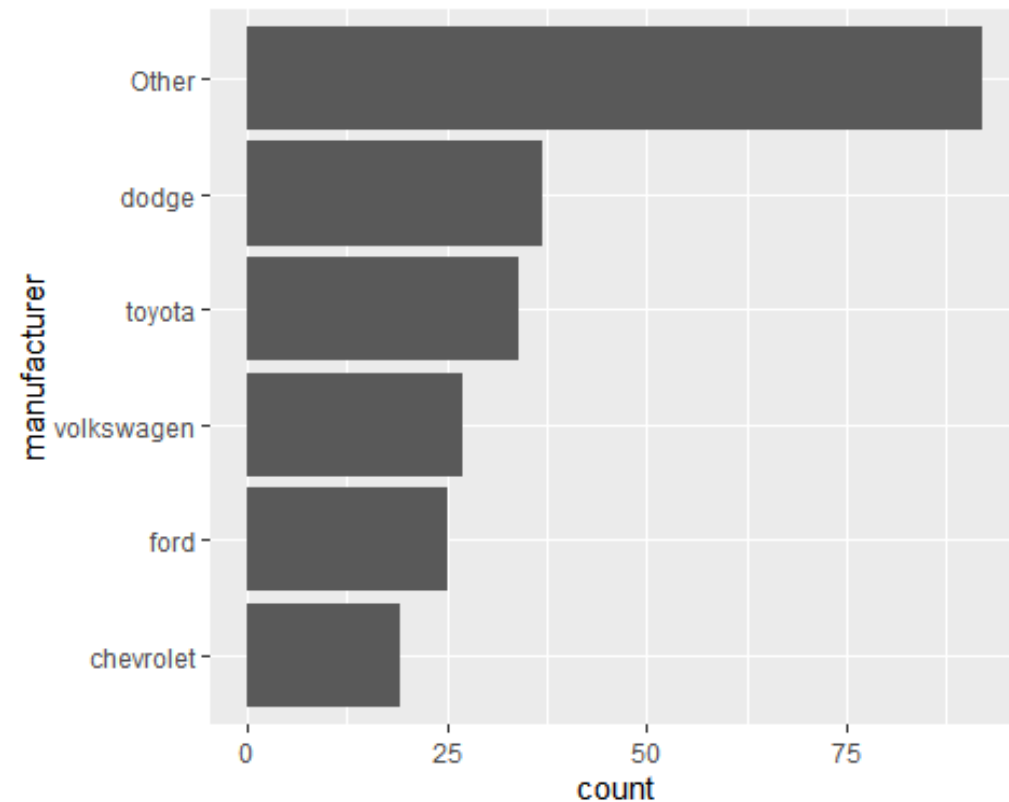
# Too many bars

```
ggplot(mpg, aes(y = model)) +  
  geom_bar()
```



# Too many bars: `fct_lump()`

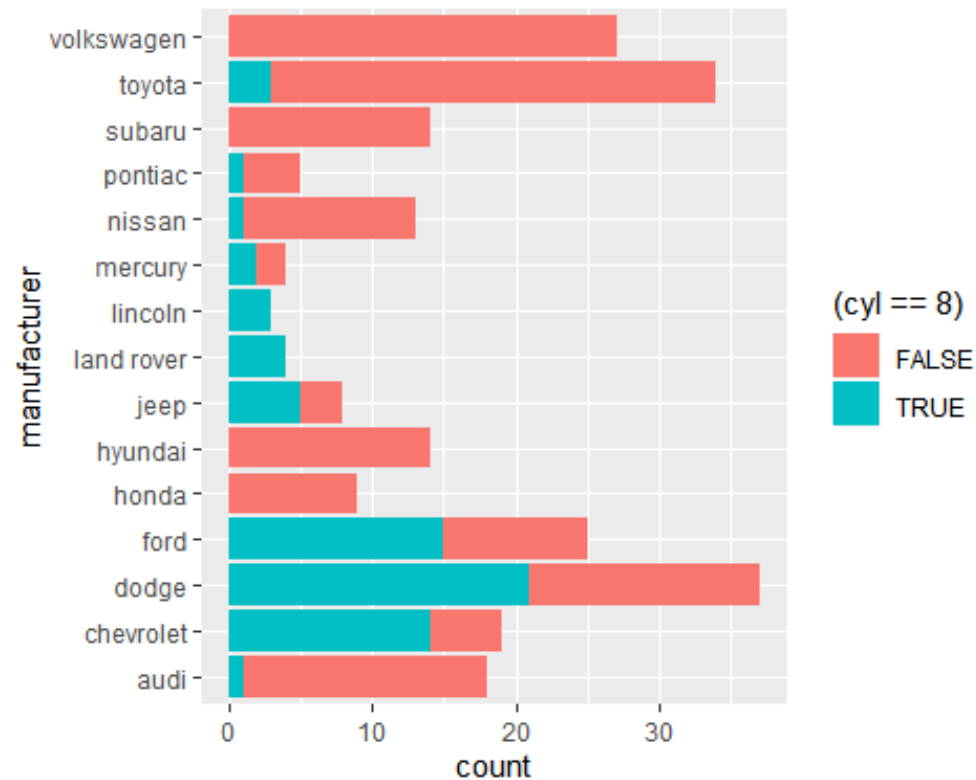
```
mpg %>%  
  mutate(  
    manufacturer =  
      fct_lump(manufacturer, 5) %>%  
      fct_infreq() %>%  
      fct_rev()  
  ) %>%  
  ggplot(aes(y = manufacturer)) +  
  geom_bar()
```





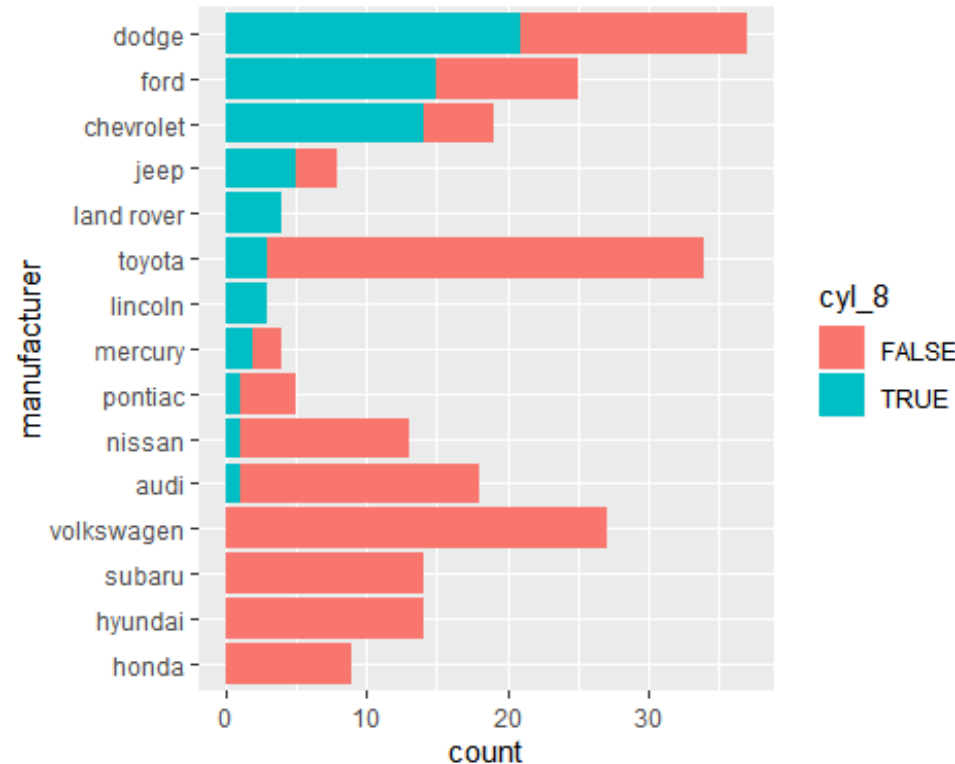
# Sorting by fill

```
ggplot(mpg, aes(y = manufacturer, fill = (cyl == 8))) +  
  geom_bar()
```



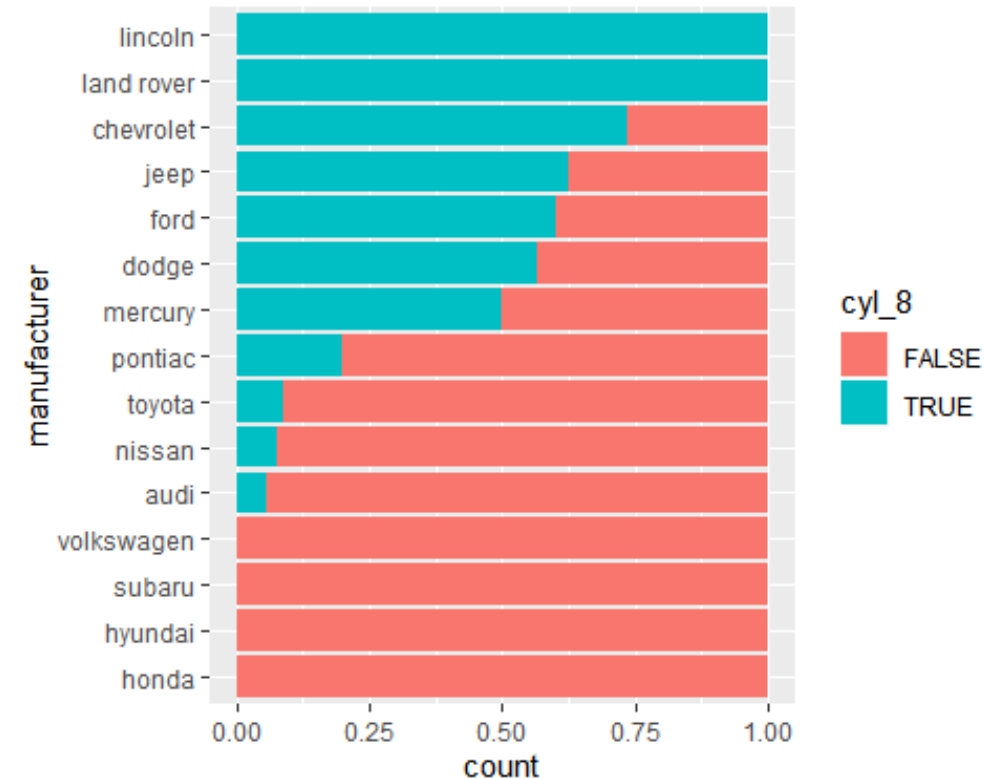
# Sorting by fill

```
mpg %>%  
  mutate(  
    cyl_8 = (cyl == 8),  
    manufacturer =  
      fct_reorder(manufacturer, cyl_8, sum)  
  ) %>%  
  ggplot(aes(y = manufacturer, fill = cyl_8)) +  
  geom_bar()
```



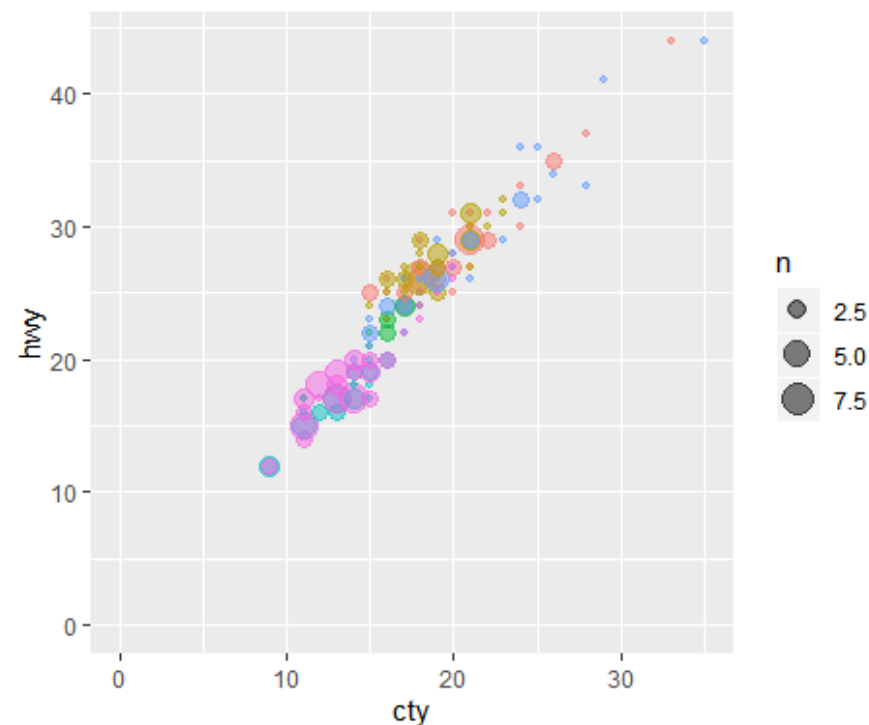
# Sorting by fill (percent)

```
mpg %>%  
  mutate(  
    cyl_8 = (cyl == 8),  
    manufacturer =  
      fct_reorder(manufacturer, cyl_8, mean)  
  ) %>%  
  ggplot(aes(y = manufacturer, fill = cyl_8)) +  
  geom_bar(position = "fill")
```



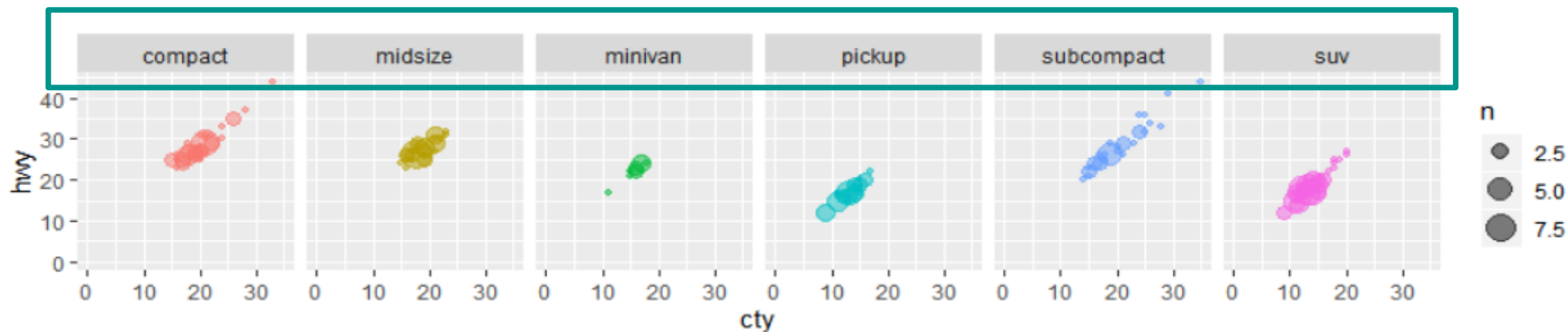
# facet\_grid() vs facet\_wrap()

```
p <-  
mpg %>%  
  filter(class != "2seater", cyl != 5) %>%  
  ggplot(aes(cty, hwy, color = class)) +  
  geom_count(alpha = 0.5) +  
  lims(x = c(0, NA), y = c(0, NA)) +  
  # can also use xlim() or scale_x_continuous  
  guides(color = FALSE) +  
  theme(aspect.ratio = 1)
```



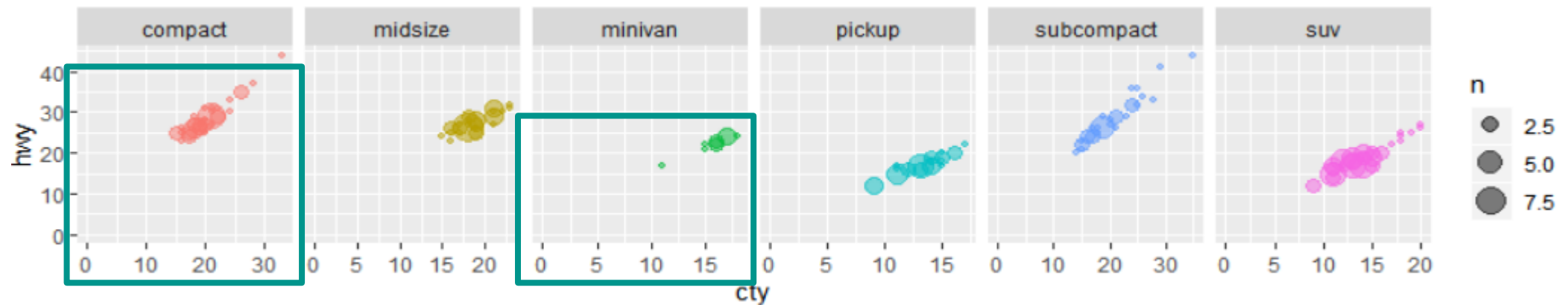
# facet\_grid(): new syntax

```
# this is the new syntax, replaces `facet_grid(~class)`  
p + facet_grid(cols = vars(class))
```



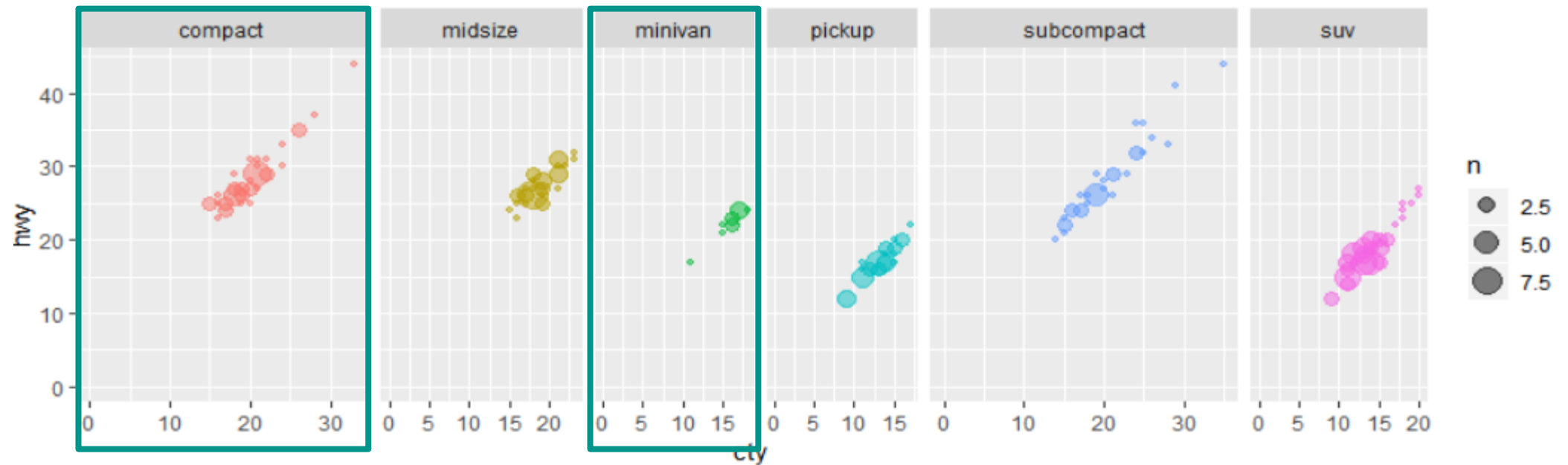
# facets: scales

```
# scales allows the x & y to vary  
# also "free_x", "free_y"  
p + facet_grid(cols = vars(class), scales = "free")
```



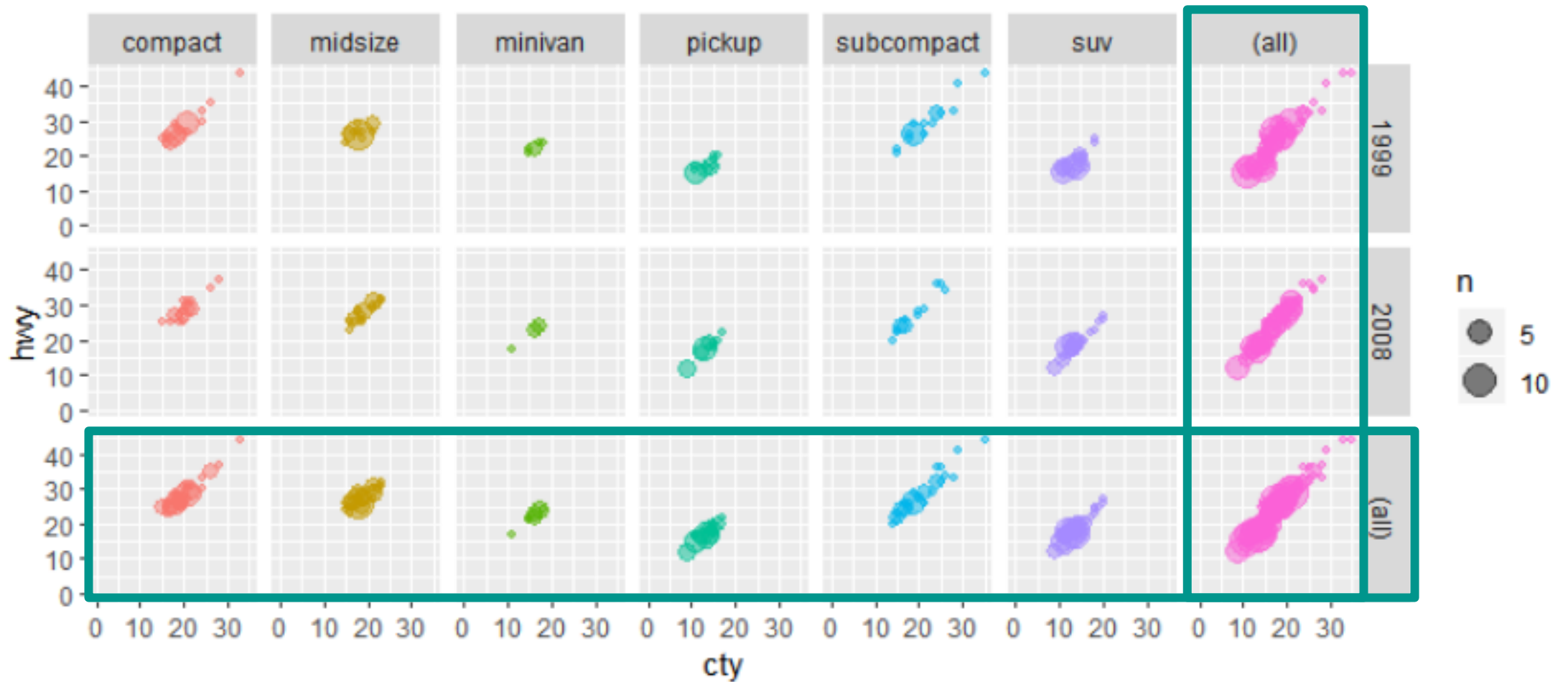
# facets: scales & space

```
p + facet_grid(cols = vars(class), scales = "free", space = "free")
```



# facets: margins

```
p + facet_grid(rows = vars(year), cols = vars(class), margins = TRUE)
```

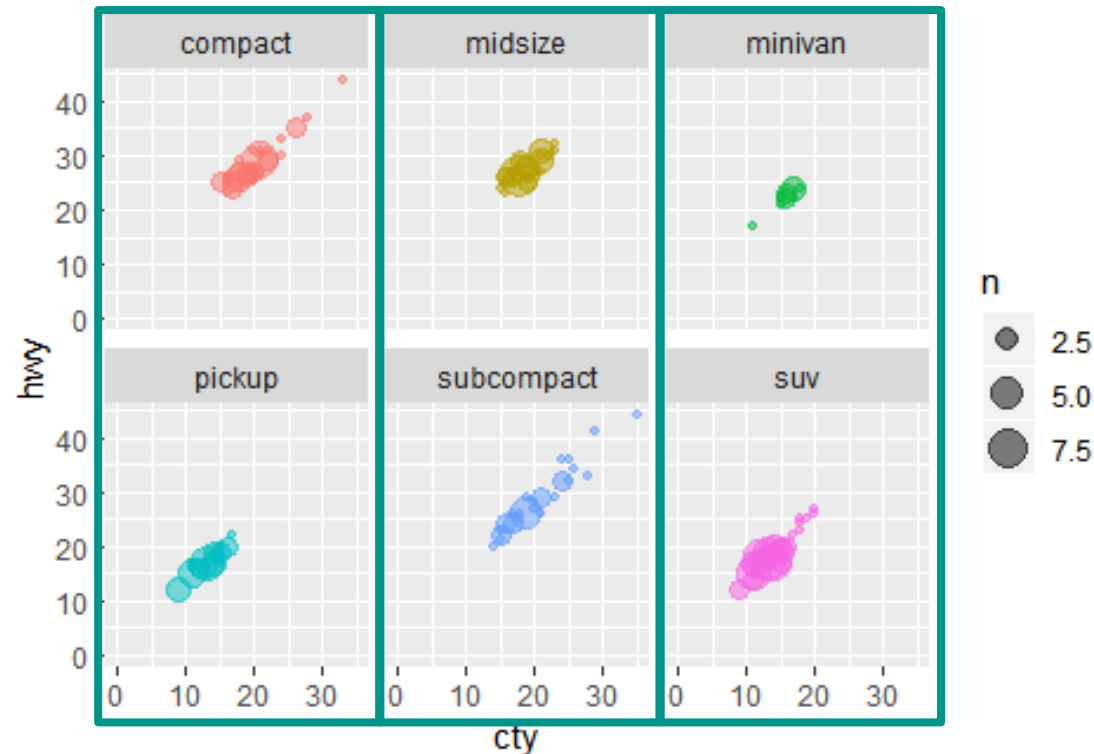




# facet\_wrap: # of columns/rows

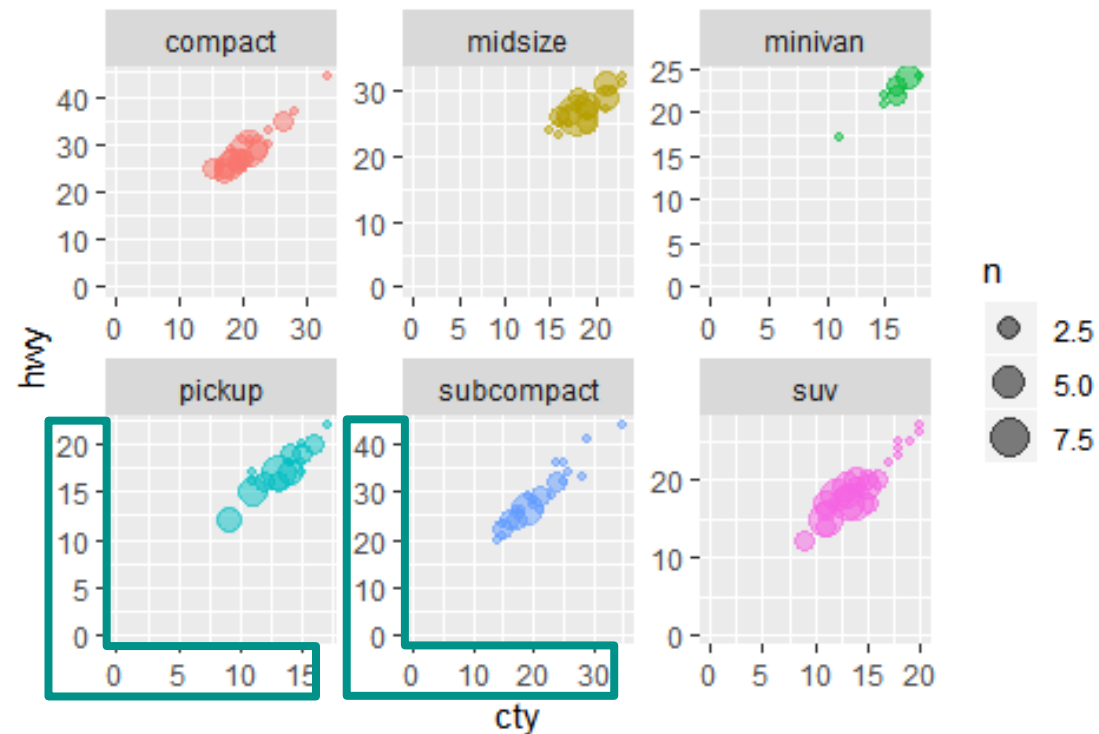
*# also nrow*

```
p + facet_wrap(~class, ncol = 3)
```



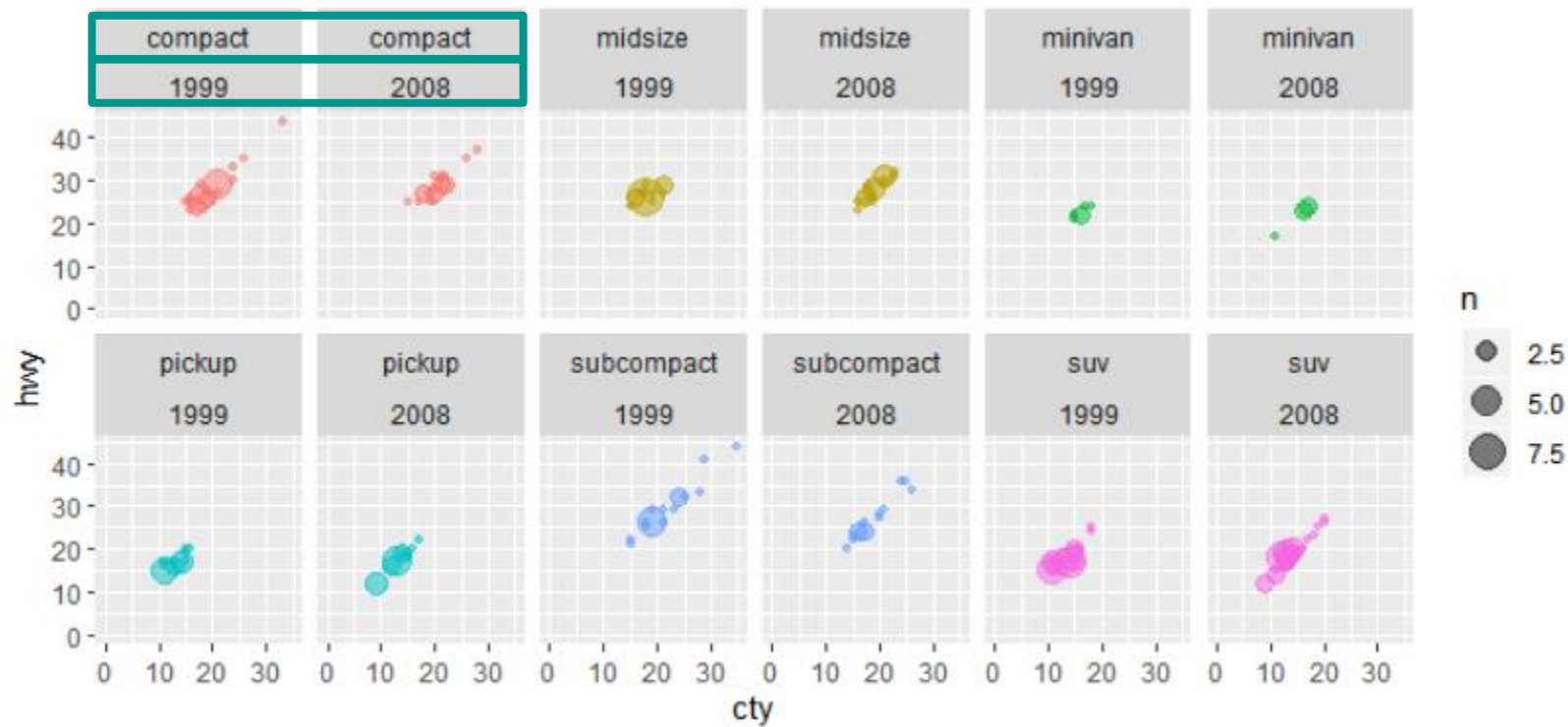
# facet\_wrap: scales

```
# space does not work with facet_wrap()  
p + facet_wrap(~class, ncol = 3, scales = "free")
```



# facets: (a + b)

```
# also works with facet_grid  
p + facet_wrap(~class + year, nrow = 2)
```



# scale\_\*\_identity()

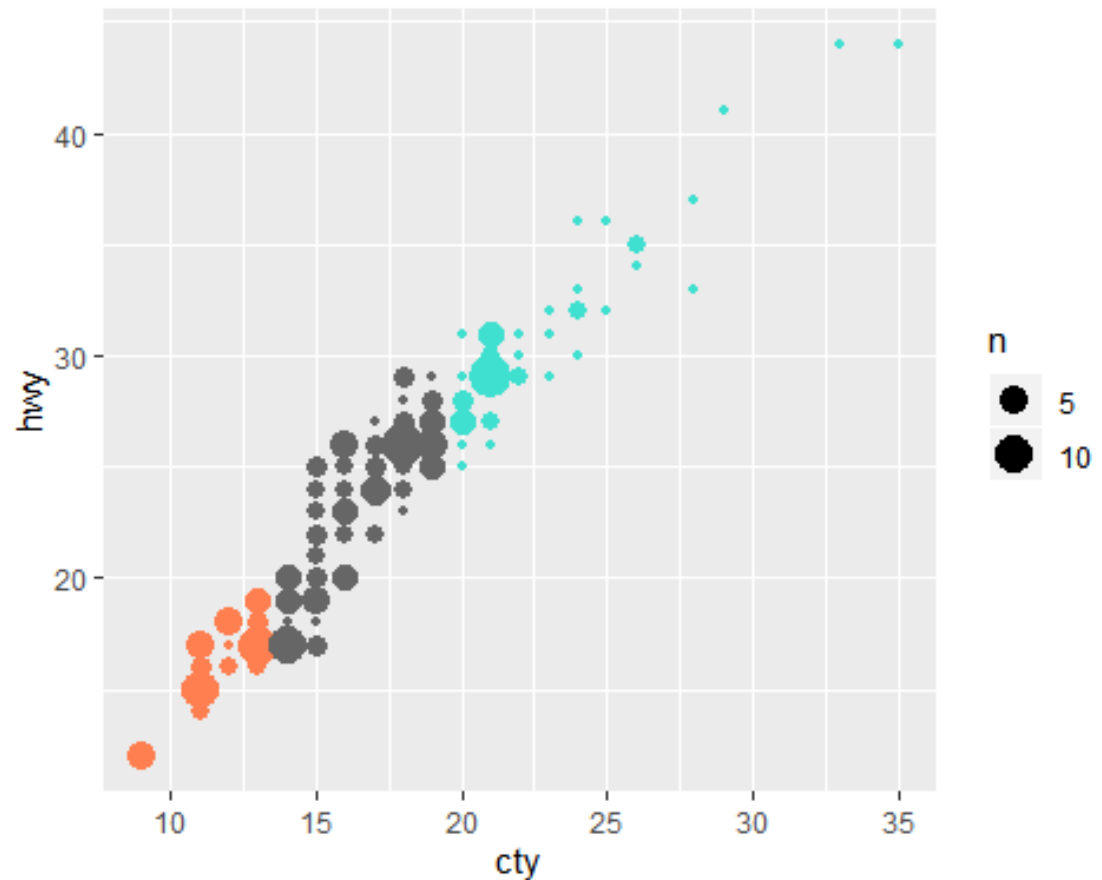
Sometimes I want to have better control over colors & sizes.

Here, I am hard coding the colors

```
df <-  
  mpg %>%  
  mutate(category =  
    case_when(  
      cty < 14 ~ "coral",  
      cty > 19 ~ "turquoise",  
      TRUE ~ "grey40"  
    )  
  )
```

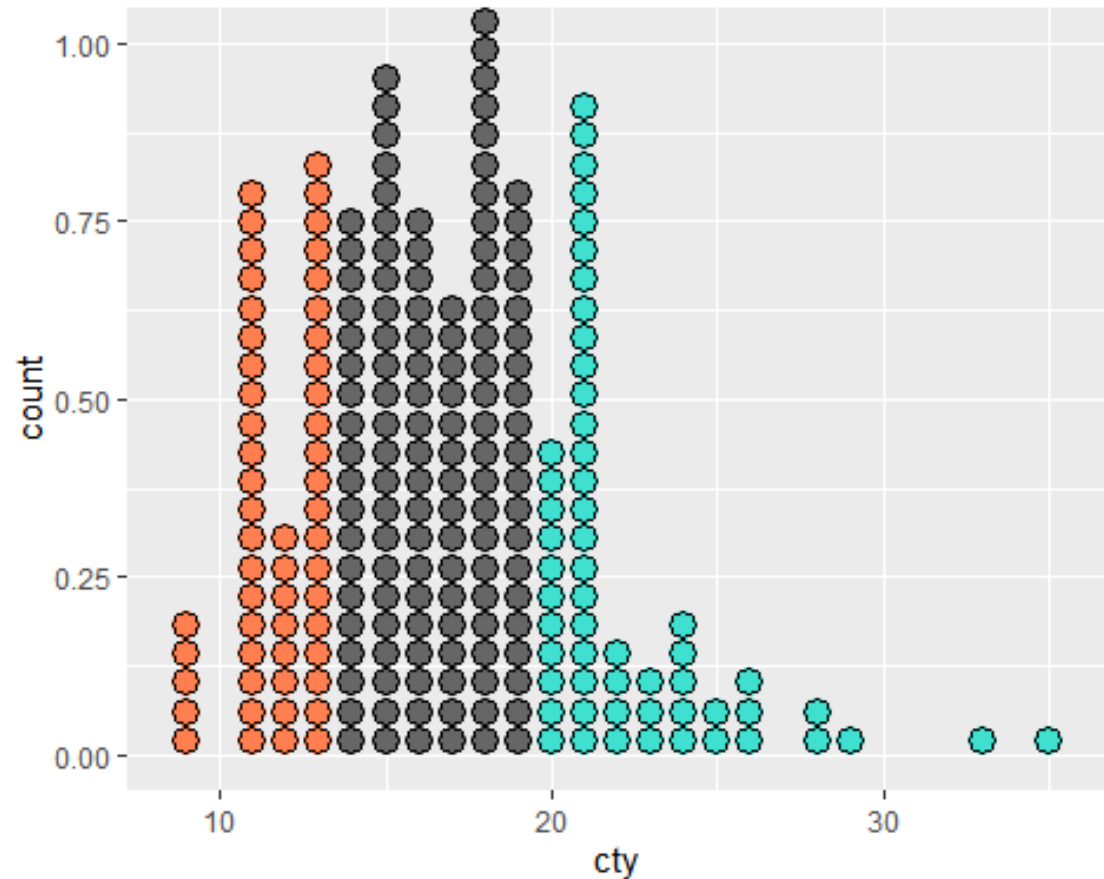
# scale\_color\_identity()

```
ggplot(df, aes(cty, hwy, color = category)) +  
  geom_count() +  
  scale_color_identity()
```



# scale\_fill\_identity()

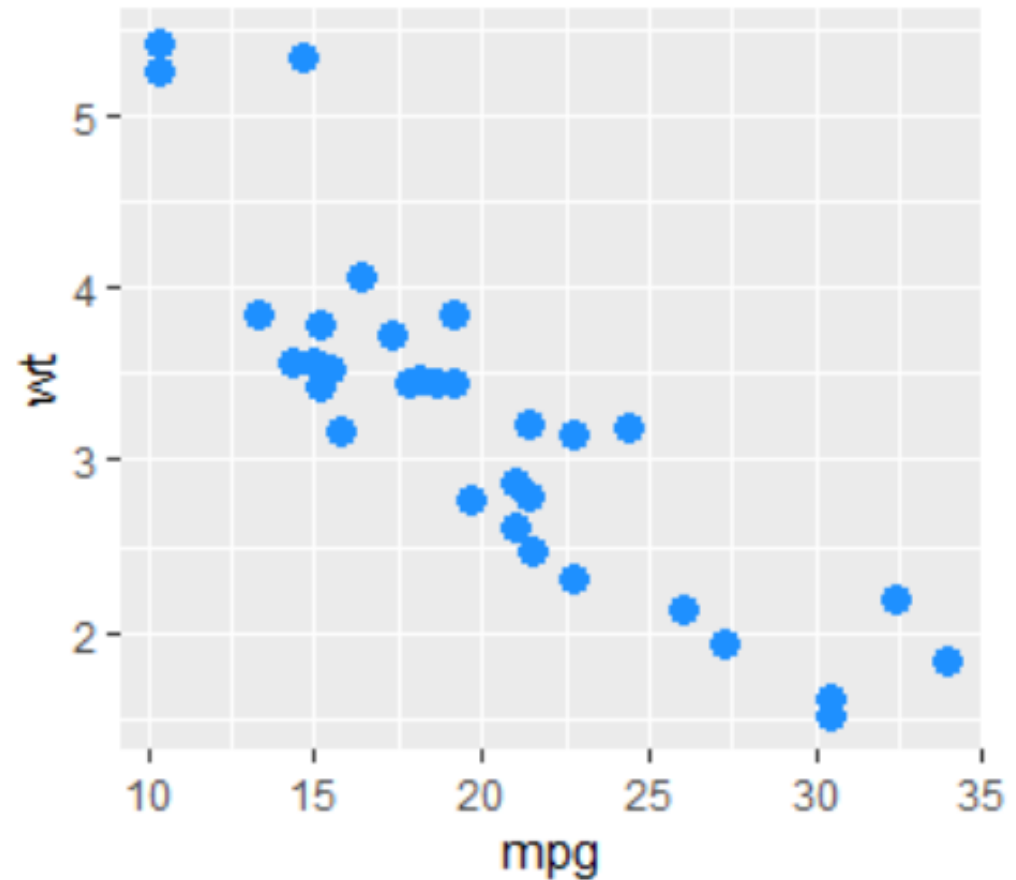
```
ggplot(df, aes(cty, fill = category)) +  
  geom_dotplot() +  
  scale_fill_identity()
```



# Default colors

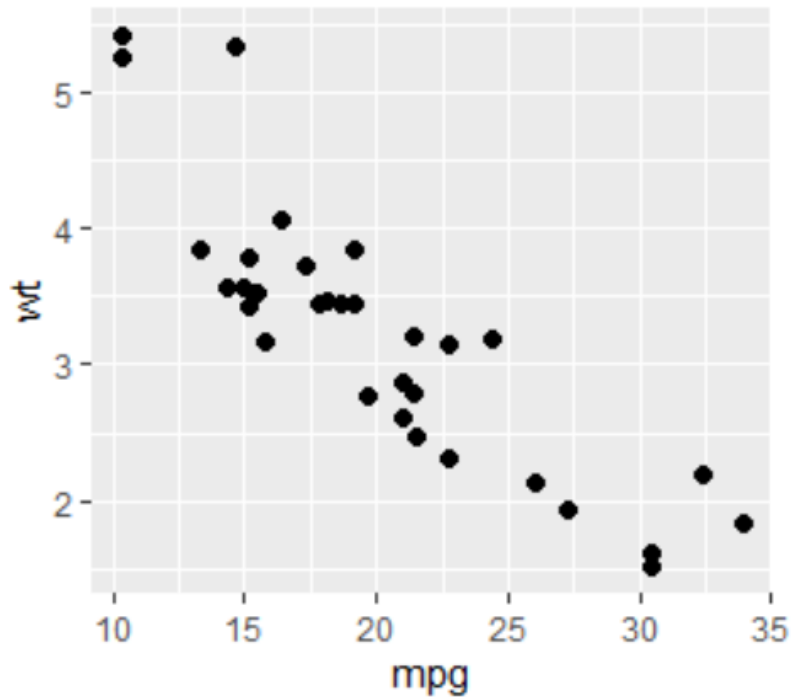
```
update_geom_defaults(  
  "point",  
  list(color = "dodgerblue", size = 3)  
)
```

```
ggplot(mtcars, aes(mpg, wt)) +  
  geom_point()
```

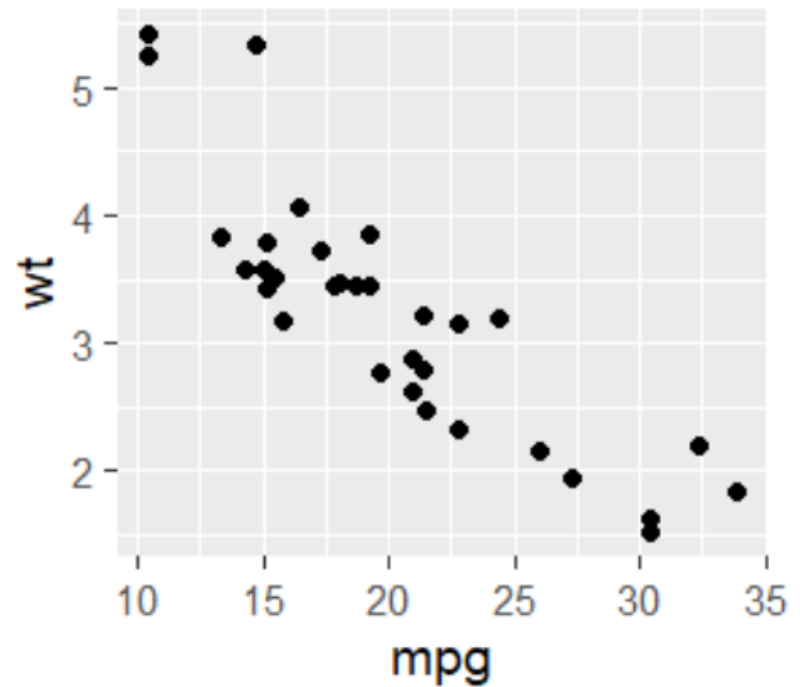


# Base font size

```
ggplot(mtcars, aes(mpg, wt)) +  
  geom_point()
```



```
ggplot(mtcars, aes(mpg, wt)) +  
  geom_point() +  
  theme_gray(base_size = 14)
```

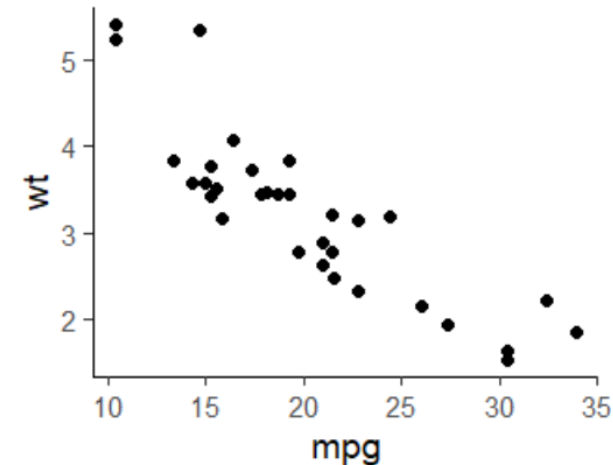




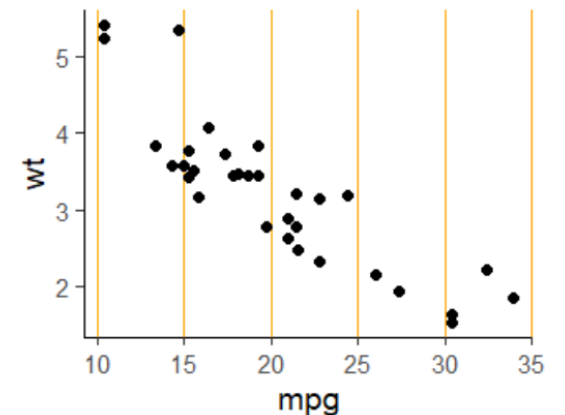
# Default theme

```
# will apply to all future charts  
theme_set(theme_classic(base_size = 14))
```

```
ggplot(mtcars, aes(mpg, wt)) +  
  geom_point()
```



```
# can update with theme()  
ggplot(mtcars, aes(mpg, wt)) +  
  geom_point() +  
  theme(  
    panel.grid.major.x = element_line(color = "orange")  
  )
```



```
# can revert back using this code  
theme_set(theme_gray())
```

**Best practices**

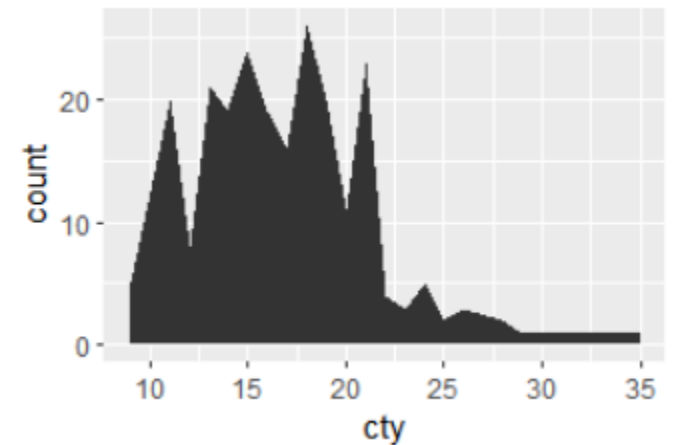
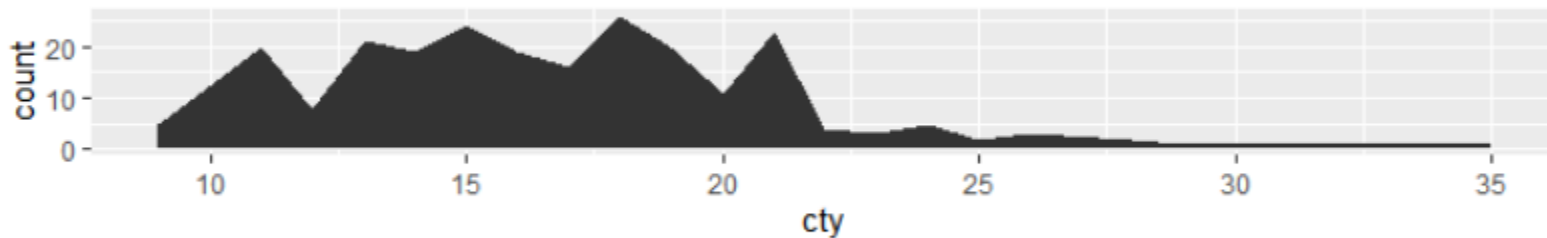
# The golden ratio 1:1.6

- Try to **give your charts the proportion of a credit card**
- Also look this up

```
p <- ggplot(mpg, aes(cty)) + geom_bar()
```

```
p + coord_fixed(1/10)
```

```
p + theme(aspect.ratio = 1/1.6) # ratio depends on the units
```

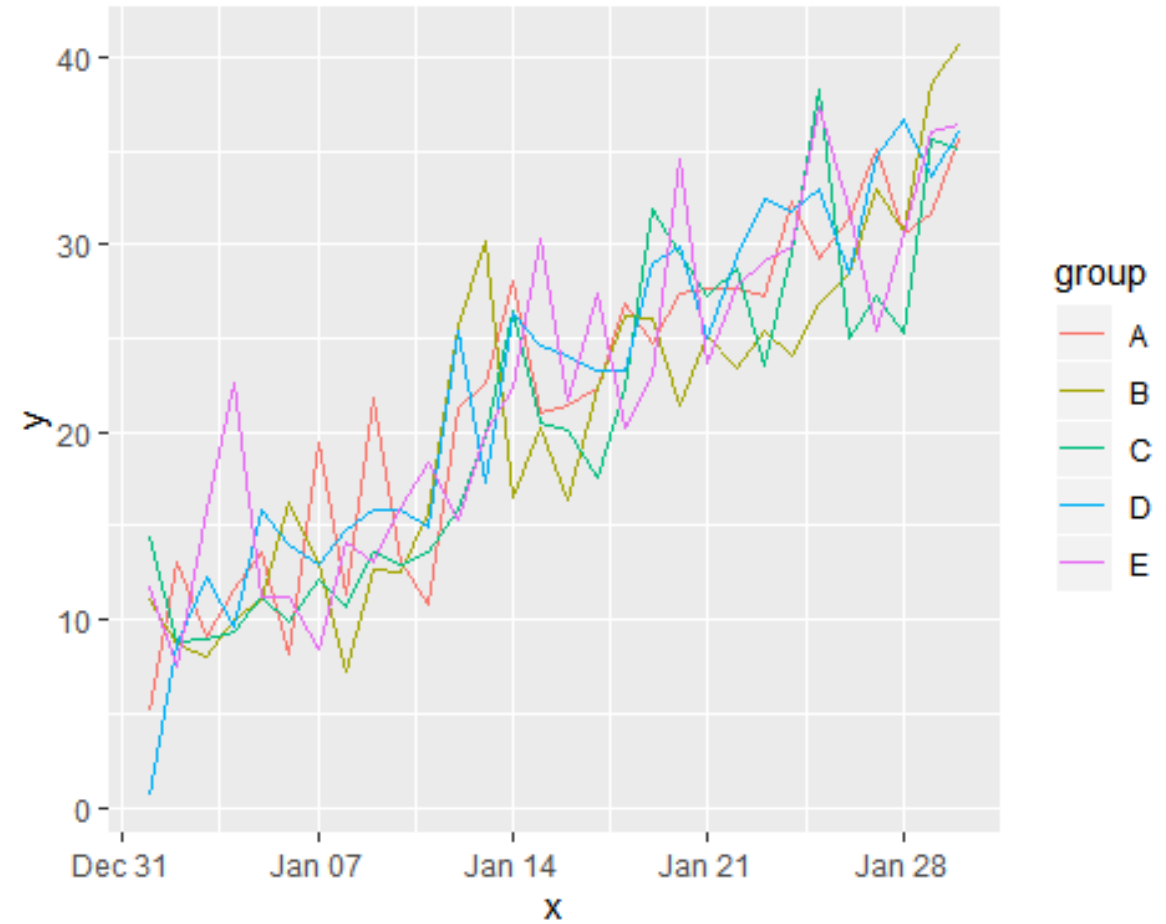


# Dealing with spaghetti charts

This is one of the most common questions:  
**change** for **multiple categories** over **time**

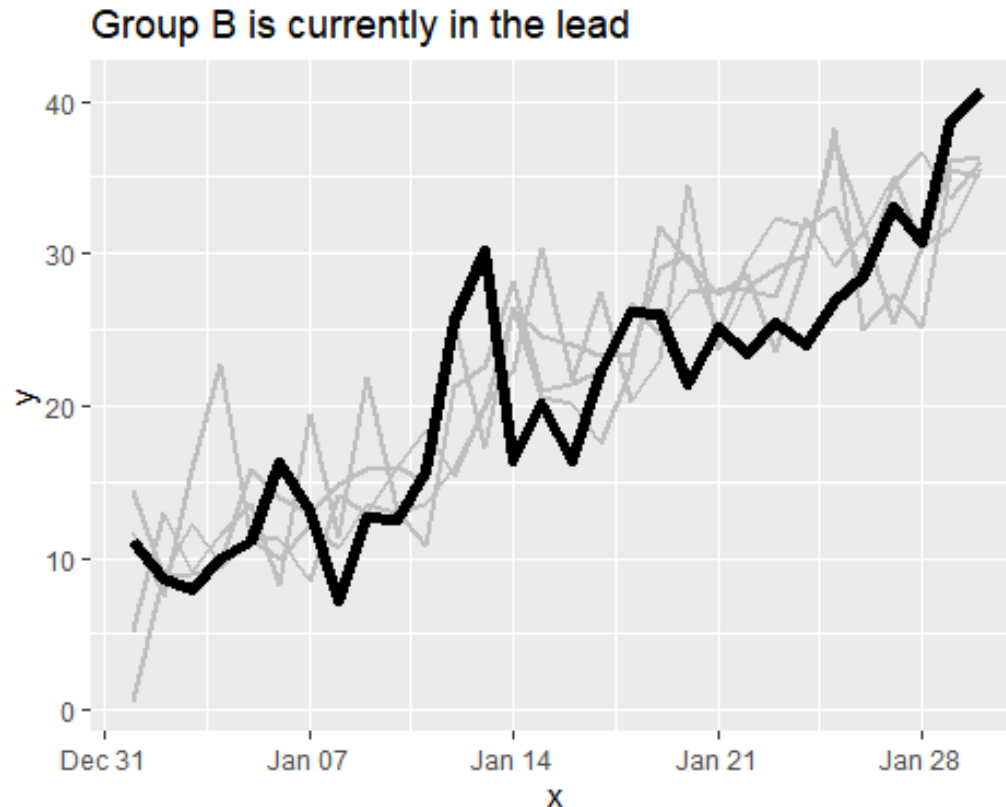
This often results in a chart like the one here. It is hard to read but there are some ways you can **help your audience**

```
ggplot(df, aes(x, y, color = group)) +  
  geom_line()
```



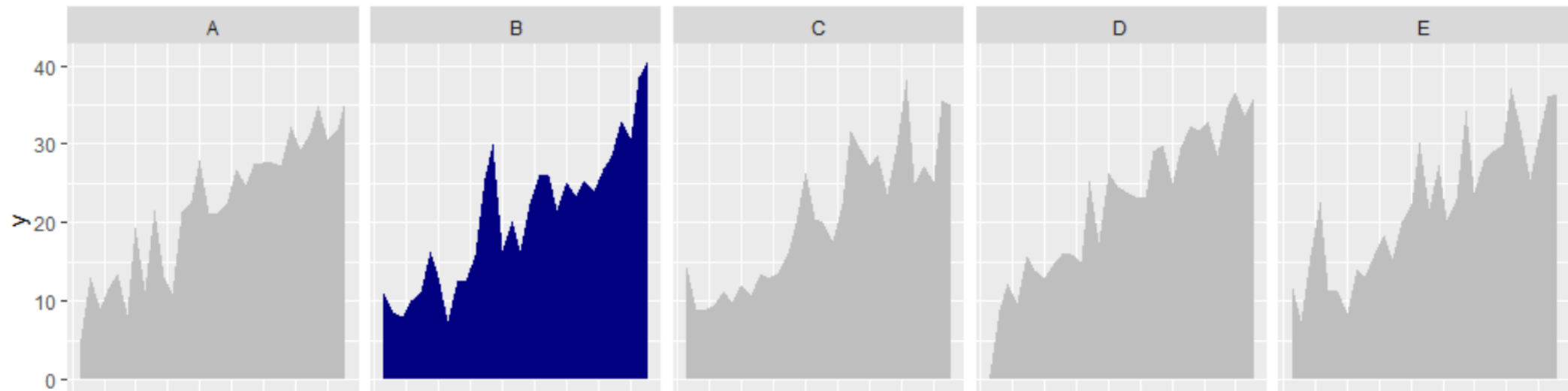
# Highlight the focus & use an informative title

```
ggplot(df, aes(x, y, group = group)) +  
  geom_line(data = filter(df, group != "B"), color = "grey", size = 1) +  
  geom_line(data = filter(df, group == "B"), color = "black", size = 2) +  
  labs(title = "Group B is currently in the lead")
```



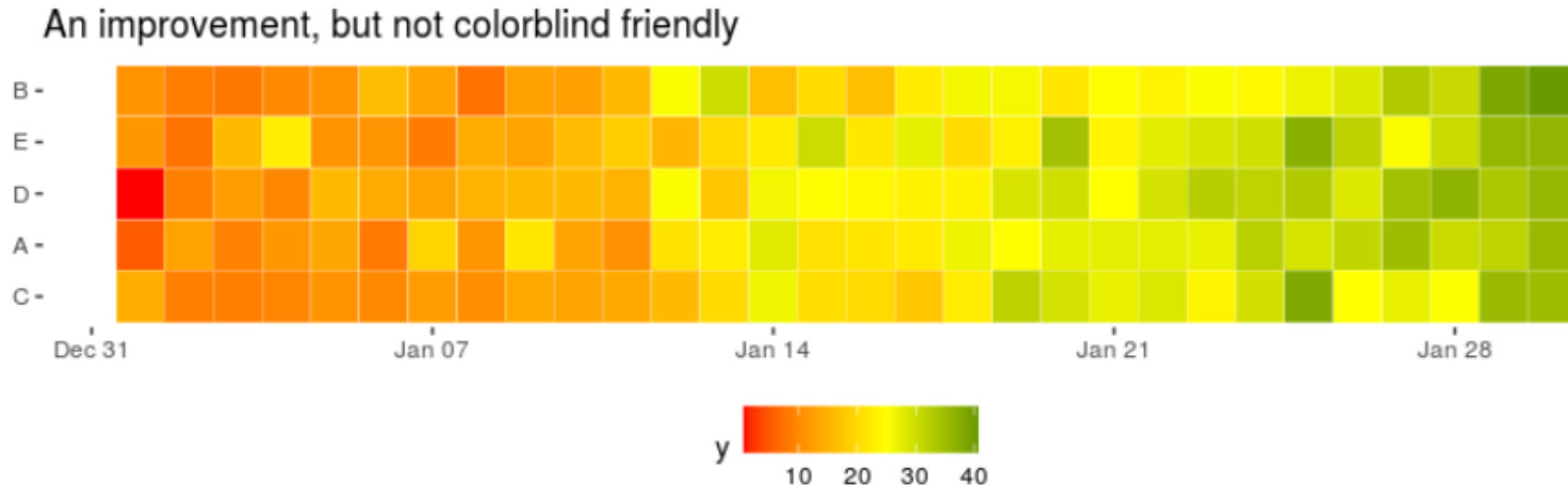
# Use facets

```
ggplot(df, aes(x, y, fill = (group == "B"))) +  
  geom_area() +  
  facet_grid(cols = vars(group)) +
```



# Try a heatmap but beware

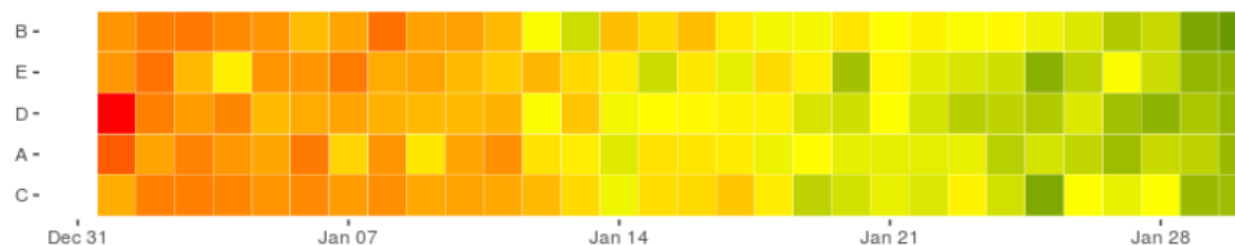
```
ggplot(df, aes(x, fct_reorder(group, y, last), fill = y)) +  
  geom_tile(color = "white") +  
  scale_fill_gradient2(  
    low = "red", mid = "yellow", high = "darkgreen", midpoint = 25  
  ) +  
  my_theme +  
  labs(title = "An improvement, but not colorblind friendly")
```



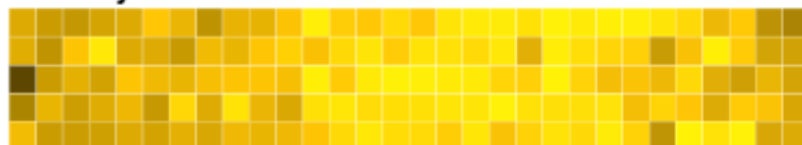
```
#remotes::install_github("clauswilke/colorblindr")
```

```
colorblindr::cvd_grid(h + theme_void())
```

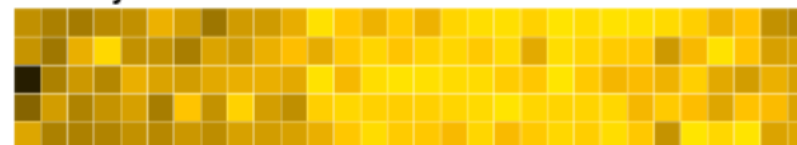
An improvement, but not colorblind friendly



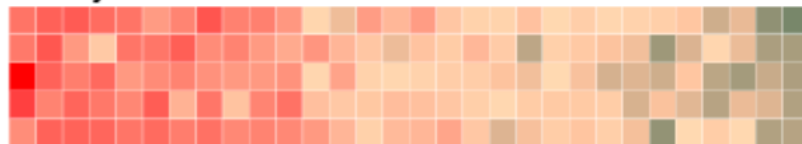
**Deutanomaly**



**Protanomaly**



**Tritanomaly**

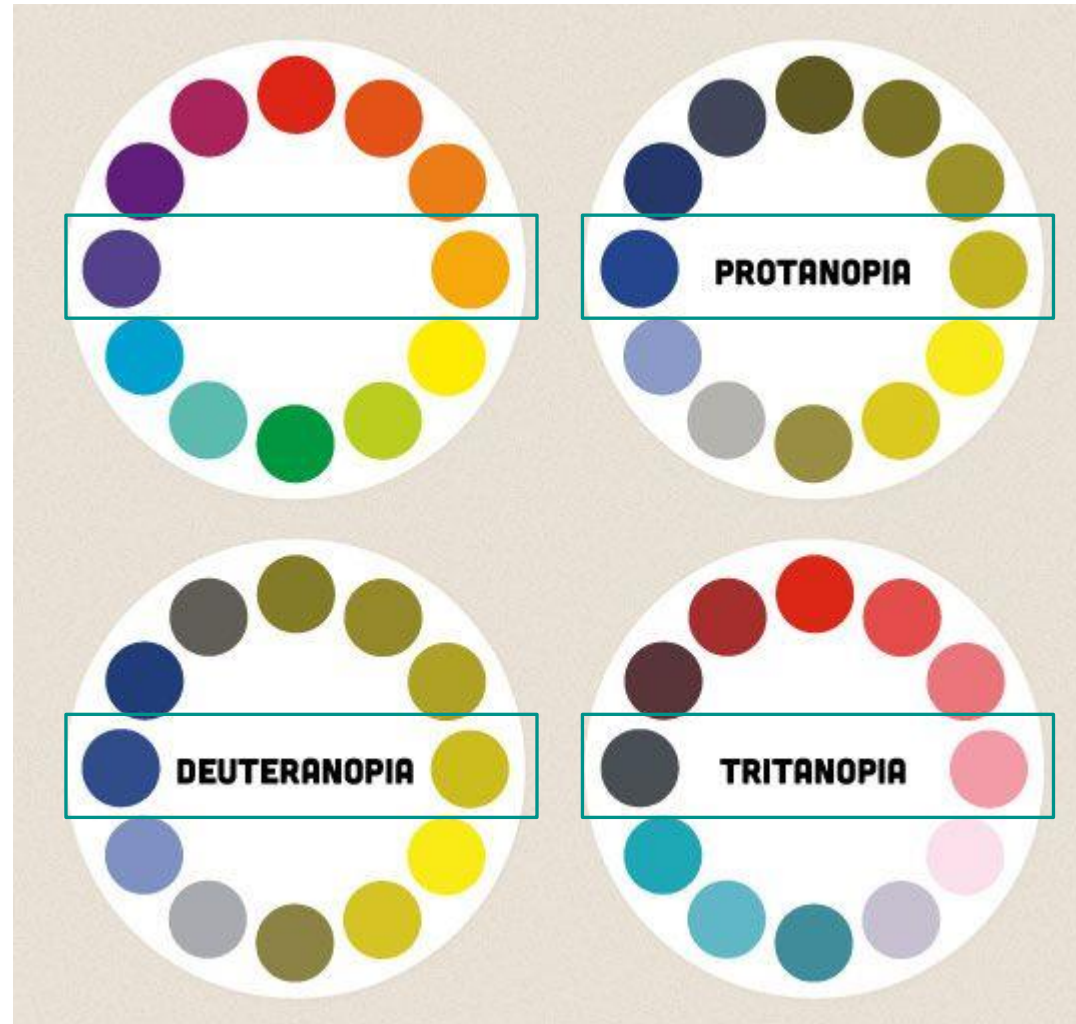


**Desaturated**





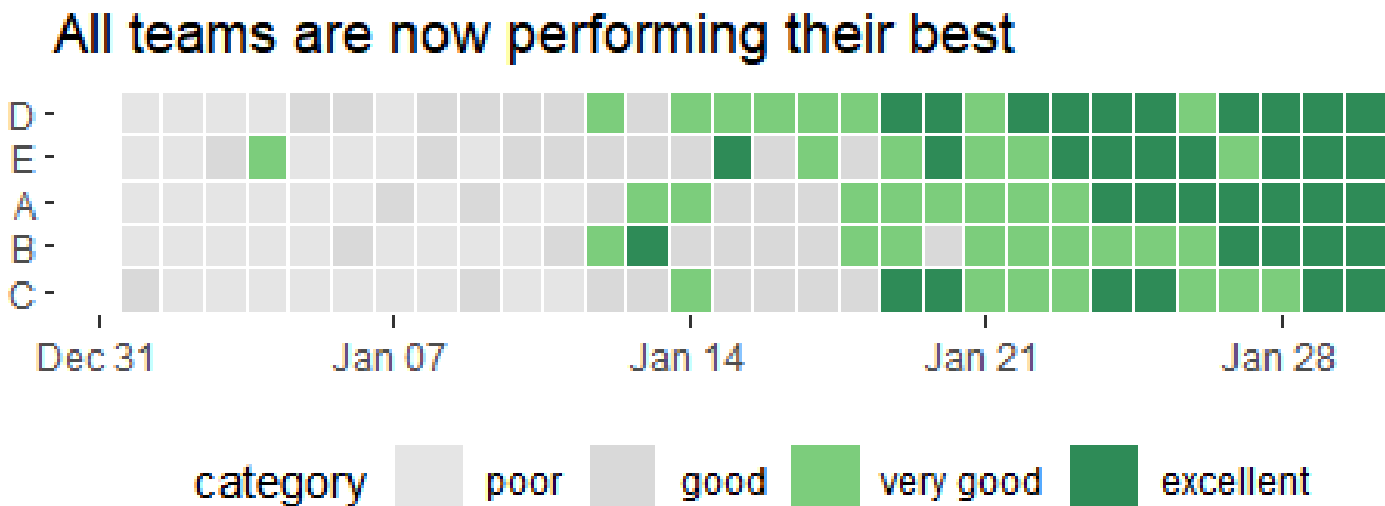
# Blue & Orange



# Not every point needs a color

```
ggplot(df, aes(x, fct_reorder(group, y), fill = category)) +  
  geom_tile(color = "white", size = 0.1) +  
  scale_fill_manual(  
    values = c("grey90", "grey85", "palegreen3", "seagreen4"),  
    labels = c("poor", "good", "very good", "excellent")  
  ) +  
  my_theme +  
  labs(title = "All teams are now performing their best")
```

You can see this in the  
code sample:  
**category = factor(ntile(y, 4))**



# Take care when cropping data

The usual methods to “zoom in” can yield unexpected results when `stat_*()` geoms are used.

For example, `geom_boxplot()` calls `stat_boxplot()`

- filters out data outside the coordinates **before** doing the stats
- the boxplot will keep readjusting the quartiles

# Use `coord_cartesian()` to zoom in

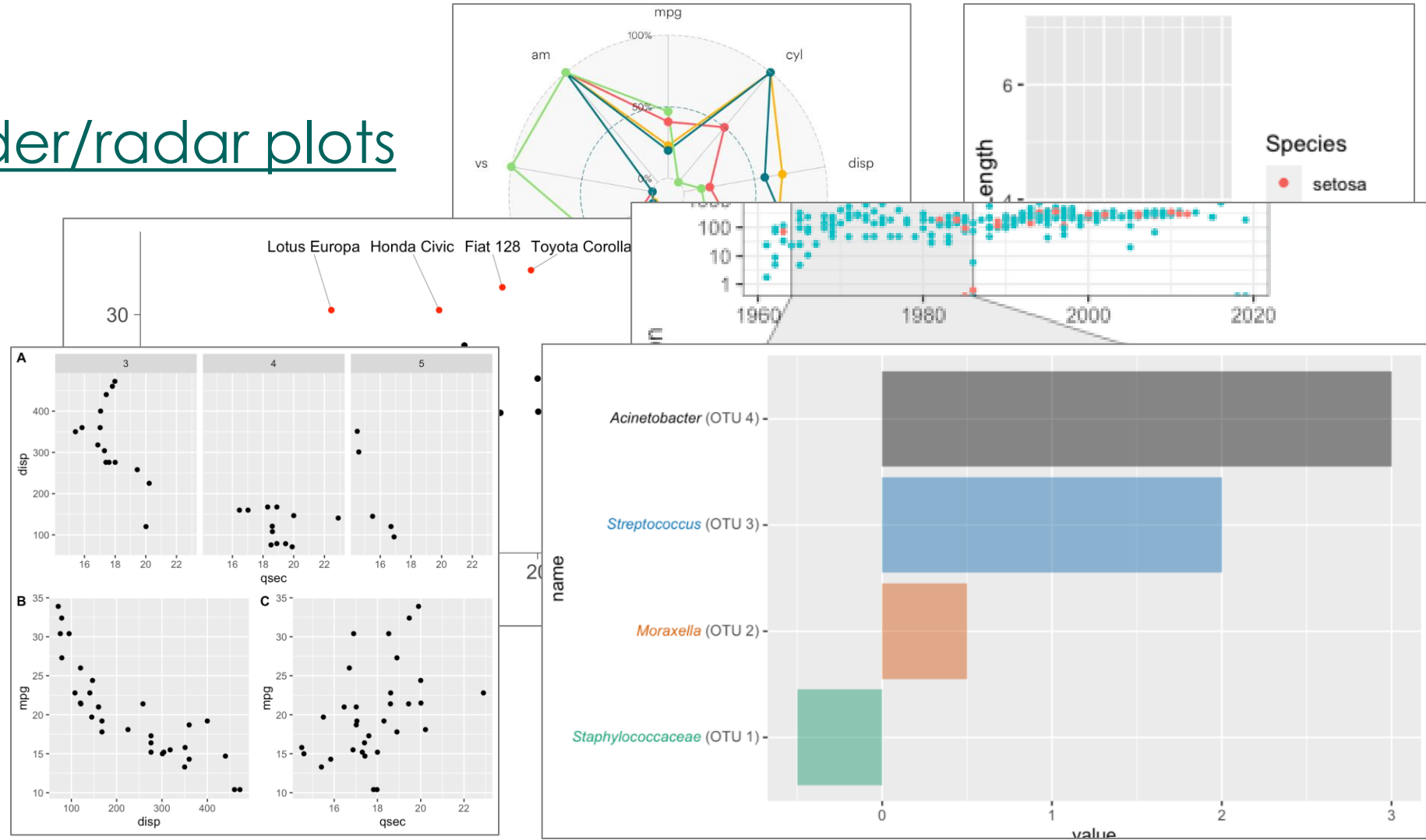
Use `coord_cartesian()` instead of `ylim()` or `scale*_continuous()` on summary stats

```
# find_limits() is a custom function
bind_rows(
  find_limits(p),
  find_limits(p + ylim(0, 12000)),
  find_limits(p + scale_y_continuous(limits = c(0, 12000))),
  find_limits(p + coord_cartesian(ylim = c(0, 12000)))
)
```

	<u>lower</u>	<u>middle</u>	<u>upper</u>
##	950	2401	5324
##	911	2161	4679
##	911	2161	4679
##	950	2401	5324

# Extensions

- ggradar - spider/radar plots
- gganimate
- ggrepel
- ggforce
- cowplot
- ggtext



# addinlist

```
addinlist::addinlistAddin()  
p <-  
  ggplot(mpg, aes(cty, hwy)) +  
  geom_point()
```

*# esquisse*

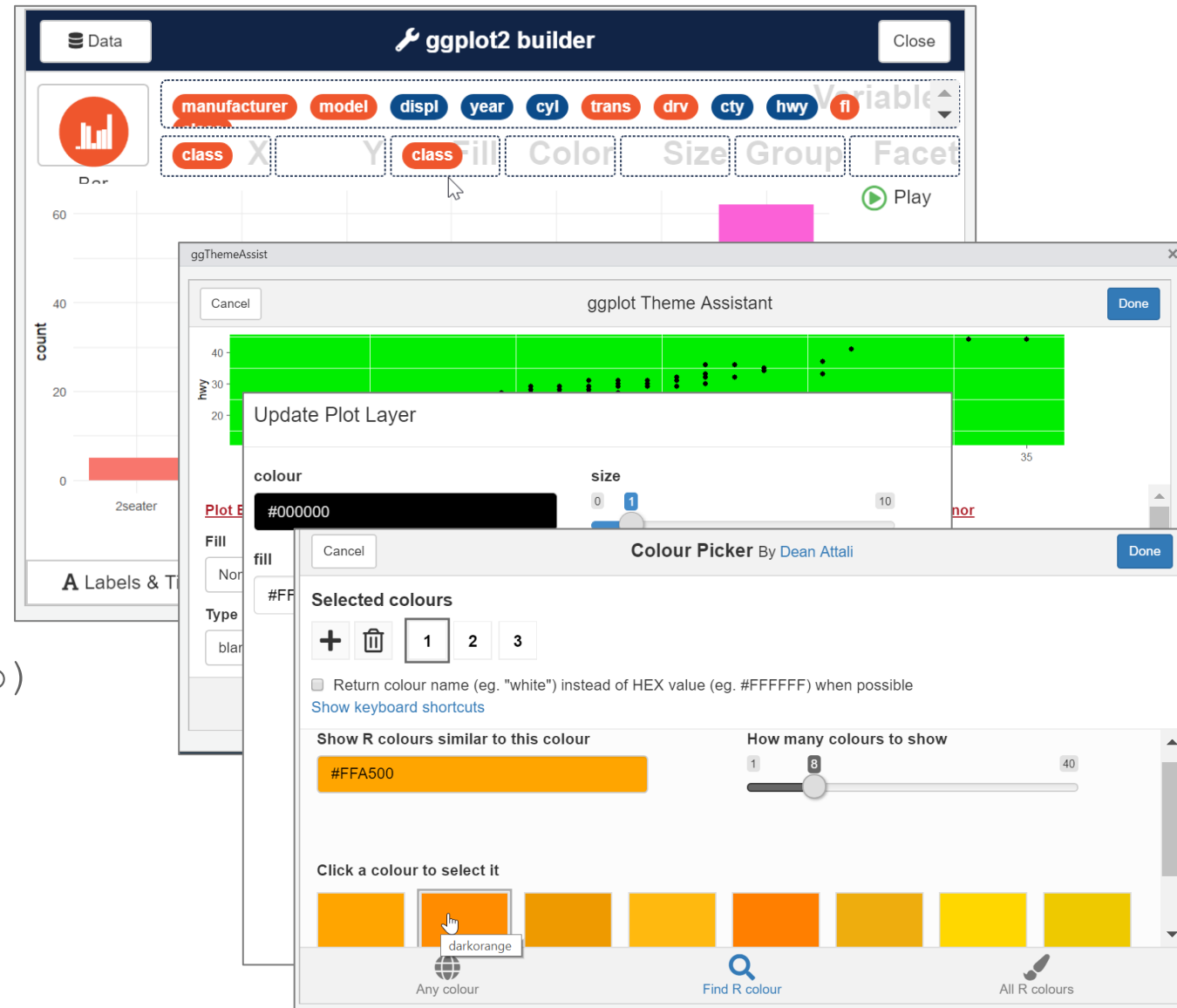
```
esquisse:::esquisser()  
esquisse:::esquisser(mpg)
```

*# others*

```
ggThemeAssist::ggThemeAssistGadget(p)
```

```
ggedit::ggedit(p)
```

```
colourpicker::colourPicker()
```



# simplecolors

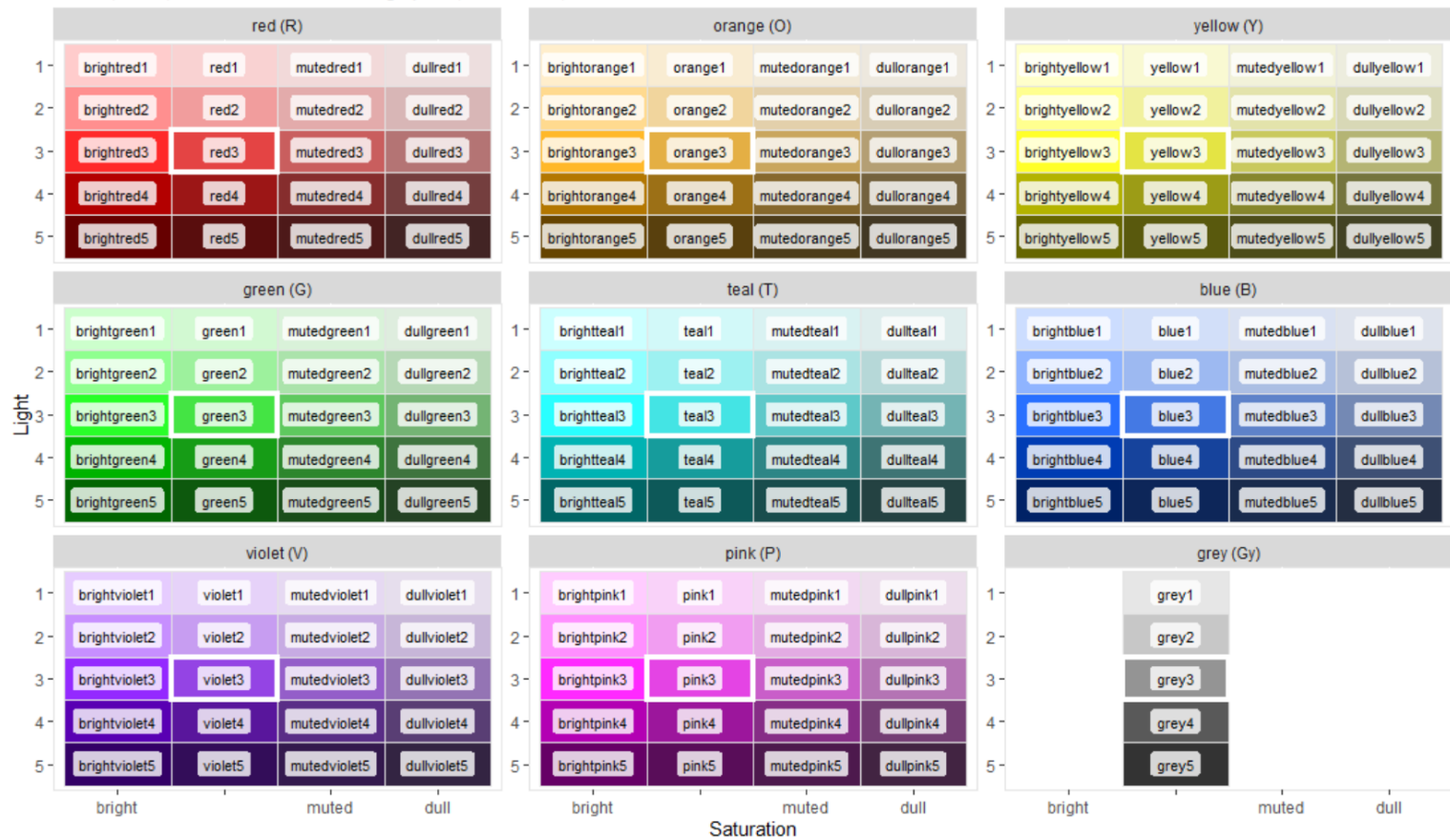
<https://rjake.github.io/simplecolors/articles/intro.html>

```
# devtools::install_github("rjake/simplecolors")
```

```
library(simplecolors)
show_colors(labels = TRUE)
```



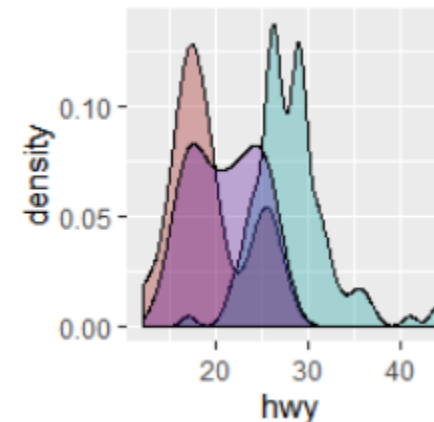
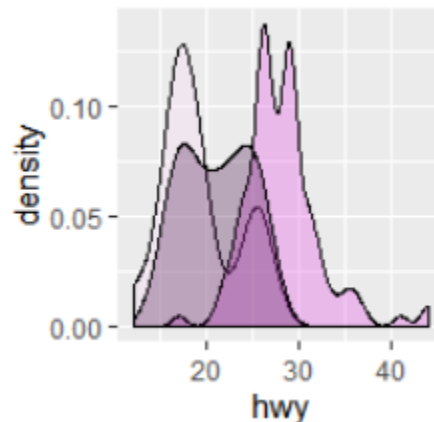
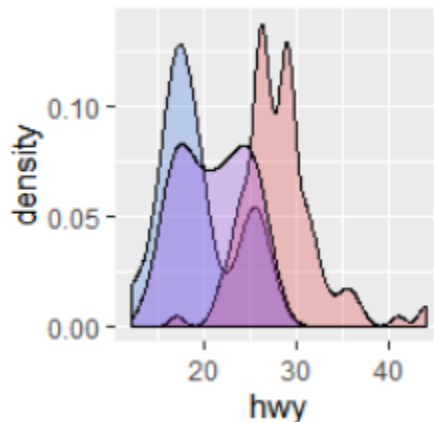
The default is a lightness of 3 and can be specified by color name alone  
ex. red, violet, teal or with modifiers: brightpink2, mutedred3, blue4



# simplecolors

Three main functions: `sc()` `sc_across()` `sc_*()`

```
p <-  
  ggplot(mpg, aes(hwy, fill = drv)) +  
  geom_density(alpha = 0.3)  
  
p + scale_fill_manual(values = sc("blue3", "red3", "violet3"))  
p + scale_fill_manual(values = sc_pink(light = c(1, 3, 5)))  
p + scale_fill_manual(values = sc_across("RTV", light = 4, sat = "bright"))
```





# Appendix

# R4DS

**R for Data Science** is a book all about the **tidyverse**. It is less “data science-y” and more about data manipulation and visualization. It is free online [here](#) as well as available for sale.

# Stackoverflow

- try **datapasta** for a minimal reпреx
- include images rather than links
- incorporate **styler**

# Cheatsheets

<https://rstudio.com/resources/cheatsheets/>

ggplot2



forcats

