# Data Visualization in R

https://bit.ly/2Wt7nkQ

Jake Riley - June 19, 2019

# Today's talk

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

- https://bit.ly/2Wt7nkQ

# An intro

- Jake Riley
- Clinical Data Analyst at Children's Hospital of Philadelphia
- Avid **`ggplot2`** answerer on stackoverflow
- Dogdad
- @yake_84

- https://bit.ly/2Wt7nkQ

# 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 %>% arrange(Species)          arrange(iris, Species)
```

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

# What is ggplot?

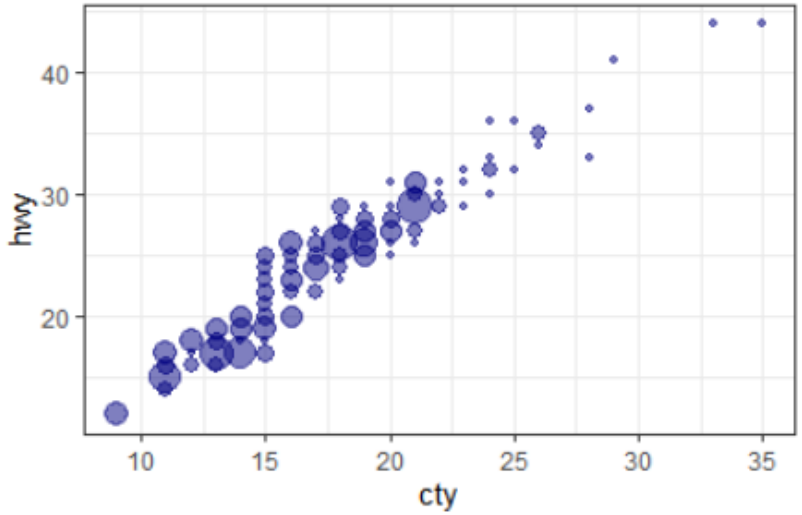
- **g**rammar of **g**raphics
- just like every sentence has a **subject, verb, and noun**, every chart has a **coordinate system, geom, and aesthetics**

- the hope is that we will invent new types of charts

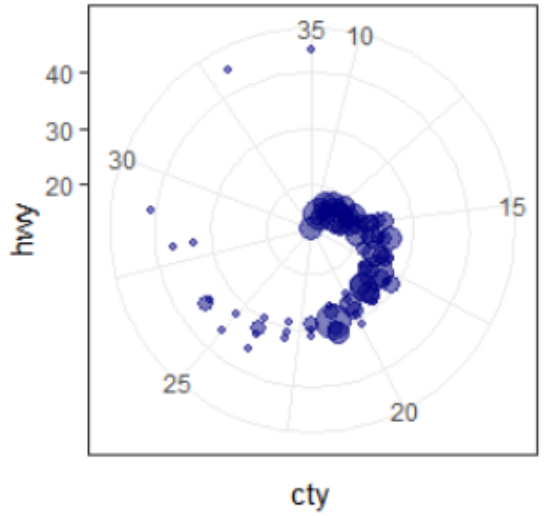
```
library(tidyverse)

p <-
  ggplot(mpg) +
  geom_count(aes(cty, hwy), alpha = 0.5, color = "navyblue") +
  theme_bw() +
  theme(legend.position = "none")
```

# an example

```
p
```




```
p + coord_polar()
```

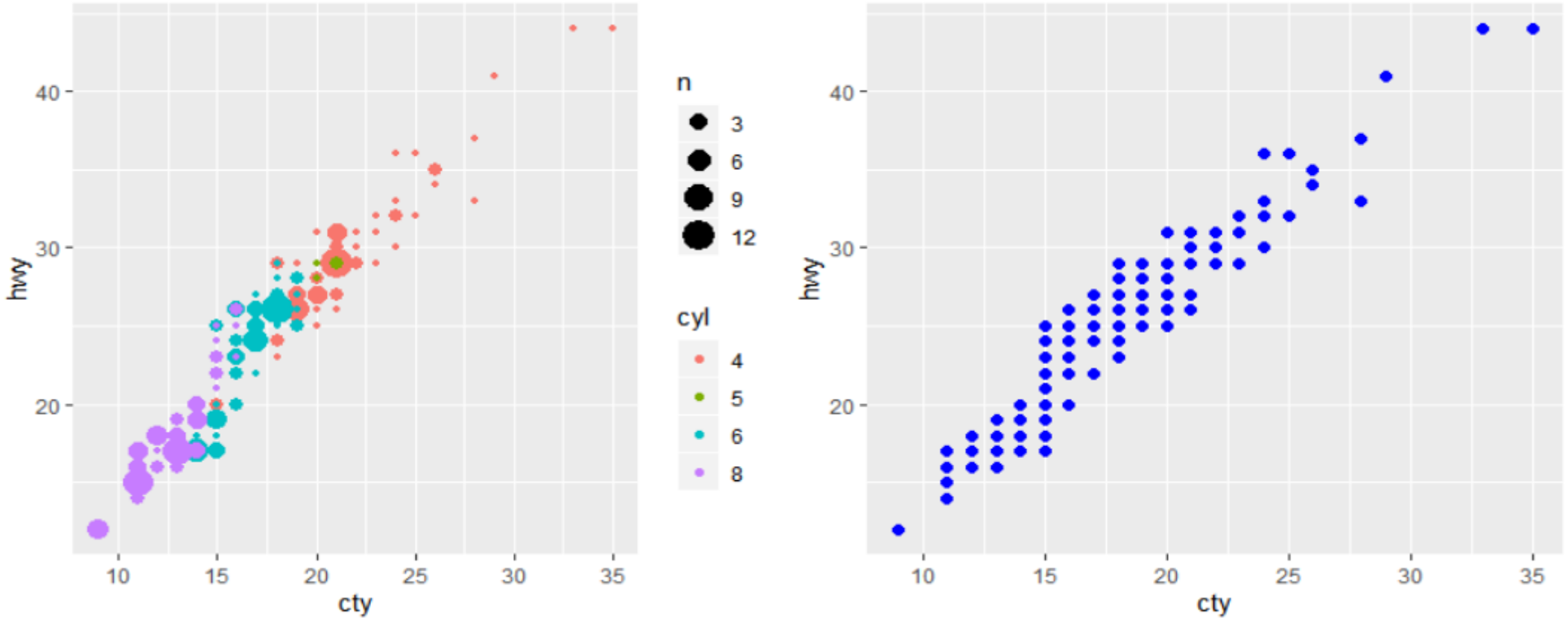


# Demystifying `aes()`

- **`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)`

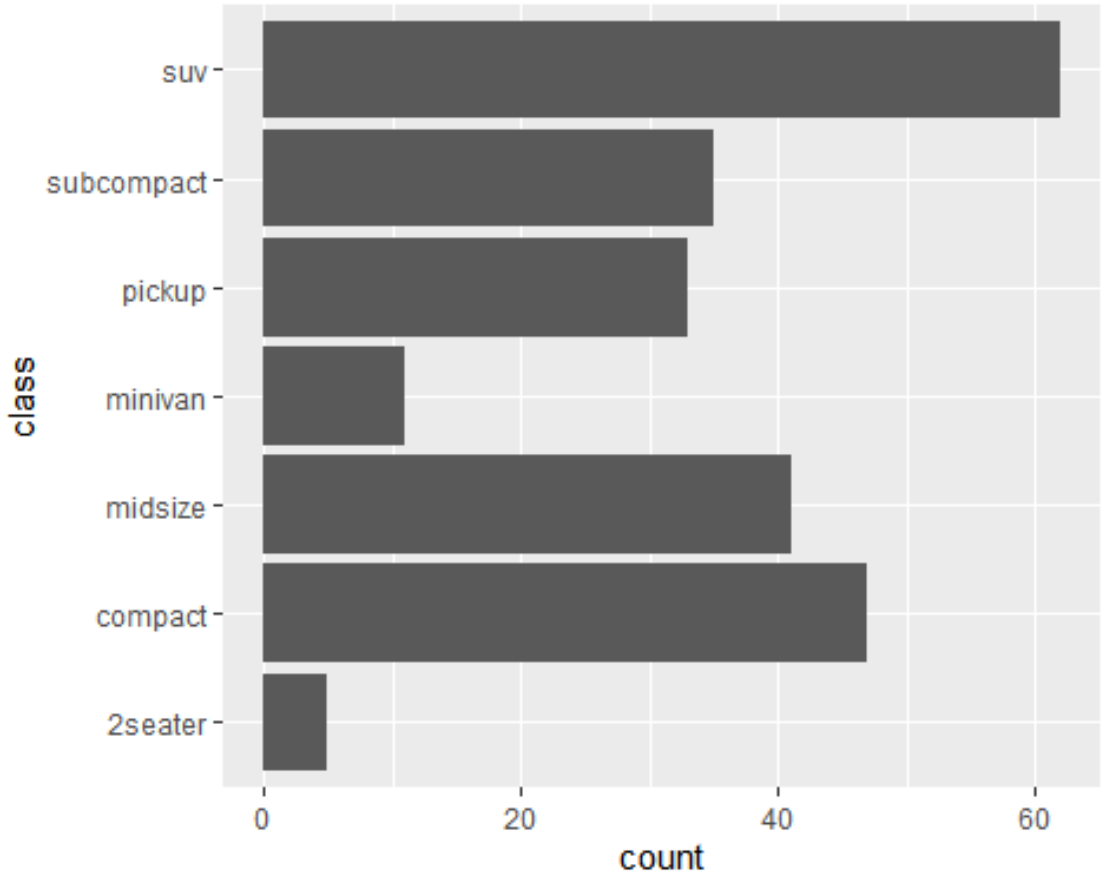


# Tips & Tricks

# Descending bar charts

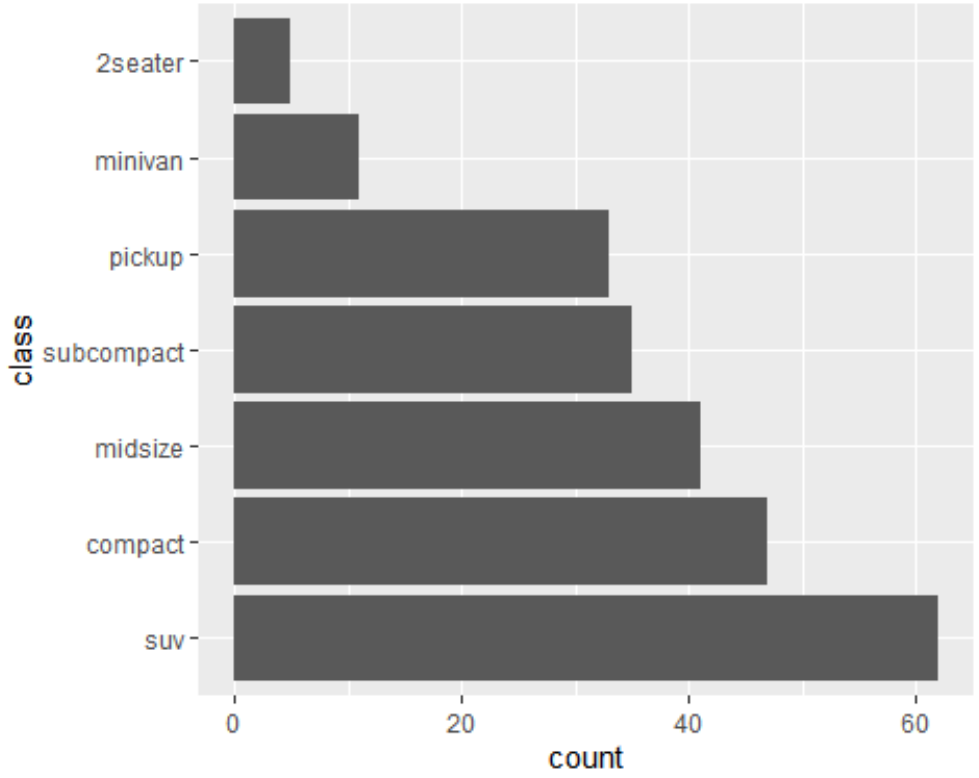
The number one things I get asked is how to make a barchart go in descending order.

```
ggplot(mpg, aes(class)) +
  geom_bar() +
  coord_flip()
```



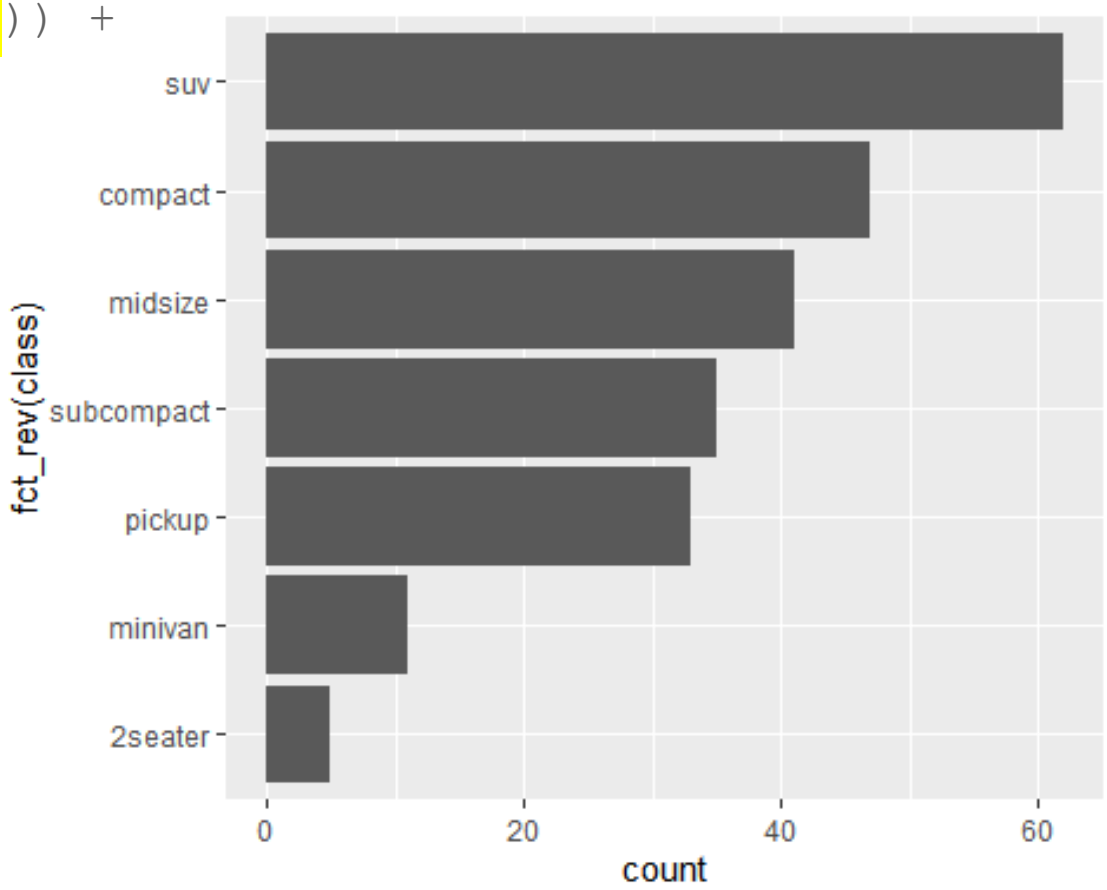
# Arrange by volume: `fct_infreq()`

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



# Arrange in descending order: `fct_rev()`

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



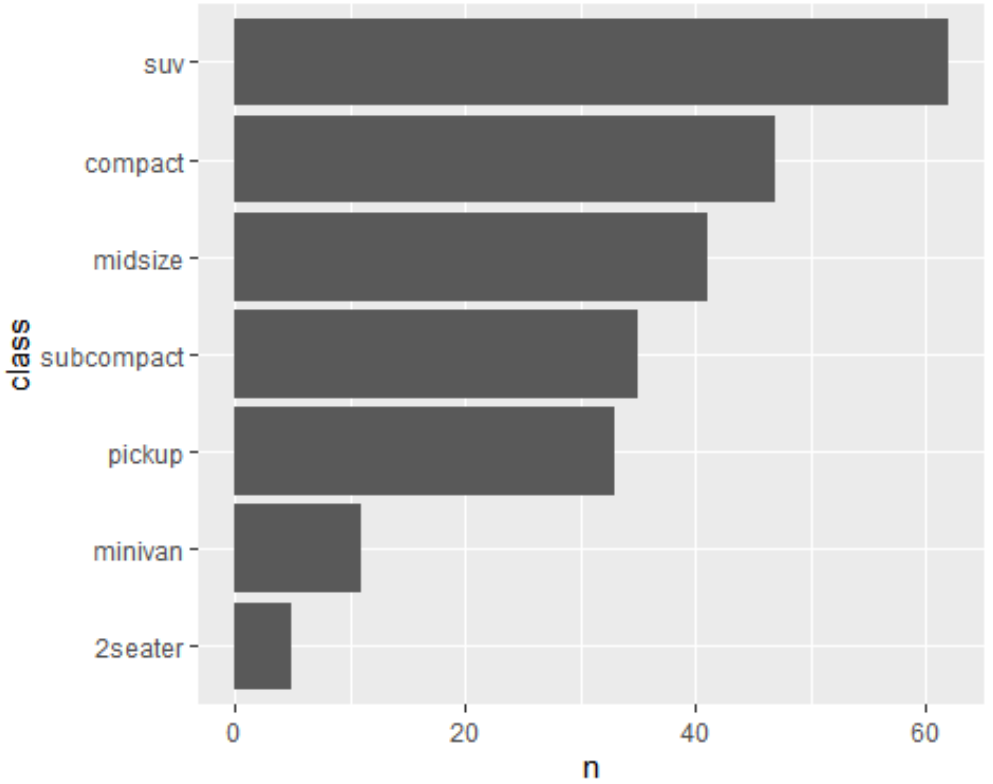
# Aggregated data: `fct_reorder()`

```
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
```

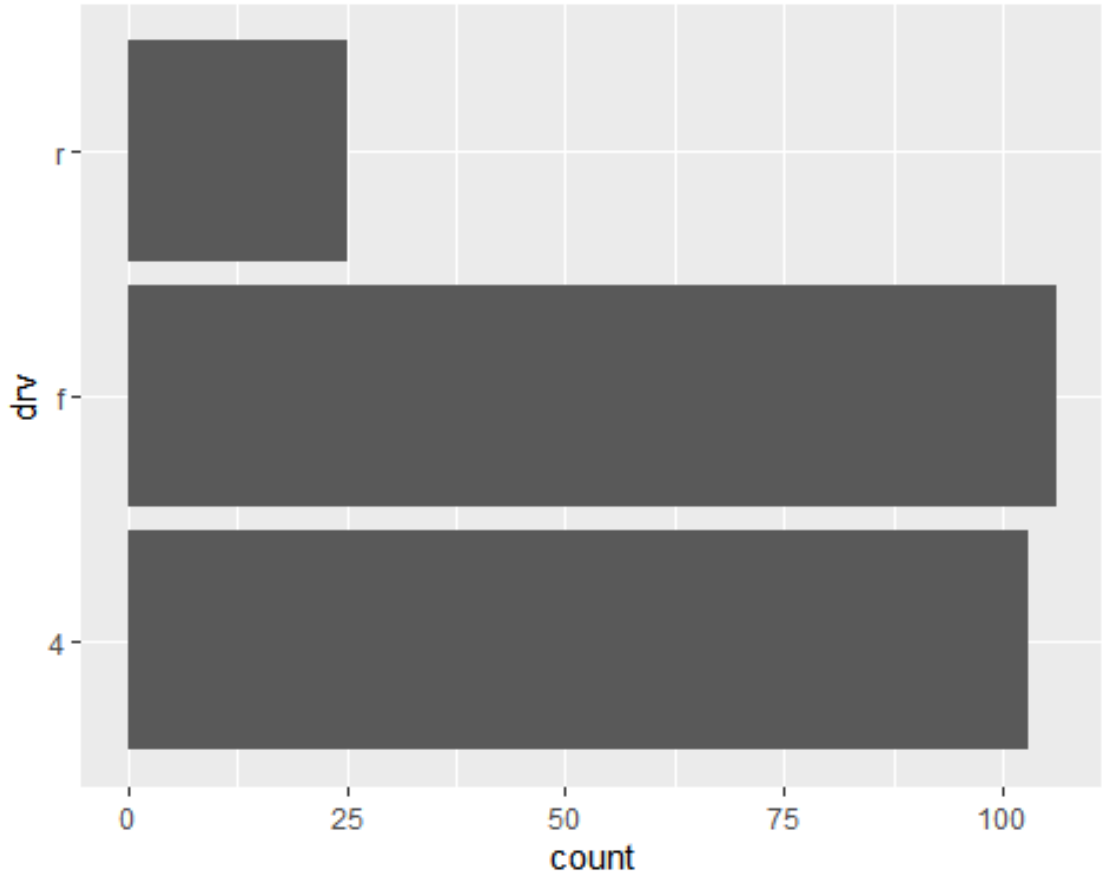
# Aggregated data: geom_col()

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



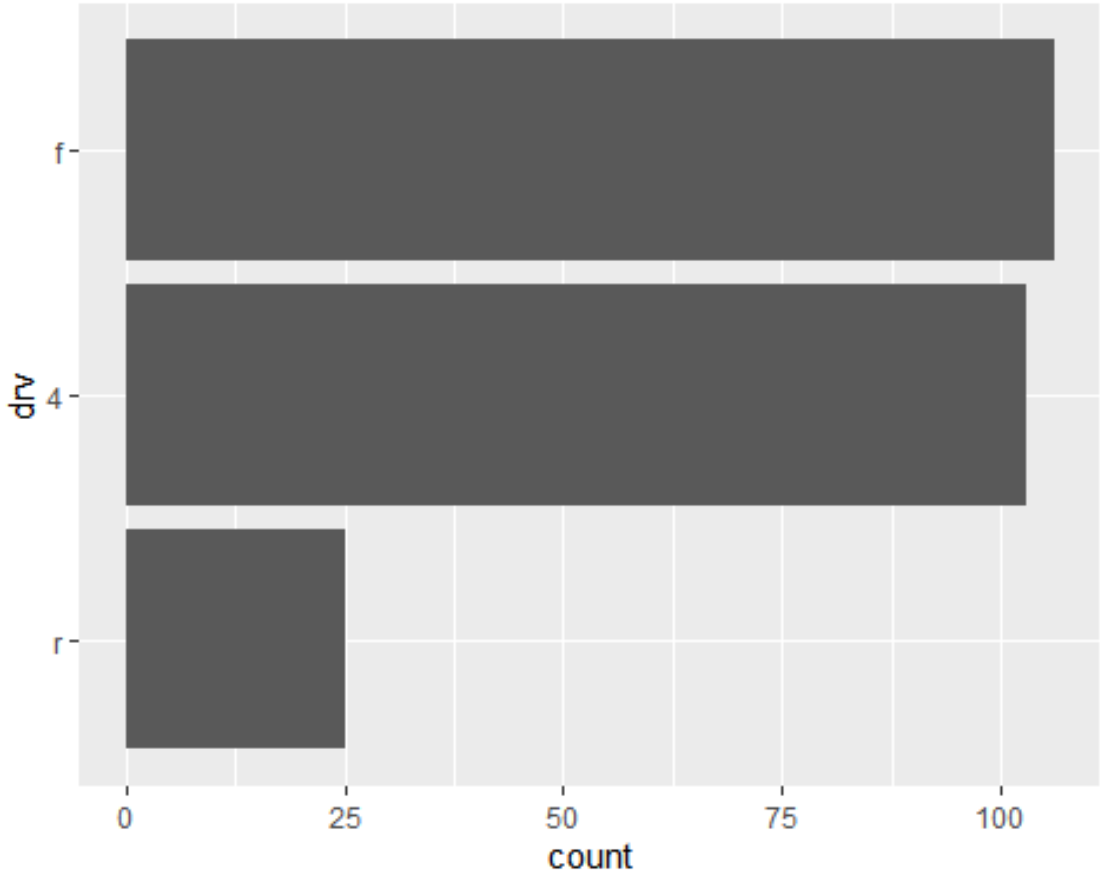
# Q1: can you put this in descending order?

```
ggplot(mpg, aes(drv)) +
  geom_bar() +
  coord_flip()
```



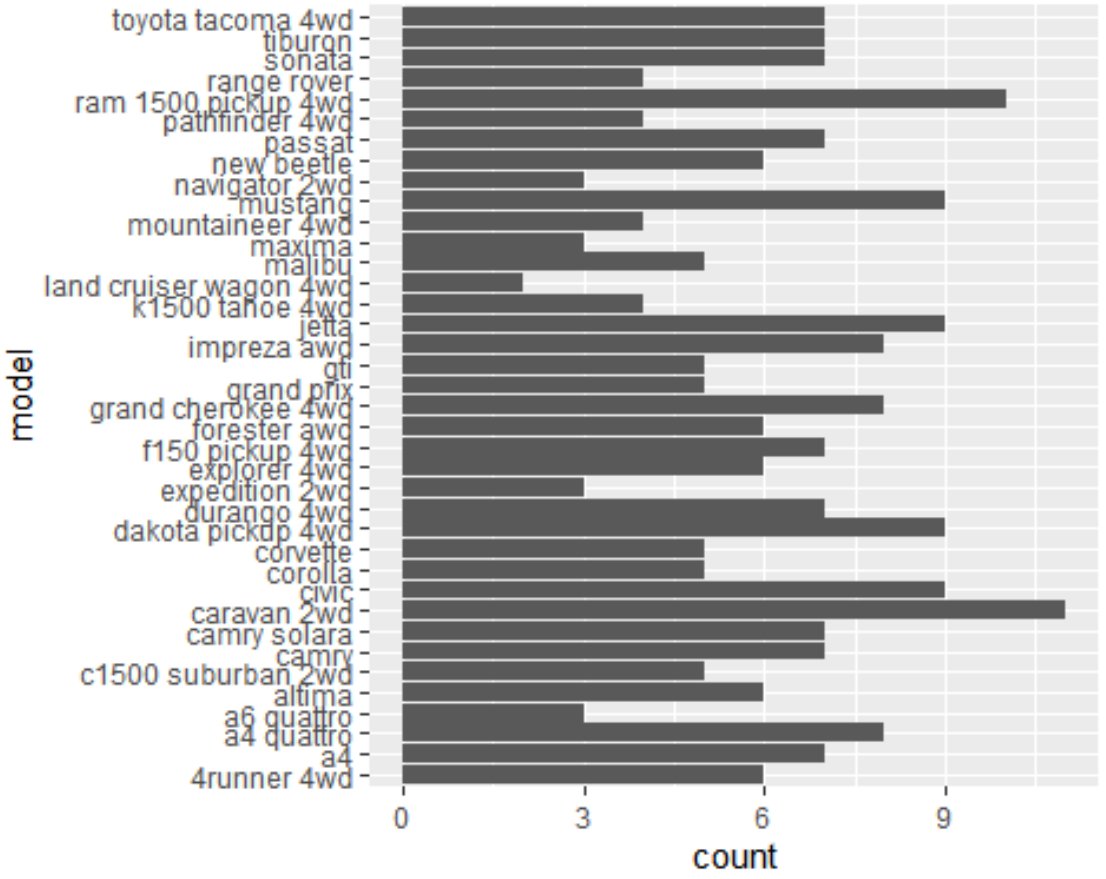
# Q1 Answer

```
mpg %>%
  mutate(
    drv = fct_infreq(drv),
    drv = fct_rev(drv)
  ) %>%
  ggplot(aes(drv)) +
  geom_bar() +
  coord_flip()
```



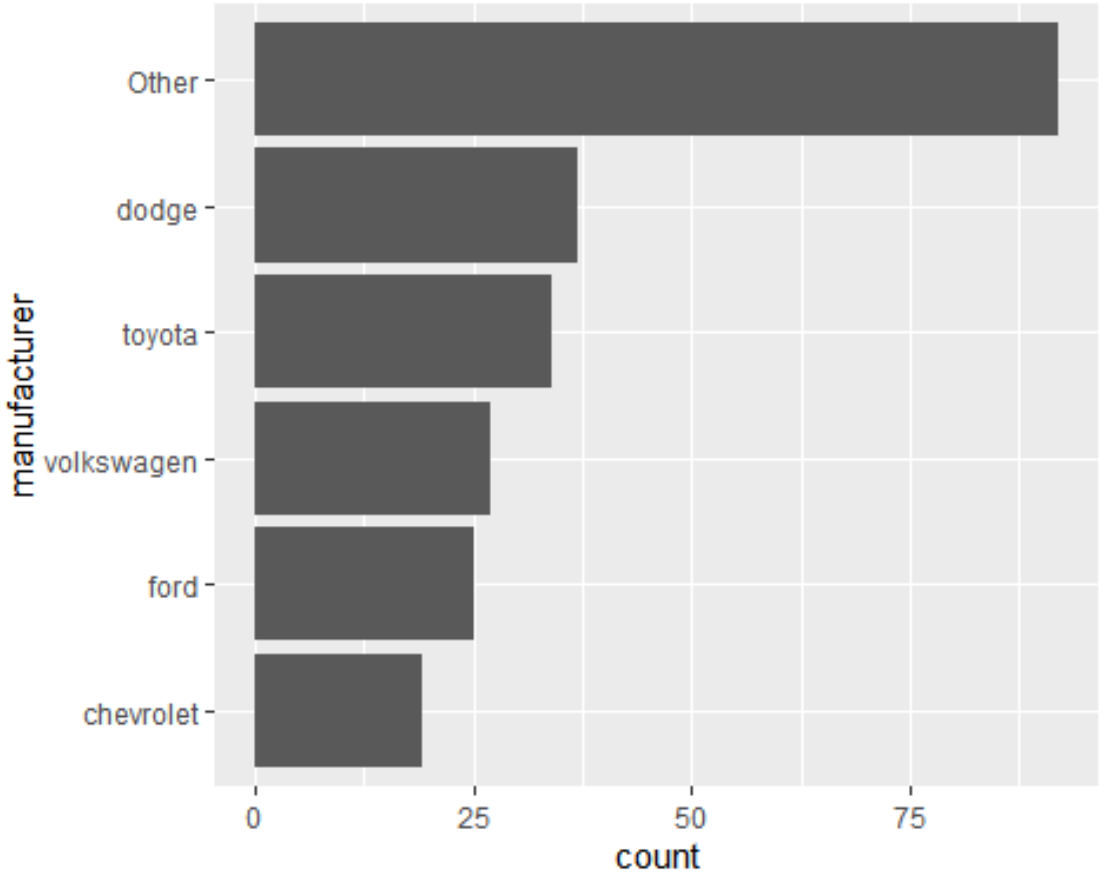
# Too many bars

```
ggplot(mpg, aes(model)) +
  geom_bar() +
  coord_flip()
```



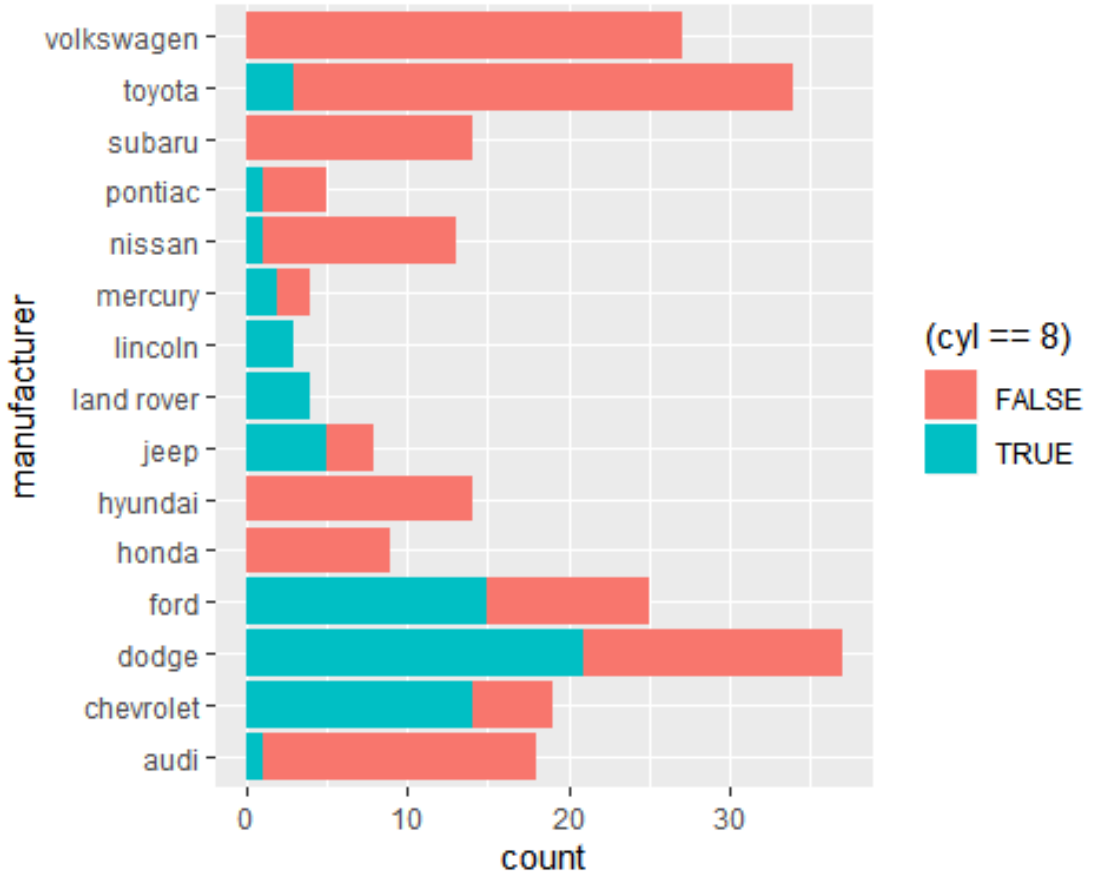
# Too many bars: `fct_lump()`

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



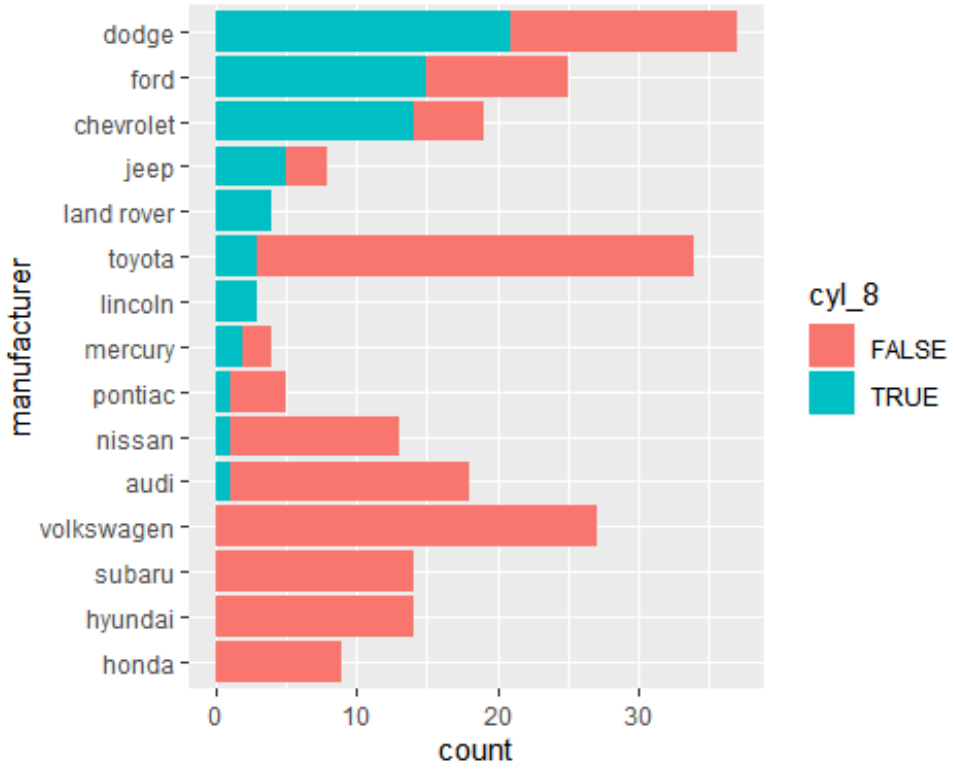
# Order of fill

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



# Order of fill

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

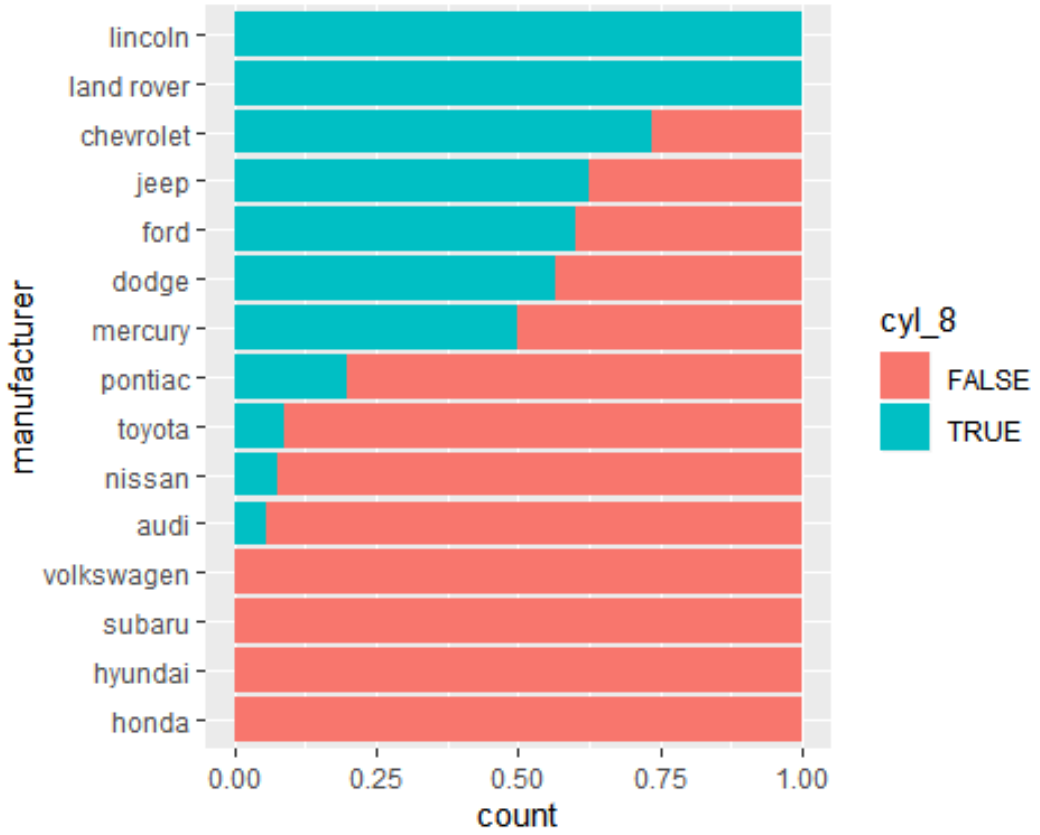


# Q2: can you show the manufacturers with the highest proportion

*(bonus: can you make it 100% fill?)*

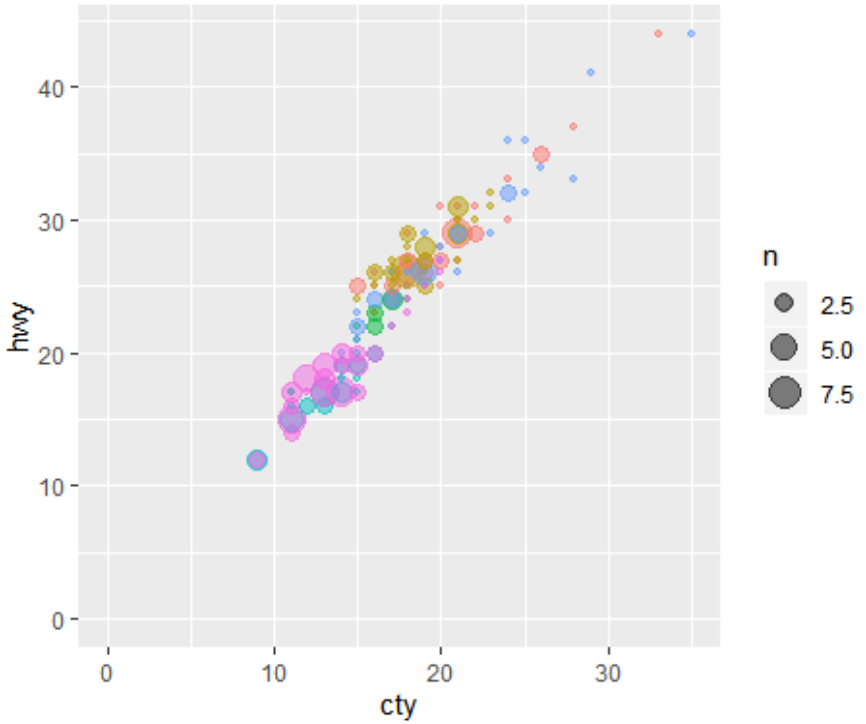
# Q2 Answer

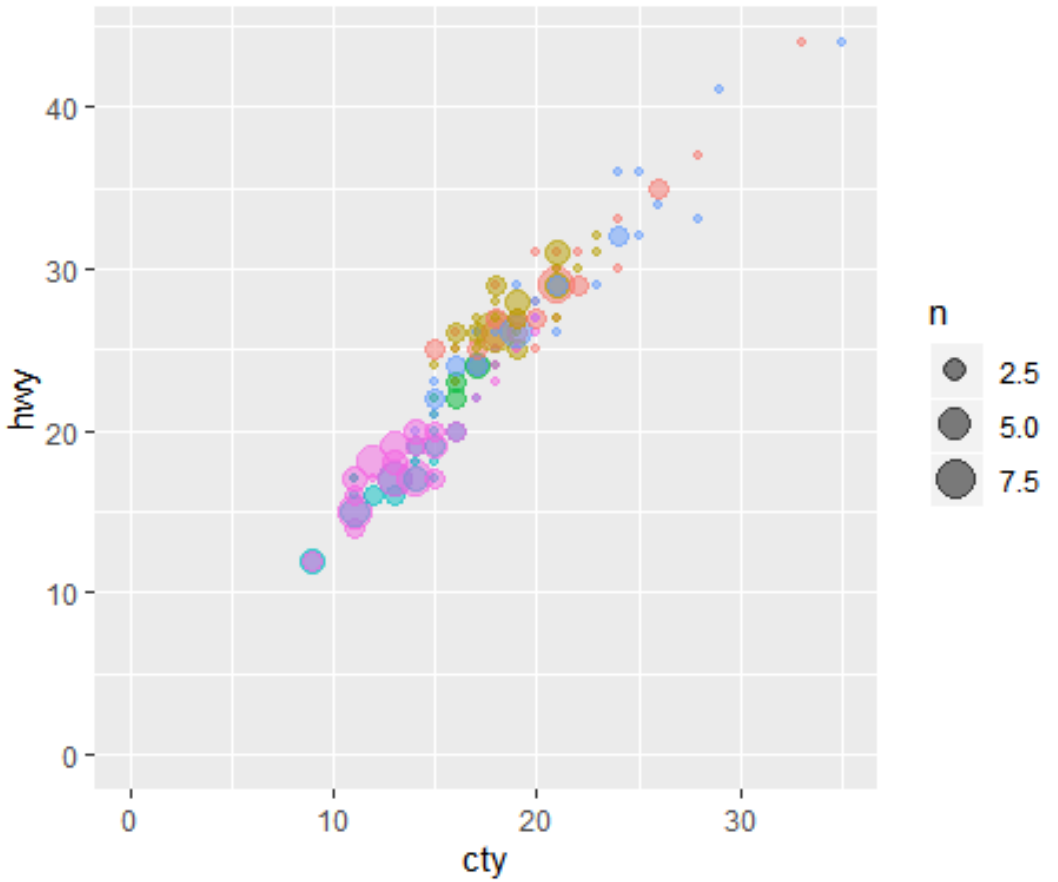
```
mpg %>%
  mutate(
    cyl_8 = (cyl == 8),
    manufacturer = fct_reorder(manufacturer, cyl_8, mean)
  ) %>%
  ggplot(aes(manufacturer, fill = cyl_8)) +
  geom_bar(position = "fill") +
  coord_flip()
```



# facet_grid() vs facet_wrap()

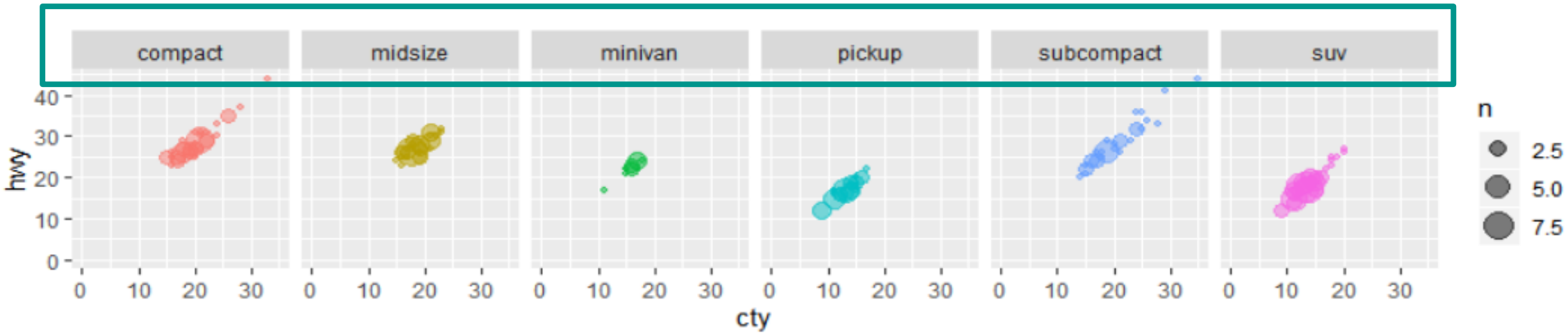
```
p <-
  mpg %>%
  filter(class != "2seater", cyl != 5) %>%
  ggplot(aes(cty, hwy, color = class)) +
  geom_count(alpha = 0.5) +
  lims(x = c(0, NA), y = c(0, NA)) +
  # can also use xlim() or scale_x_continuous
  guides(color = FALSE) +
  theme(aspect.ratio = 1)
```




40
30
20
10
0
hwy
0
10
20
30
cty
n
2.5
5.0
7.5

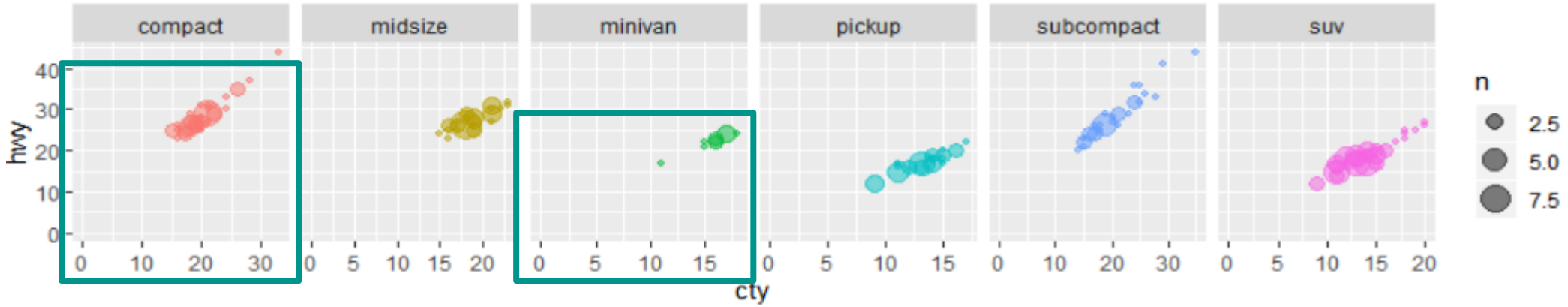
# facet_grid(): new syntax

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



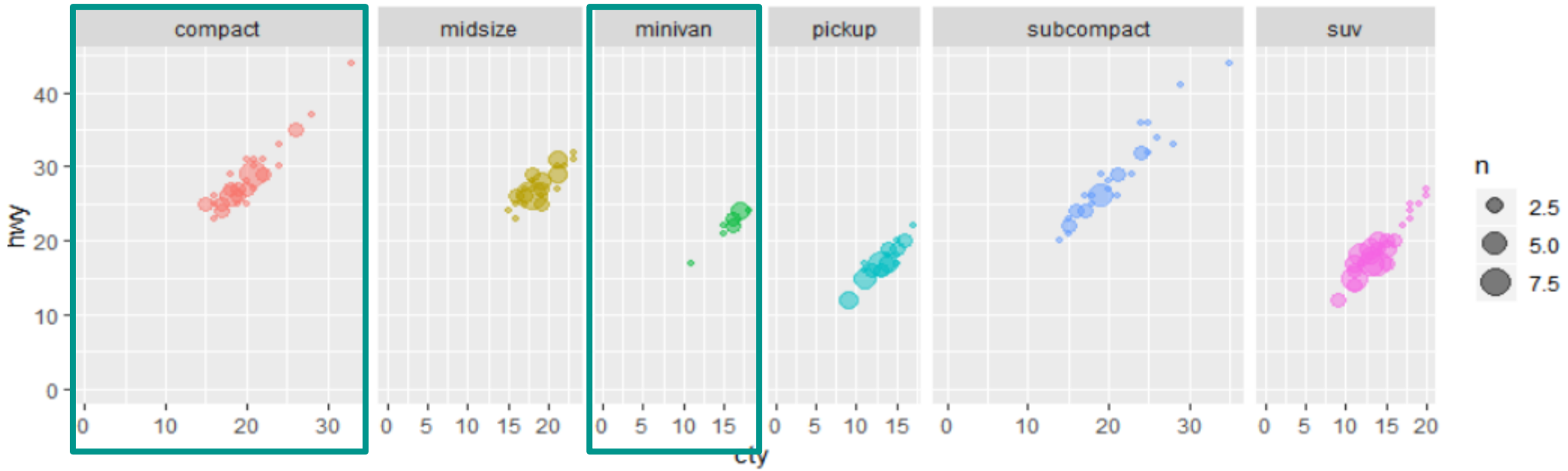
# facets: scales

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



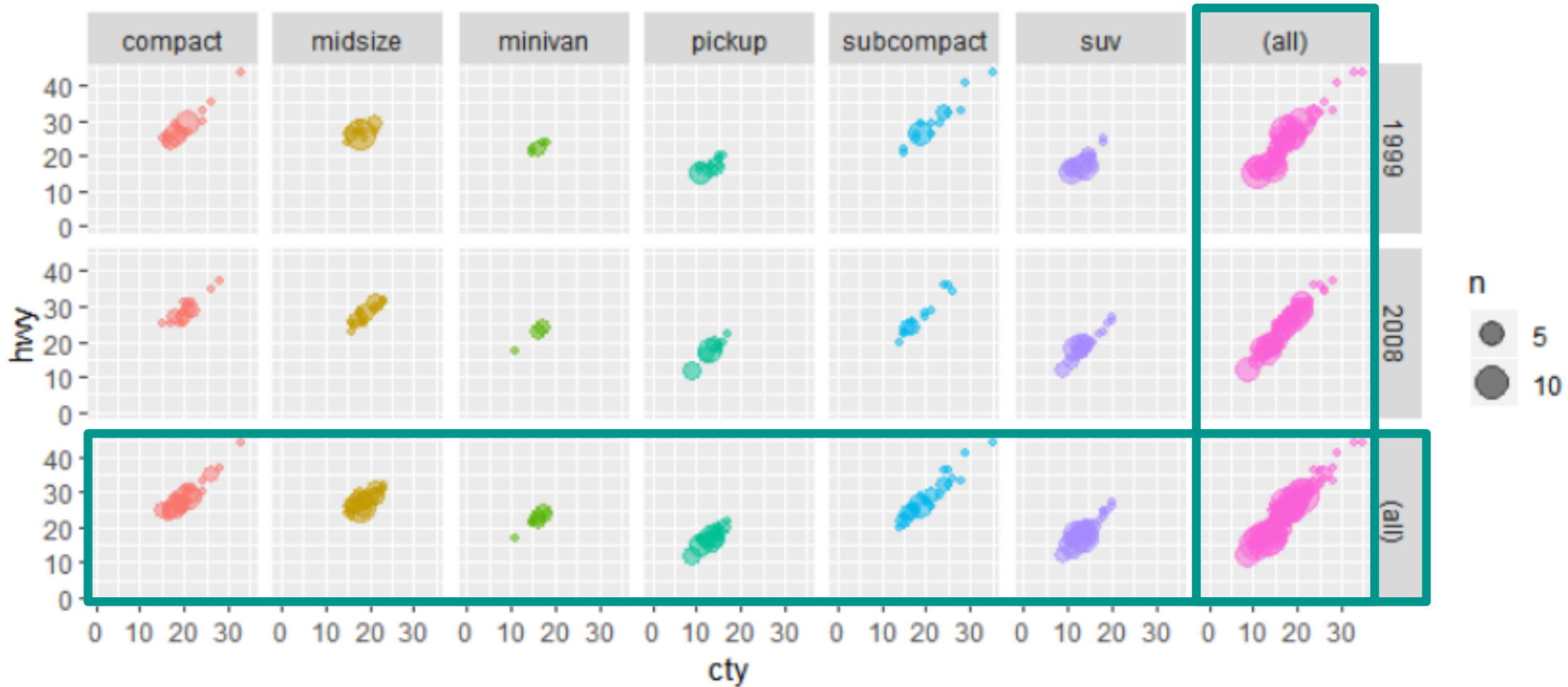
# facets: scales & space

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



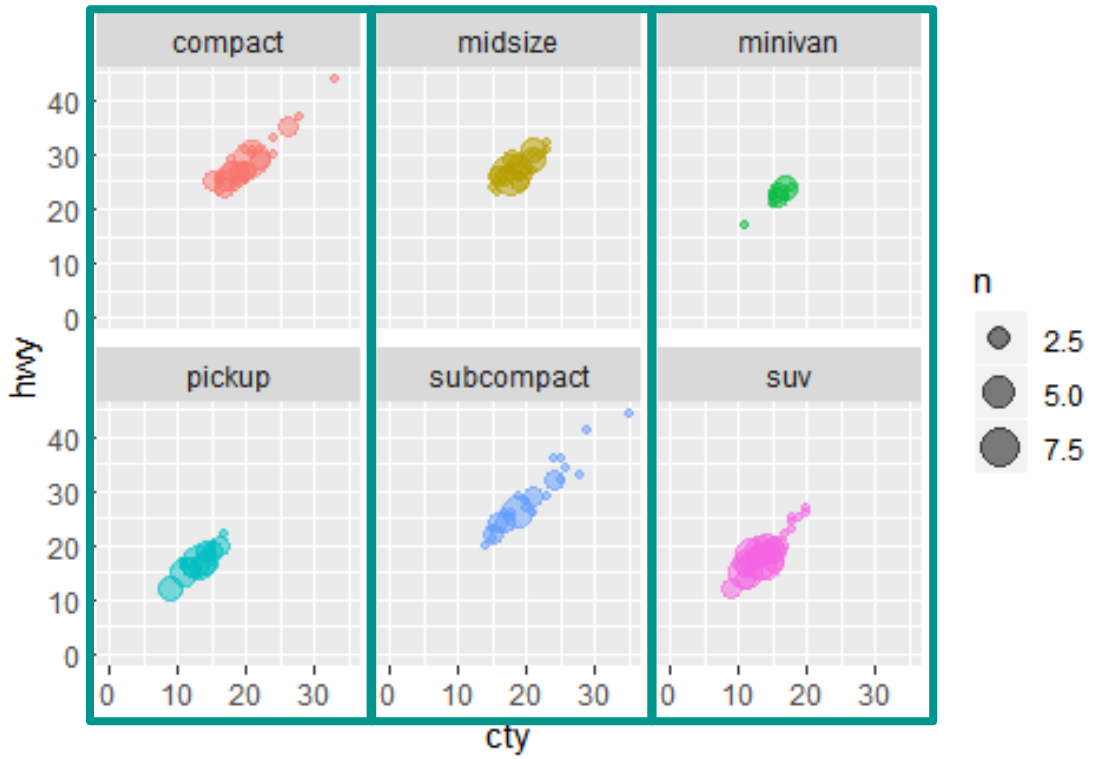
# facets: margins

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



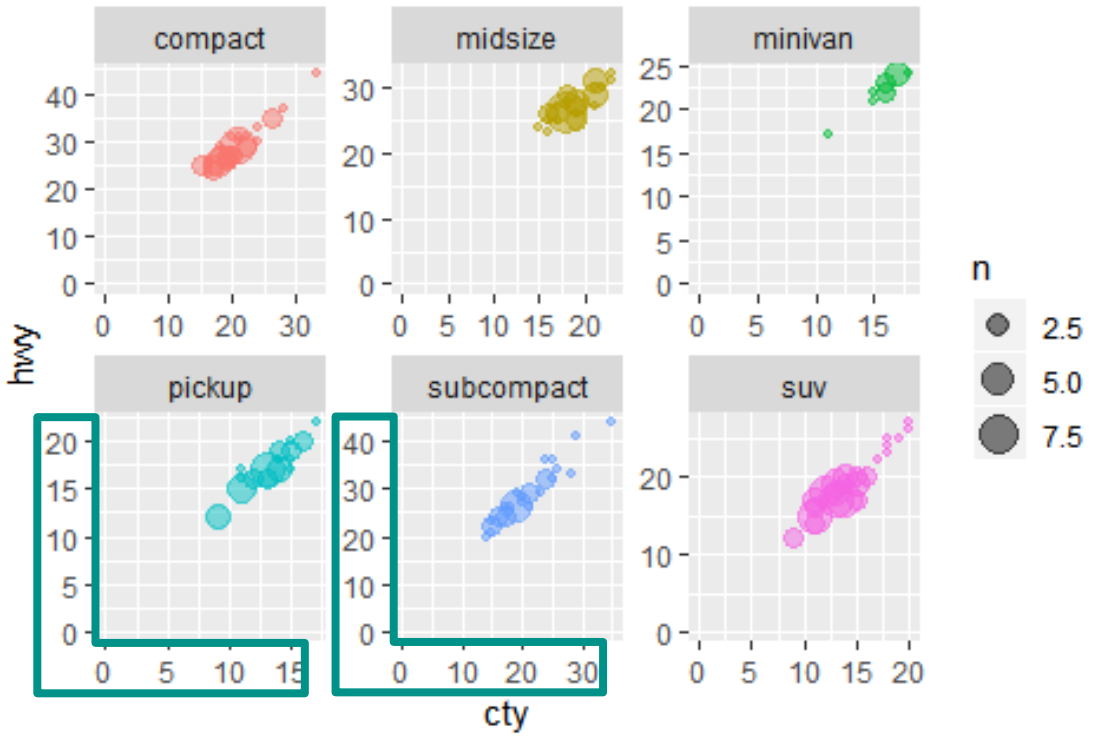
# facet_wrap: # of columns/rows

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



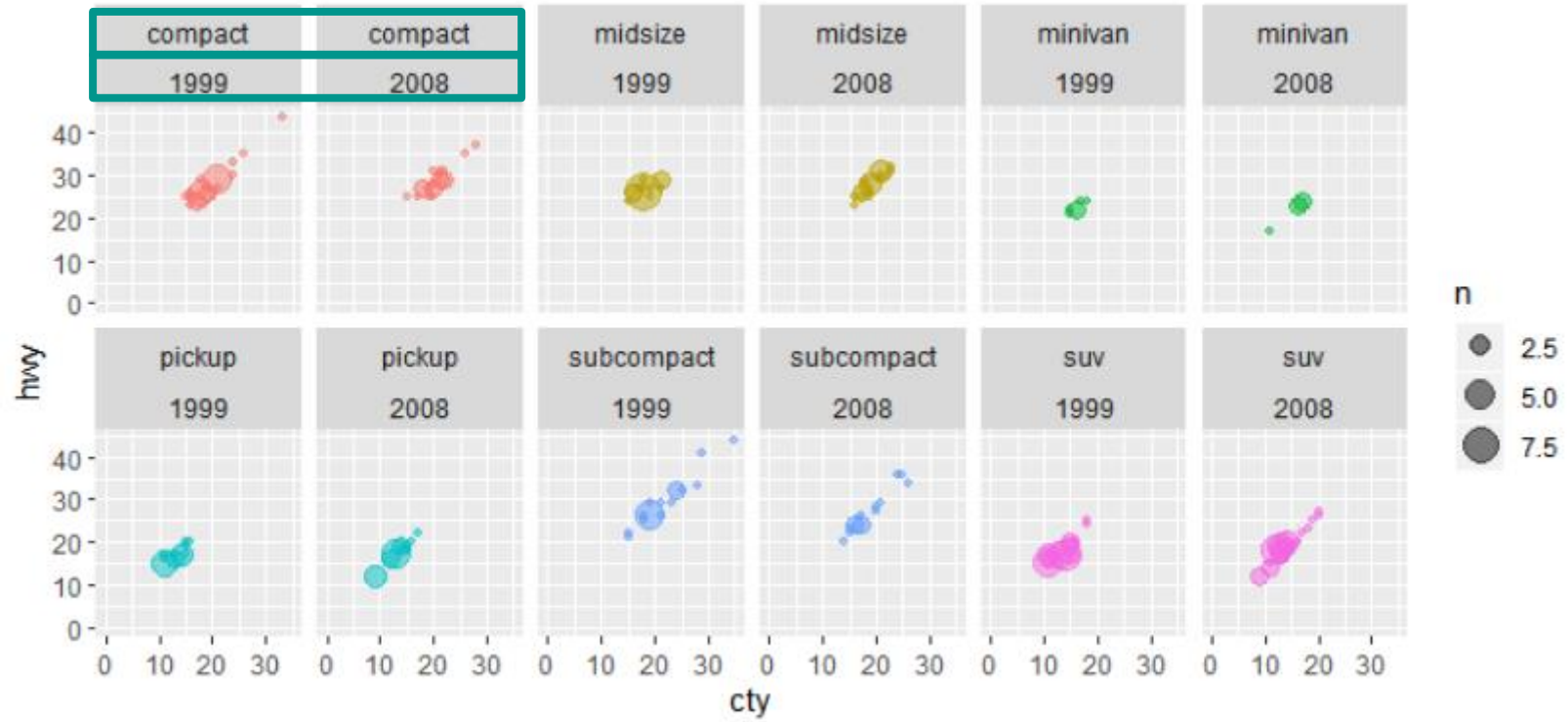
# facet_wrap: scales

```
# 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()

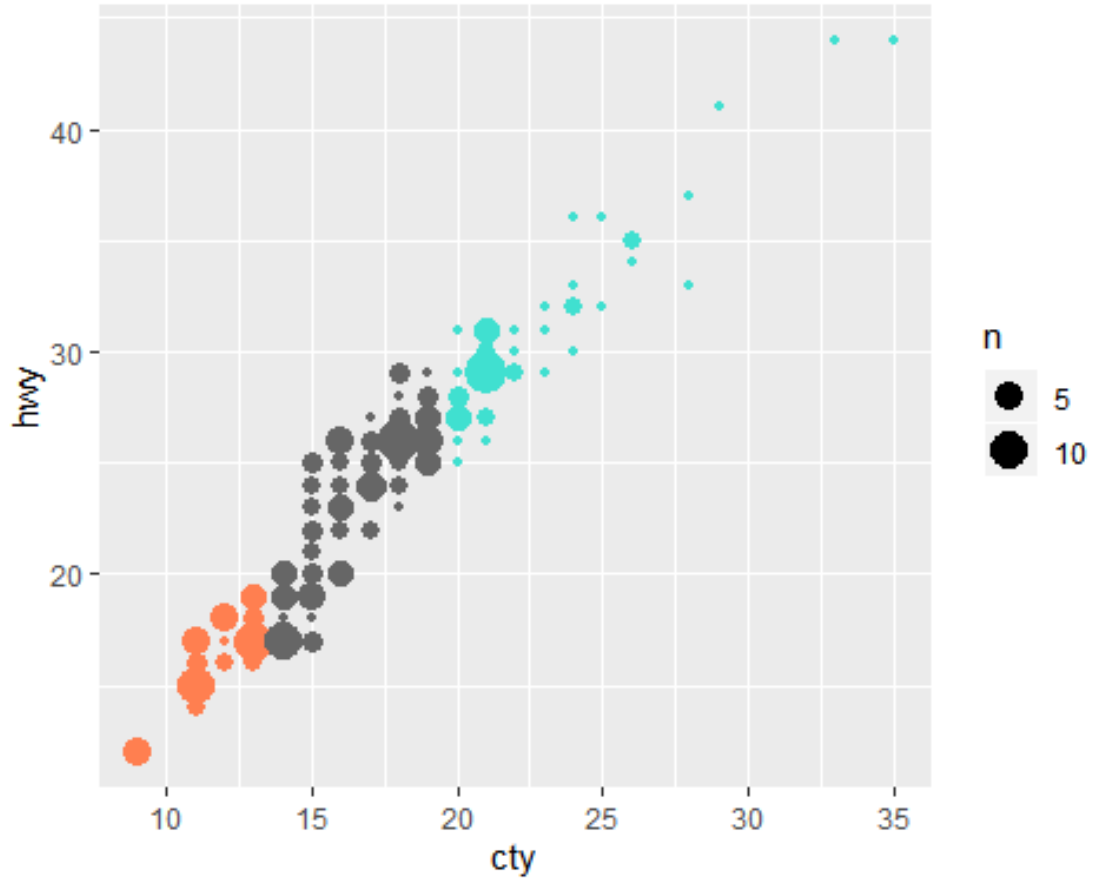
Sometimes I want to have better control over colors & sizes.

Here, I am hard coding the colors

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

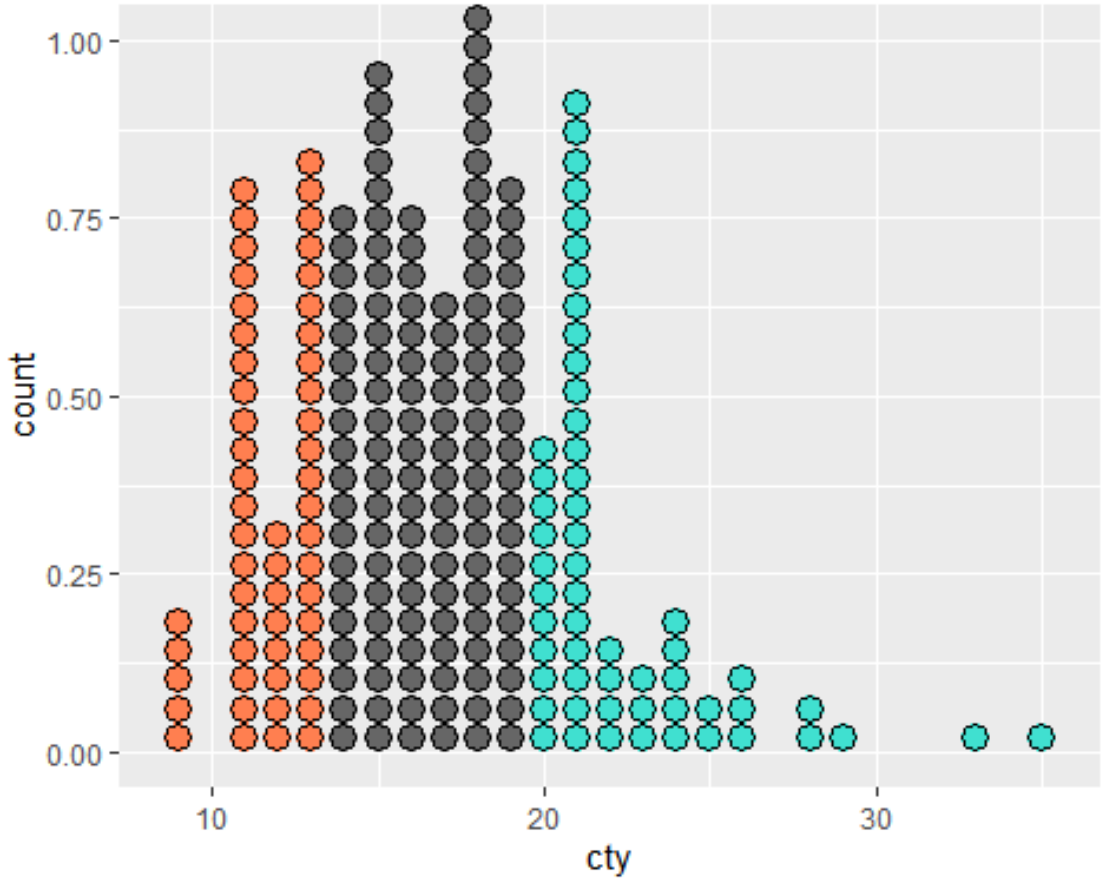
# scale_color_identity()

```
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()
```

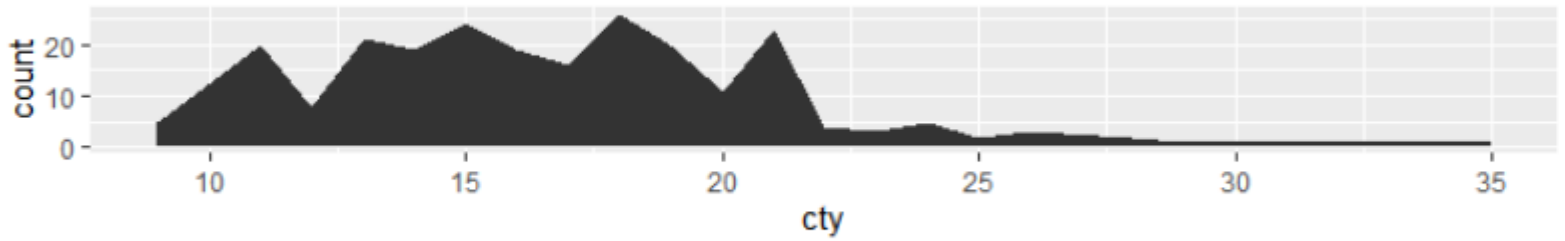


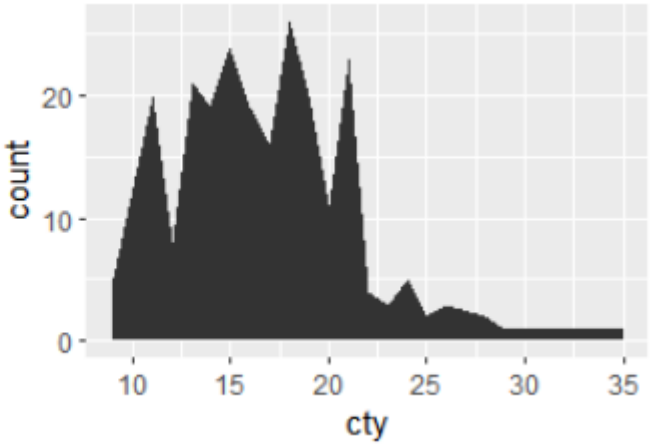
# Best practices

# 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
```



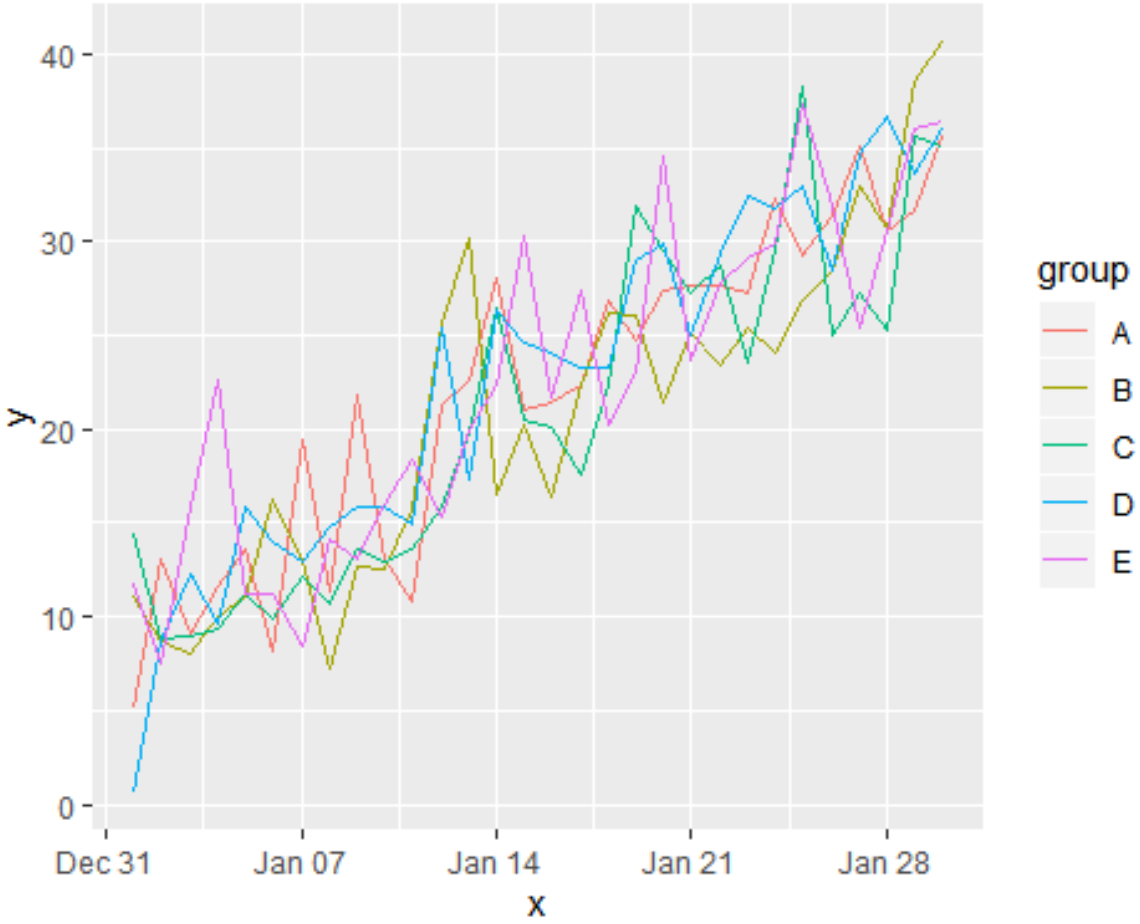
# Dealing with spaghetti charts

This is one of the most common questions:

**change** for **multiple categories** over **time**

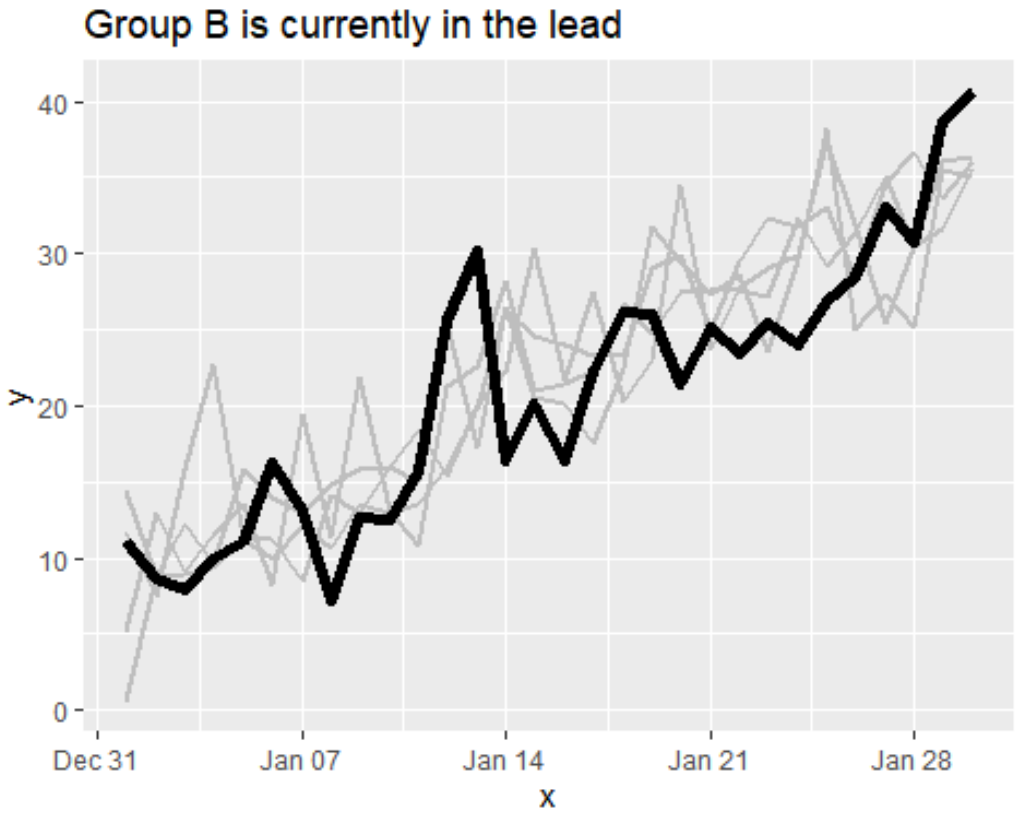
This often results in a chart like the one here. It is hard to read but there are some ways you can **help your audience**

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



# Highlight the focus & use an informative title

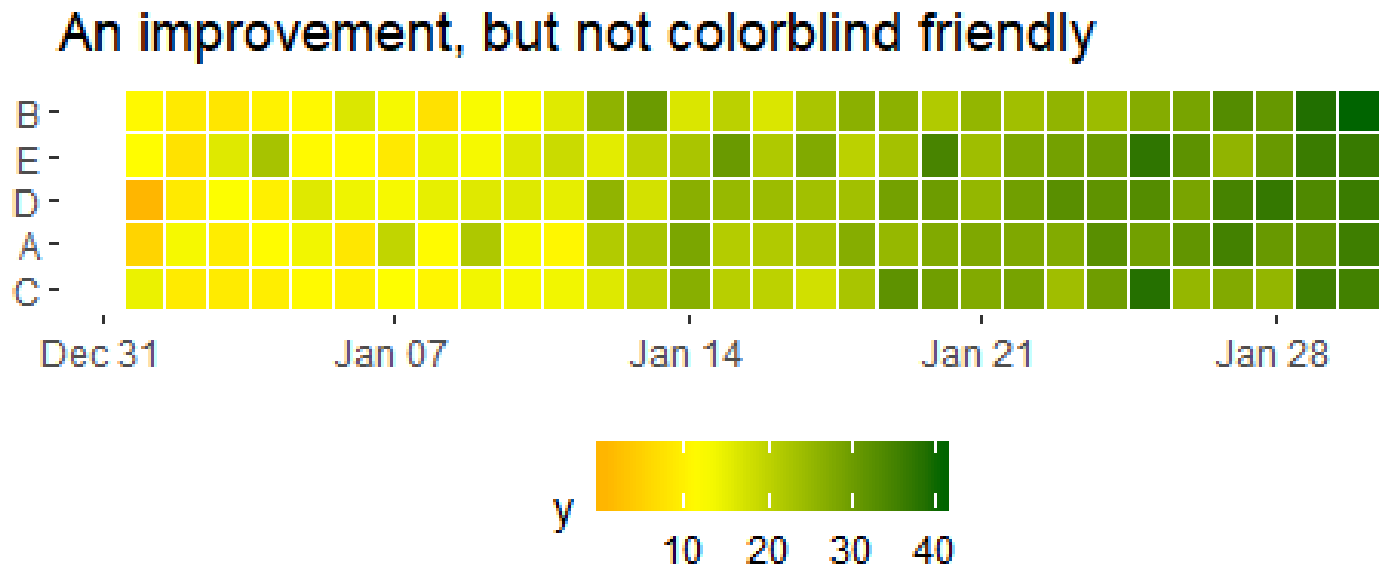
```
ggplot(df, aes(x, y, group = group)) +
  geom_line(data = filter(df, group != "B"), color = "grey", size = 1) +
  geom_line(data = filter(df, group == "B"), color = "black", size = 2) +
  labs(title = "Group B is currently in the lead")
```



# 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 = 12
  ) +
  my_theme +
  labs(title = "An improvement, but not colorblind friendly")
```

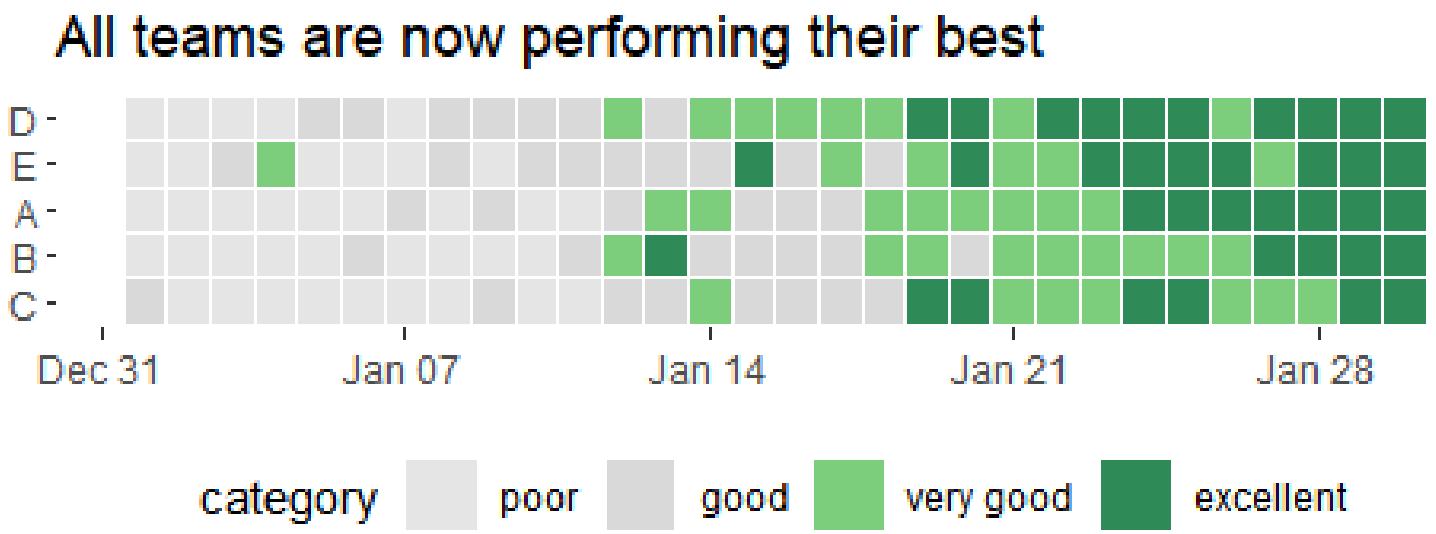
link: colorblind viewer



# 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))



# Partner Activity: Extensions & Addins

# Partner activity

- **Group 1**: focus on ggplot extensions
- **Group 2**: focus on ggplot addins
- *bonus: my* ***`simplecolors`*** *package*

- with your partner, **review the code** and resources below
- **find a function or feature** that you think is interesting or useful
- **place screenshots** here https://bit.ly/2XiG5C7
- you don't need to run the code, **you can use images from the vignettes**
- **we'll share at the end**

# Extensions

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

# addinslist

```
addinslist::addinslistAddin()
```

```
# install.packages("addinslist")
# install.packages("esquisse")
# install.packages("ggedit")
# install.packages("ggThemeAssist")
# install.packages("colourpicker")
```

```
data(iris)
data(mpg)
```

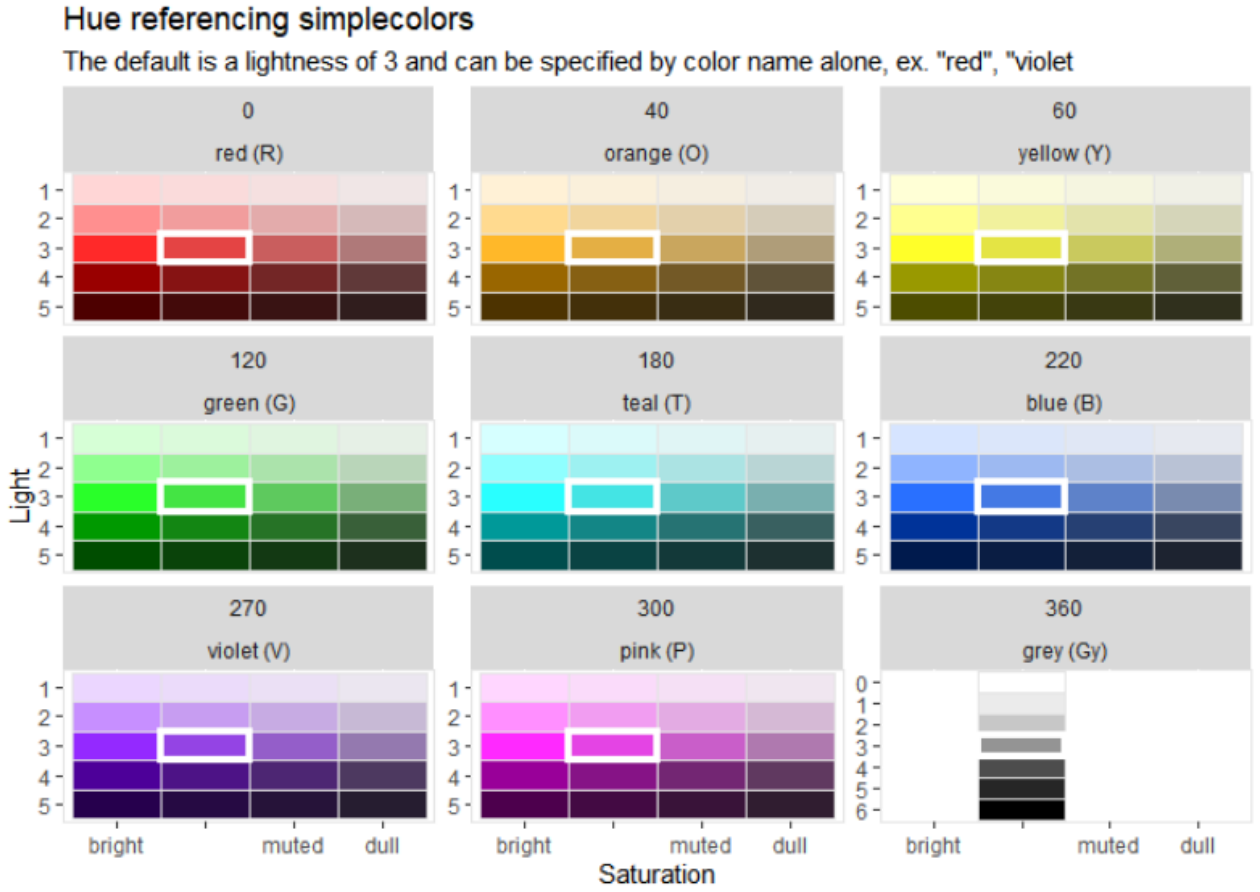
```
p <-
  ggplot(mpg, aes(cty, hwy)) +
  geom_point()
```

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

# others
  ggThemeAssist::ggThemeAssistGadget(p)
  ggedit(p)
  colourpicker::colourPicker()
```

# simplecolors

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

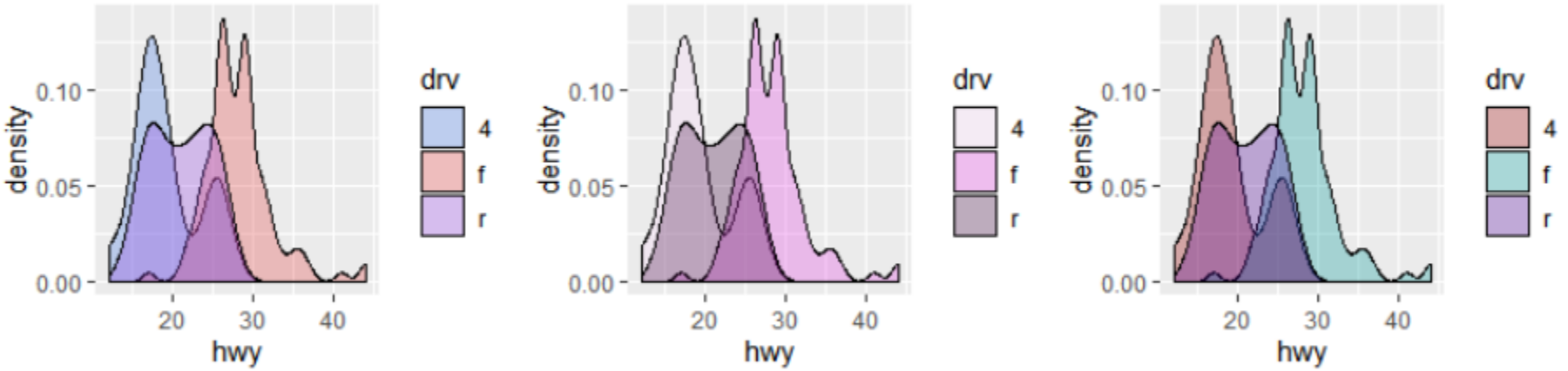
```
# devtools::install_github("rjake/simplecolors")
library(simplecolors)
show_colors(labels = FALSE)
```



# 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 reprex
- include images rather than links
- incorporate **`styler`**

# Cheatsheet

https://github.com/rstudio/cheatsheets/raw/master/data-visualization-2.1.pdf

# 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()` and filters out data **before** doing the stats and your boxplot will keep readjusting the quartiles

# Use `coord_cartesian()` to zoom in

Do not use **`ylim()`** or **`scale_*_continuous()`**

```
# 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)))
)
```

```
##  lower middle upper
##    950   2401  5324
##    911   2161  4679
##    911   2161  4679
##    950   2401  5324
```