Visual Perception!

Maithreyi Gopalan

Week 3

Reminder

Your final project proposals are due next Monday at midnight. Like I mentioned, there is some flexibility in those deadlines. Please look at the syllabus for the requirements and come and talk to me after class today during the lab session if you want to discuss this.

Agenda

- Finish up on some text pattern matching/replacement stuff + R-Markdown Refresher
- Guidelines of Better Visualizations
- Aesthetic mappings and visual encodings of data
- data/ink ratio
- Some do's and don't's (which are all rules of 👍)
- Review Lab 2 and then move onto Lab-PS1 and Lab 3, if time permits

Learning Objectives

- Understand and reflect on some of the basic guidelines for better visualizations
- Understand how decisions you make with your visualizations may help or hinder comprehension
- Refresh the various ways in which data can be accessed in R-Environment
- Refresh basic understanding of R-Markdowns

String manipulations

Reminder: base vs tidyverse

The tidyverse package is {stringr}

It is more consistent than base functions and occassionally faster

However, I tend to prefer the base functions, and they are still more commonly seen "in the wild" than **stringr**.

We'll therefore briefly cover both.

Special characters

- `^`: Anchor - matches the start of a string
- `$`: Anchor - matches the end of a string
- `*`: Matches the preceding character **zero or more** times
- `?`: Matches the preceding character **zero or one** times
- `+`: Matches the preceding character **one or more** times
- `{`: Used to specify number of matches, `a{n}`, `a{n,}`, and `a{n, m}`
- `.`: Wildcard - matches any character
- `|`: OR operator
- `[`: Alternates, also used for character matching (e.g., `[:digit:]`)
- `(`: Used for backreferencing, look aheads, or groups
- `\`: Used to escape special characters

More detail

Both of the below are good places to get more comprehensive information

- RStudio cheatsheet
- Regular expression vignette

A few quick examples

Please follow along

- Because we had some troubles accessing this data directly from the edld652 user-written package, I also downloaded and saved the data in class repo (acgr_lea_2019.rds)

```
library(edld652)
d <- get_data("EDFacts_acgr_lea_2011_2019")
d
```

```
## # A tibble: 11,326 × 29
##   ALL_COHORT ALL_RATE CWD_COHORT CWD_RATE DATE_CUR ECD_COHORT ECD_RATE FIPST FI
##        <dbl> <chr>         <dbl> <chr>    <chr>         <dbl> <chr>    <chr> <c
## 1        252 80                3 PS       03OCT15         121 65-69    01    ht
## 2        398 75               47 70-79    03OCT15         233 65-69    01    ht
## 3       1020 89               51 40-49    03OCT15         175 75-79    01    ht
## 4        750 91               35 60-69    03OCT15         102 80-84    01    ht
## 5        128 55-59            15 LT50     03OCT15          68 40-44    01    ht
## 6        166 90-94             9 GE50     03OCT15          53 70-79    01    ht
## # ℹ 11,320 more rows
```

Challenge

Do three things:

- Create a new variable that identifies if the LEA is associated with a city or a county
- Drop "City" or "County" from the LEA name (e.g., "Albertville City" would be "Albertville")
- Replace all `.` with `--DOT--` in `FILEURL` (so they are not actual links)

You'll need `base::gsub()` or `stringr::str_replace_all()`

05:00

City or county

Ideas?

```
d <- d %>%
  mutate(county = grepl("county$", tolower(LEANM)))
```

Another option

```
d <- d %>%
  mutate(county = grepl("[Cc]ounty$", LEANM))
```

Quick Check

```
d %>%
  select(LEANM, county) %>%
  print(n = 15)
```

```
## # A tibble: 11,326 × 2
##    LEANM            county
##    <chr>            <lgl>
##  1 Albertville City FALSE
##  2 Marshall County  TRUE
##  3 Hoover City      FALSE
##  4 Madison City     FALSE
##  5 Leeds City       FALSE
##  6 Boaz City        FALSE
##  7 Trussville City  FALSE
##  8 Alexander City   FALSE
##  9 Andalusia City   FALSE
## 10 Anniston City    FALSE
## 11 Arab City        FALSE
## 12 Athens City      FALSE
## 13 Attalla City     FALSE
## 14 Auburn City      FALSE
## 15 Autauga County   TRUE
## # ℹ 11,311 more rows
```

Remove city/county

- Lots of ways to do this. Use `base::gsub()` or `stringr::str_replace_all()`
- Replace everything after the space with nothing

```
d %>%
  select(LEANM) %>%
  mutate(new_name = gsub(" .+", "", LEANM))
```

```
## # A tibble: 11,326 × 2
##   LEANM            new_name
##   <chr>            <chr>
## 1 Albertville City Albertville
## 2 Marshall County  Marshall
## 3 Hoover City      Hoover
## 4 Madison City     Madison
## 5 Leeds City       Leeds
## 6 Boaz City        Boaz
## # ℹ 11,320 more rows
```

You can also use `"\\s"` to mean "space"

Another way

There are other ways too, of course

```
d %>%
  select(LEANM) %>%
  mutate(new_name = gsub(" City$| County$", "", LEANM))
```

```
## # A tibble: 11,326 × 2
##   LEANM            new_name
##   <chr>            <chr>
## 1 Albertville City Albertville
## 2 Marshall County  Marshall
## 3 Hoover City      Hoover
## 4 Madison City     Madison
## 5 Leeds City       Leeds
## 6 Boaz City        Boaz
## # ℹ 11,320 more rows
```

Final step

Handling the URLs. This is a bit artificial, but it illustrates escaping, which is important.

- Remember `.` is a special character, so needs to be escaped
- `\` itself is a special character so it needs to be escaped. Functionally, then, you escape special characters with `\\`, not `\`.

```
d %>%
  select(FILEURL)
```

```
## # A tibble: 11,326 × 1
##   FILEURL
##   <chr>
## 1 https://www2.ed.gov/about/inits/ed/edfacts/data-files/acgr-lea-sy2010-11.csv
## 2 https://www2.ed.gov/about/inits/ed/edfacts/data-files/acgr-lea-sy2010-11.csv
## 3 https://www2.ed.gov/about/inits/ed/edfacts/data-files/acgr-lea-sy2010-11.csv
## 4 https://www2.ed.gov/about/inits/ed/edfacts/data-files/acgr-lea-sy2010-11.csv
## 5 https://www2.ed.gov/about/inits/ed/edfacts/data-files/acgr-lea-sy2010-11.csv
## 6 https://www2.ed.gov/about/inits/ed/edfacts/data-files/acgr-lea-sy2010-11.csv
## # ℹ 11,320 more rows
```

```
d %>%
  select(FILEURL) %>%
  mutate(FILEURL = gsub("\\.", "--DOT--", FILEURL)) %>%
  as.data.frame()
```

```
##
## 1    https://www2--DOT--ed--DOT--gov/about/inits/ed/edfacts/data-files/acgr-lea
## 2    https://www2--DOT--ed--DOT--gov/about/inits/ed/edfacts/data-files/acgr-lea
## 3    https://www2--DOT--ed--DOT--gov/about/inits/ed/edfacts/data-files/acgr-lea
## 4    https://www2--DOT--ed--DOT--gov/about/inits/ed/edfacts/data-files/acgr-lea
## 5    https://www2--DOT--ed--DOT--gov/about/inits/ed/edfacts/data-files/acgr-lea
## 6    https://www2--DOT--ed--DOT--gov/about/inits/ed/edfacts/data-files/acgr-lea
## 7    https://www2--DOT--ed--DOT--gov/about/inits/ed/edfacts/data-files/acgr-lea
## 8    https://www2--DOT--ed--DOT--gov/about/inits/ed/edfacts/data-files/acgr-lea
## 9    https://www2--DOT--ed--DOT--gov/about/inits/ed/edfacts/data-files/acgr-lea
## 10   https://www2--DOT--ed--DOT--gov/about/inits/ed/edfacts/data-files/acgr-lea
## 11   https://www2--DOT--ed--DOT--gov/about/inits/ed/edfacts/data-files/acgr-lea
## 12   https://www2--DOT--ed--DOT--gov/about/inits/ed/edfacts/data-files/acgr-lea
## 13   https://www2--DOT--ed--DOT--gov/about/inits/ed/edfacts/data-files/acgr-lea
## 14   https://www2--DOT--ed--DOT--gov/about/inits/ed/edfacts/data-files/acgr-lea
## 15   https://www2--DOT--ed--DOT--gov/about/inits/ed/edfacts/data-files/acgr-lea
## 16   https://www2--DOT--ed--DOT--gov/about/inits/ed/edfacts/data-files/acgr-lea
## 17   https://www2--DOT--ed--DOT--gov/about/inits/ed/edfacts/data-files/acgr-lea
## 18   https://www2--DOT--ed--DOT--gov/about/inits/ed/edfacts/data-files/acgr-lea
## 19   https://www2--DOT--ed--DOT--gov/about/inits/ed/edfacts/data-files/acgr-lea
## 20   https://www2--DOT--ed--DOT--gov/about/inits/ed/edfacts/data-files/acgr-lea
## 21   https://www2--DOT--ed--DOT--gov/about/inits/ed/edfacts/data-files/acgr-lea
## 22   https://www2--DOT--ed--DOT--gov/about/inits/ed/edfacts/data-files/acgr-lea
```

A few more examples

```
some_strings <- c("abc", "a23", "def", "Ad6")
```

Remove the digits

```
gsub("\\d", "", some_strings)
```

```
## [1] "abc" "a"   "def" "Ad"
```

Extract the digits

```
gsub("\\D", "", some_strings)
```

```
## [1] ""   "23" ""   "6"
```

Starts with A

```
grepl("^[aA]", some_strings)
```

```
## [1]  TRUE  TRUE FALSE  TRUE
```

There's lots more

- Please checkout the vignette, in particular
- I end up using back-referencing a lot

```
cities <- c("Eugene, Oregon", "Denver, Colorado")
gsub(
  "^(.+), (.+)$", # pattern with two groupings
  "The state of \\2 includes the city of \\1", # reference the gr
  cities)
```

```
## [1] "The state of Oregon includes the city of Eugene"   "The state of Colorado
```

R-Markdown Quick refresher and check-in

R-Markdown Quick refresher and check-in

- Level setting - let's watch this brief video
- Let's quickly browse this cheatsheet

Five Guidelines of Data Visualization

Guidelines from Jonathan Schwabish

- Show the data
- Reduce the clutter
- Integrate the graphics and text
- Avoid the spaghetti chart :)
- Start with grey

Data viz in the wild

- Aden

Elizabeth and Erick on deck

Visual perception

Disclaimer

I'm not a (cognitive) psychologist

I don't really know why we perceive things certain ways.

I mainly care that we do, and that your visualizations should account for them.

Visual Cues

- **Position:** *Numeric*. Where in relation to other things?
- **Length:** *Numeric*. How big (in one dimension)?
- **Angle:** *Numeric*. How wide? Parallel to something else?
- **Direction:** *Numeric*. At what slope? In a time series, going up or down?

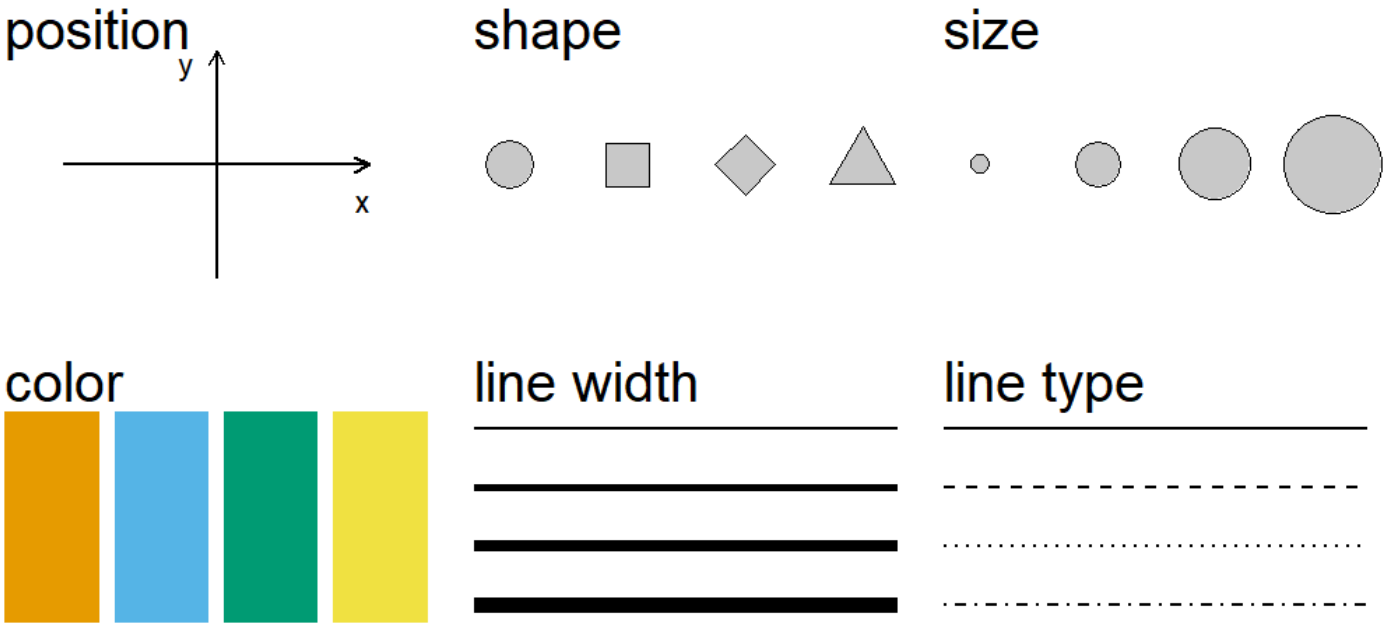
Taken from *Modern Data Science with R*, p. 15

Visual Cues

- **Shape:** *Categorical*. Belonging to which group?
- **Area:** *Numeric*. How big (in two dimensions)?
- **Volume:** *Numeric*. How big (in three dimensions)?
- **Shade:** *Numeric or Categorical*. To what extent? How Severely?
- **Color:** *Numeric or Categorical*. To what extent? How Severely?

Taken from *Modern Data Science with R*, p. 15

Encoding data



Other elements to consider

- Text
  - How is the text displayed (e.g., font, face, location)?
  - What is the purpose of the text?
- Transparency
  - Are there overlapping pieces?
  - Can transparency help?
- Type of data
  - Continuous/categorical
  - Which can be mapped to each aesthetic?
  - e.g., shape and line type can only be mapped to categorical data, whereas color and size can be mapped to either.

Talk with a neighbor

How would you encode each column of data?

Month	Day	Location	Station ID	Temperature
Jan	1	Chicago	USW00014819	25.6
Jan	1	San Diego	USW00093107	55.2
Jan	1	Houston	USW00012918	53.9
Jan	1	Death Valley	USC00042319	51.0
Jan	2	Chicago	USW00014819	25.5
Jan	2	San Diego	USW00093107	55.3
Jan	2	Houston	USW00012918	53.8
Jan	2	Death Valley	USC00042319	51.2
Jan	3	Chicago	USW00014819	25.3

You can assume that month and day can be collapsed to a single *date* column

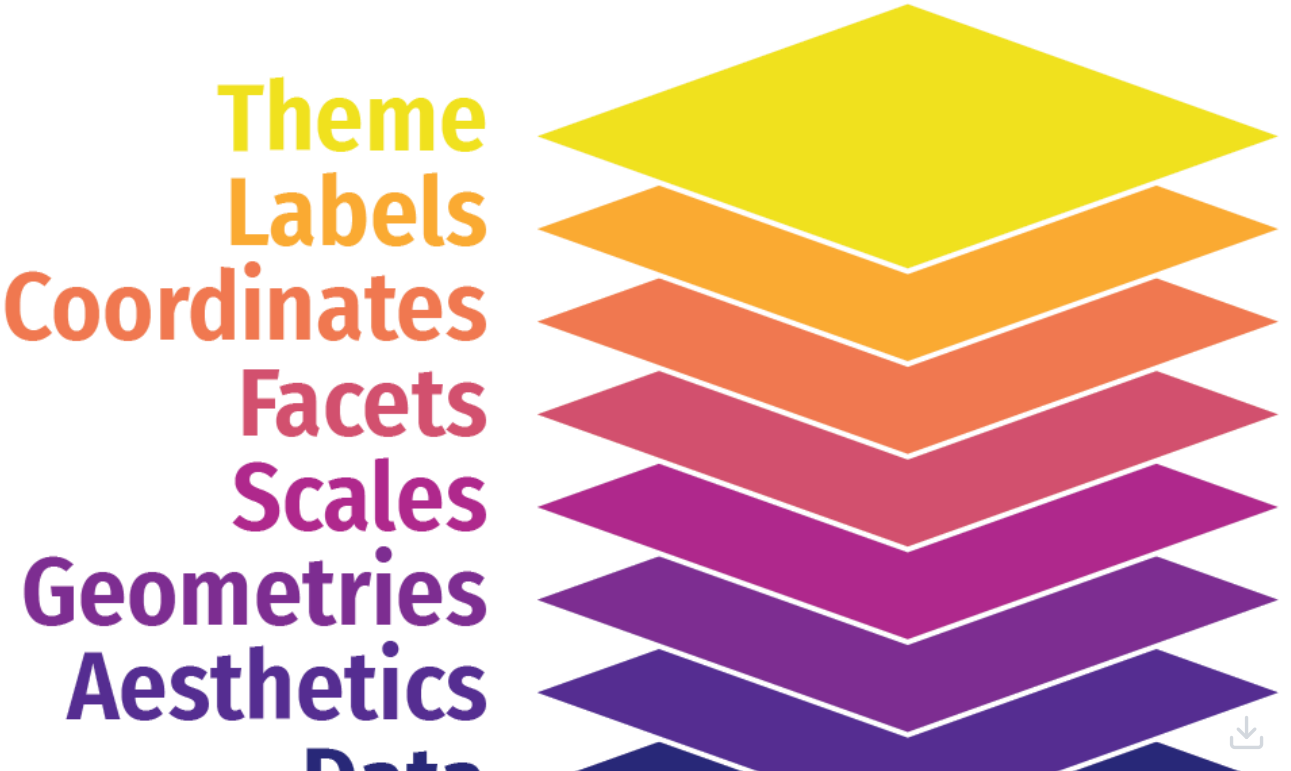
Scales

> A scale defines a unique mapping between data and aesthetics. Importantly, a scale must be one-to-one, such that for each specific data value there is exactly one aesthetics value and vice versa. If a scale isn't one-to-one, then the data visualization becomes ambiguous.

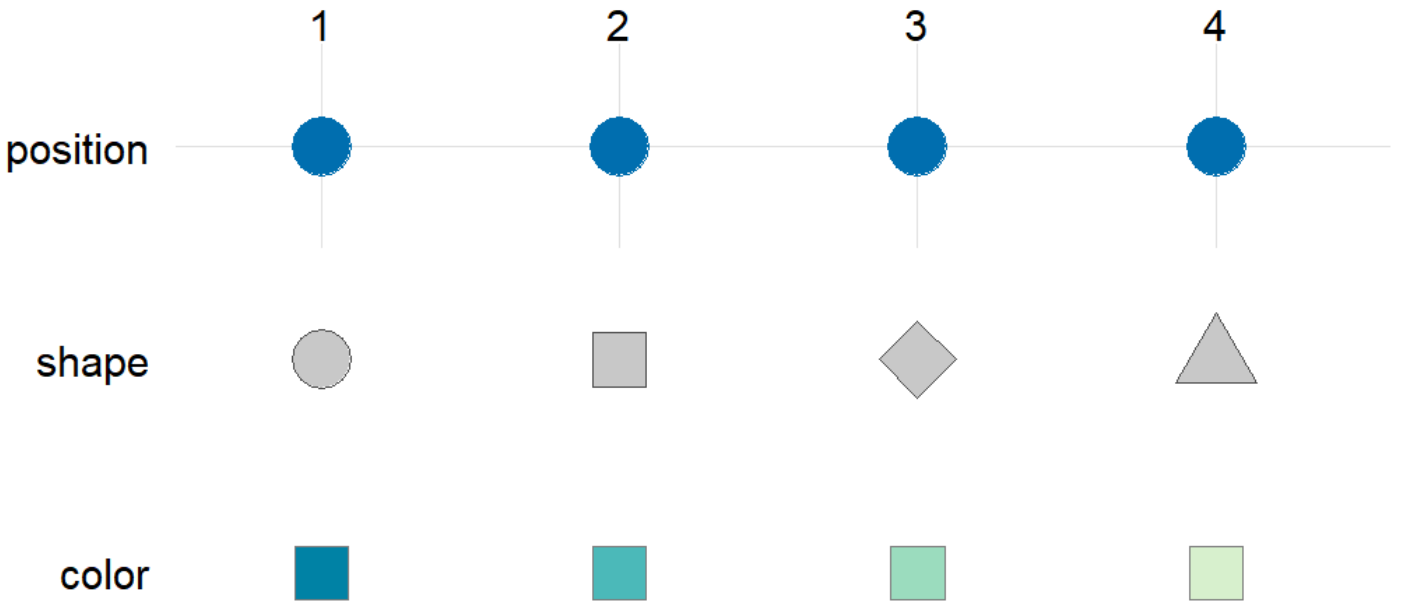
- Which data values correspond to specific aesthetic values?

Putting it to practice

- Changing colors, shapes, and sizes with `scale_*()`
- Grammar of graphic uses a set of layers to define elements of plots



Basic Scales



Putting it to practice

All functions that deal with scales conveniently follow the same namming pattern:

```
scale_AESTHETIC_DETAILS()
```

Putting it to practice

- Common Scale Functions

```
scale_x_continuous() scale_x_date()
scale_y_reverse() scale_color_viridis_c()
scale_shape_manual(values = c(19, 13, 15))
scale_fill_manual(values = c("red", "orange",
"blue"))
```

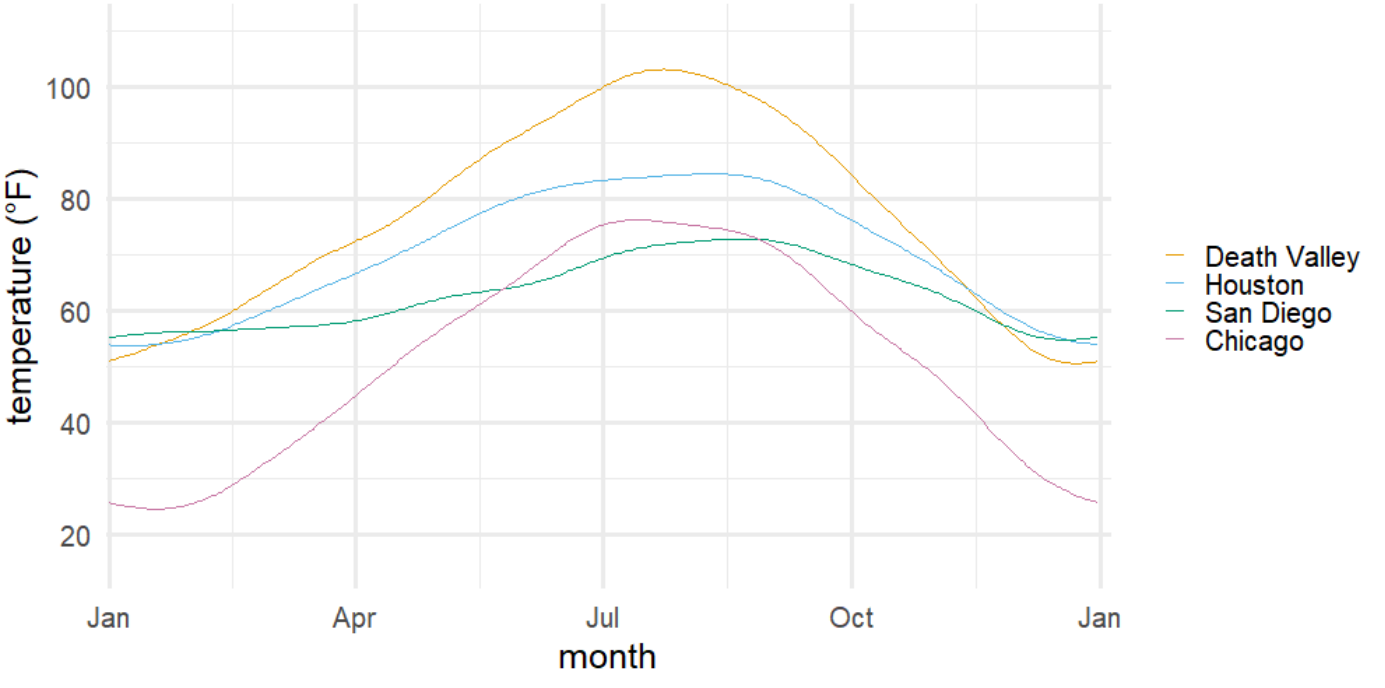
- You can see a list of all of the possible scale functions here, and you should reference that documentation (and the excellent examples) often when working with these functions.

- You can check the documentation for scales for details about all the labeling functions it has, including dates, percentages, p-values

Putting it to practice

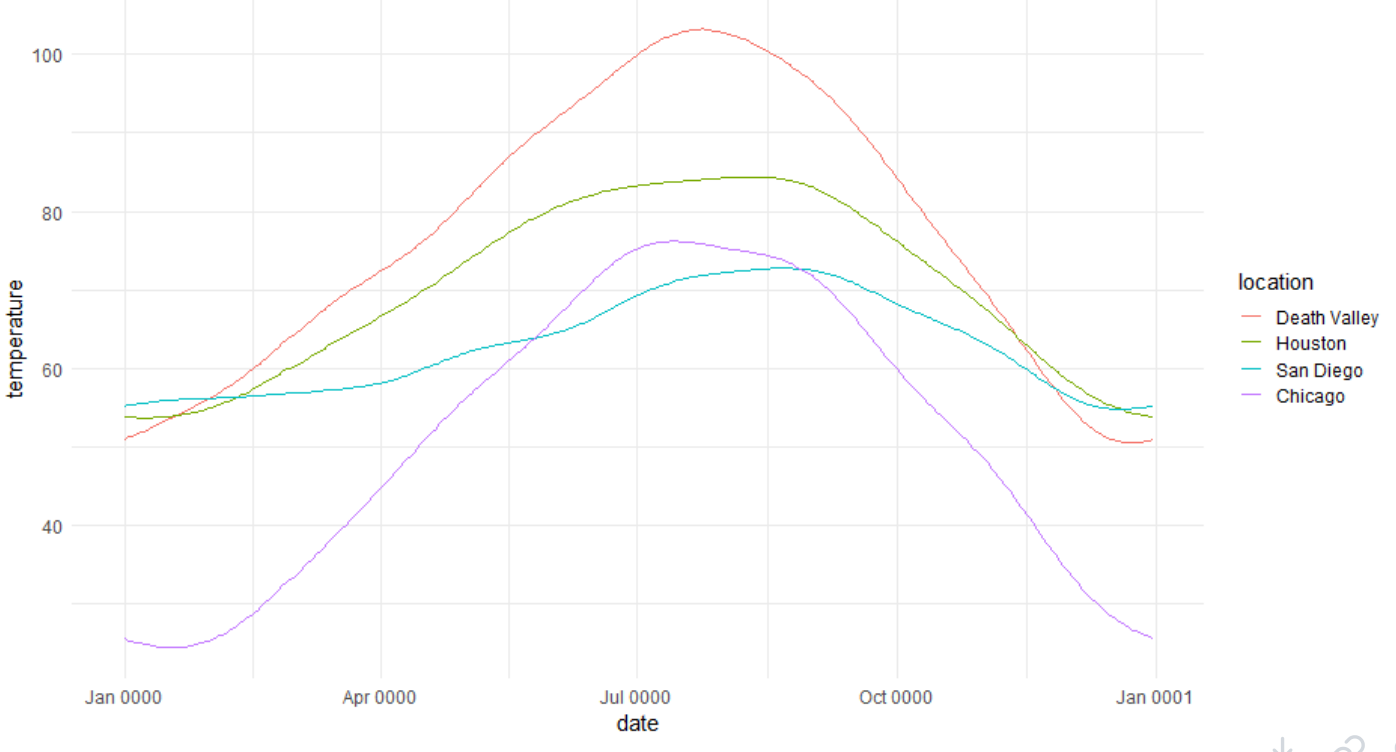
As long as you have mapped a variable to an aesthetic with `aes()`, you can use the `scale_*()` functions to deal with it.

Putting it to practice



Basic code for previous plot

```
ggplot(temps_long, aes(date, temperature)) +
  geom_line(aes(color = location))
```

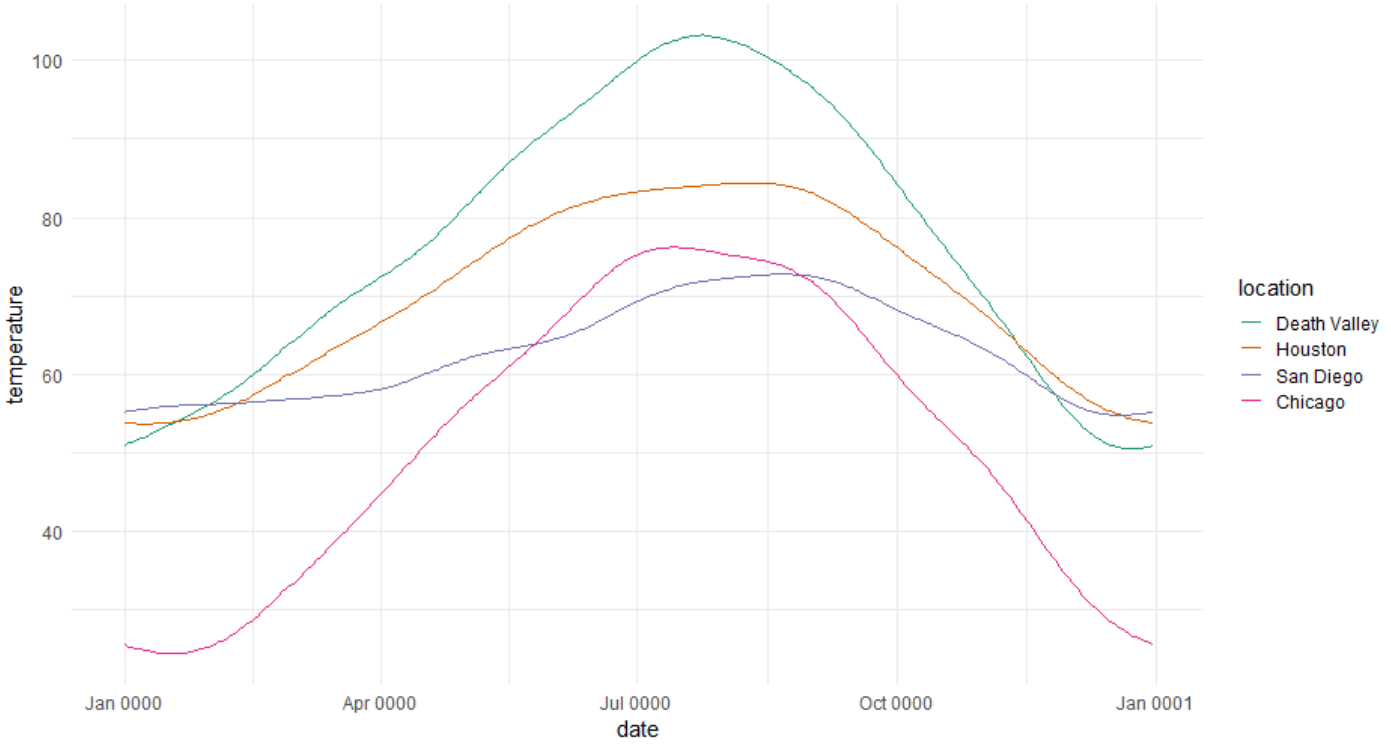


Change colors

If you want to change the colors on the previous plot, you have to change the colors of the scale for the color mapping.

In other words, color is being mapped to data, and you have to change the color scale.

```
ggplot(temps_long, aes(date, temperature)) +
  geom_line(aes(color = location)) +
  scale_color_brewer(palette = "Dark2")
```


100
80
60
40
temperature
Jan 0000
Apr 0000
Jul 0000
Oct 0000
Jan 0001
date
location
Death Valley
Houston
San Diego
Chicago

One more note on colors

There are lots of different scales and some work better than others. We'll talk about them more next week.

You **do not** use `scale_color_*()` if you are not mapping data to color

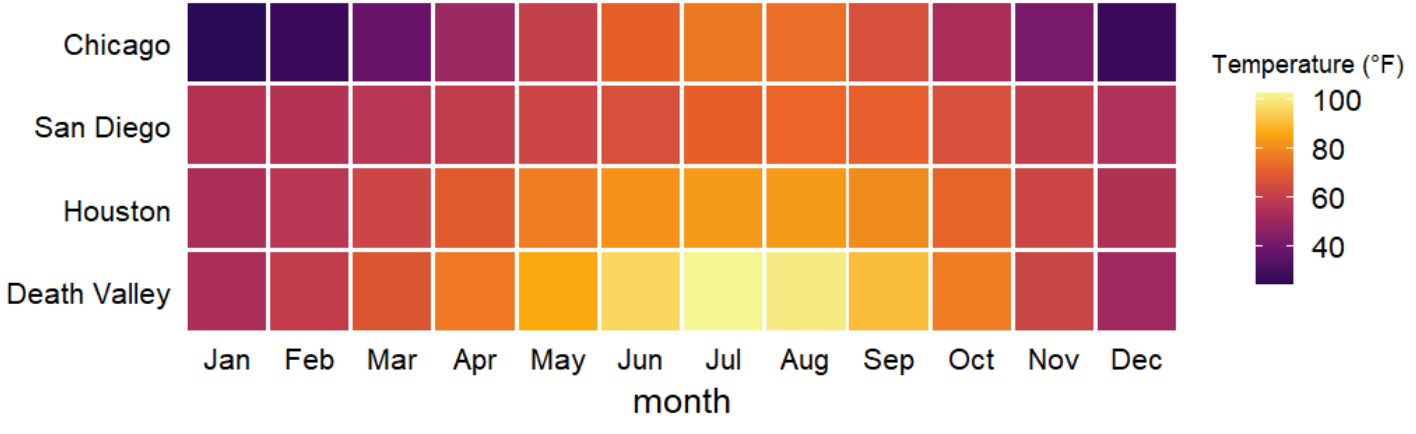
Make sure to keep straight `scale_color_*()` and `scale_fill_*()`

Alternative represenation

Can you think of other ways to show this relation?

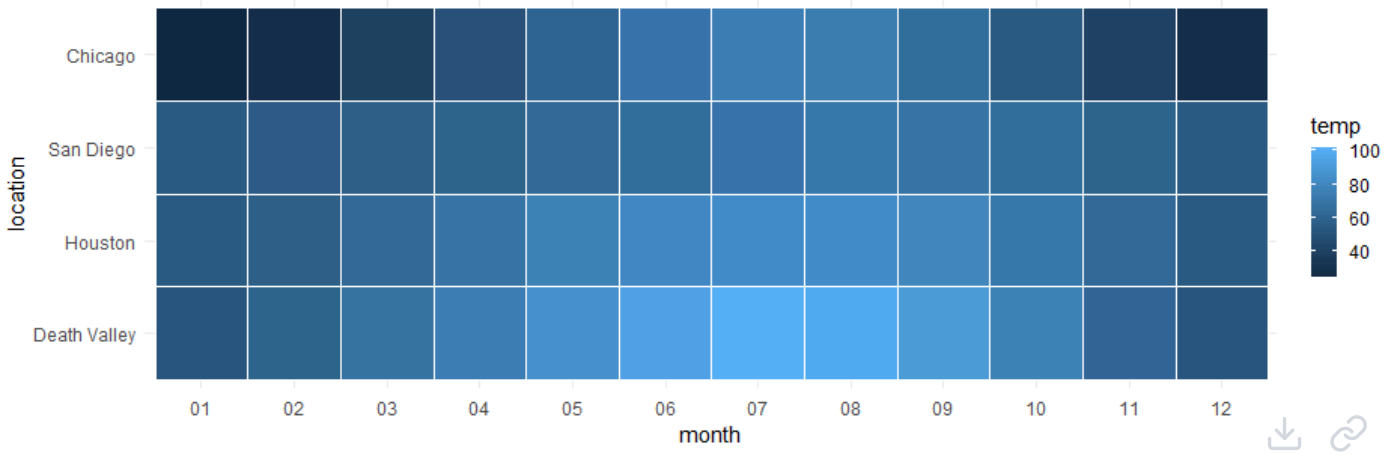
Alternative representation

Same plot as before, but with different scales



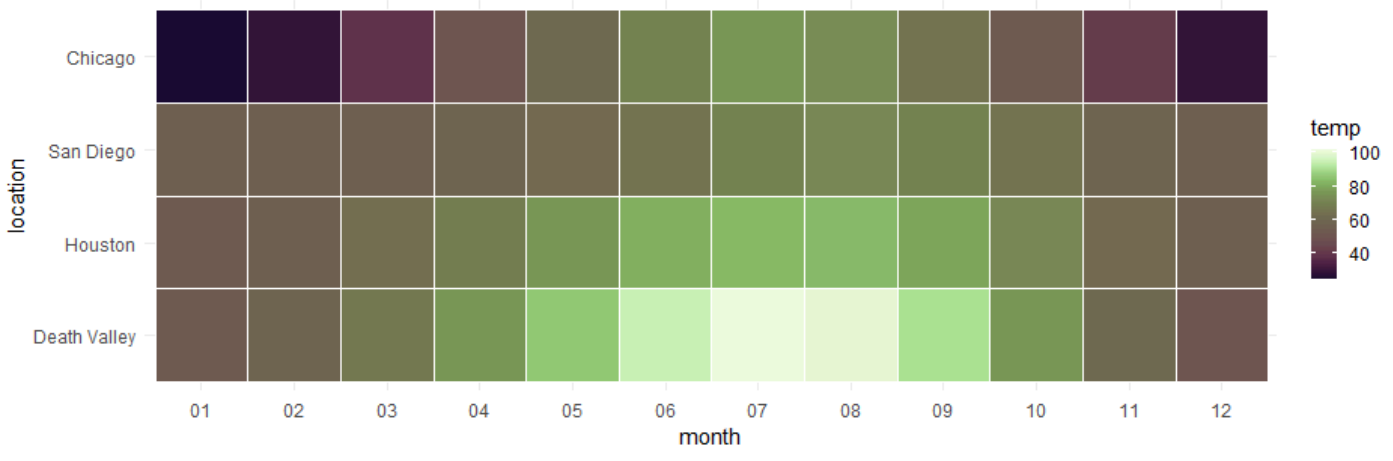
Basic code for previous plot

```
temps_long %>%
  group_by(location, month) %>%
  summarize(temp = mean(temperature)) %>%
  ggplot(aes(month, location)) +
  geom_tile(aes(fill = temp),
            color = "white") +
  coord_fixed()
```



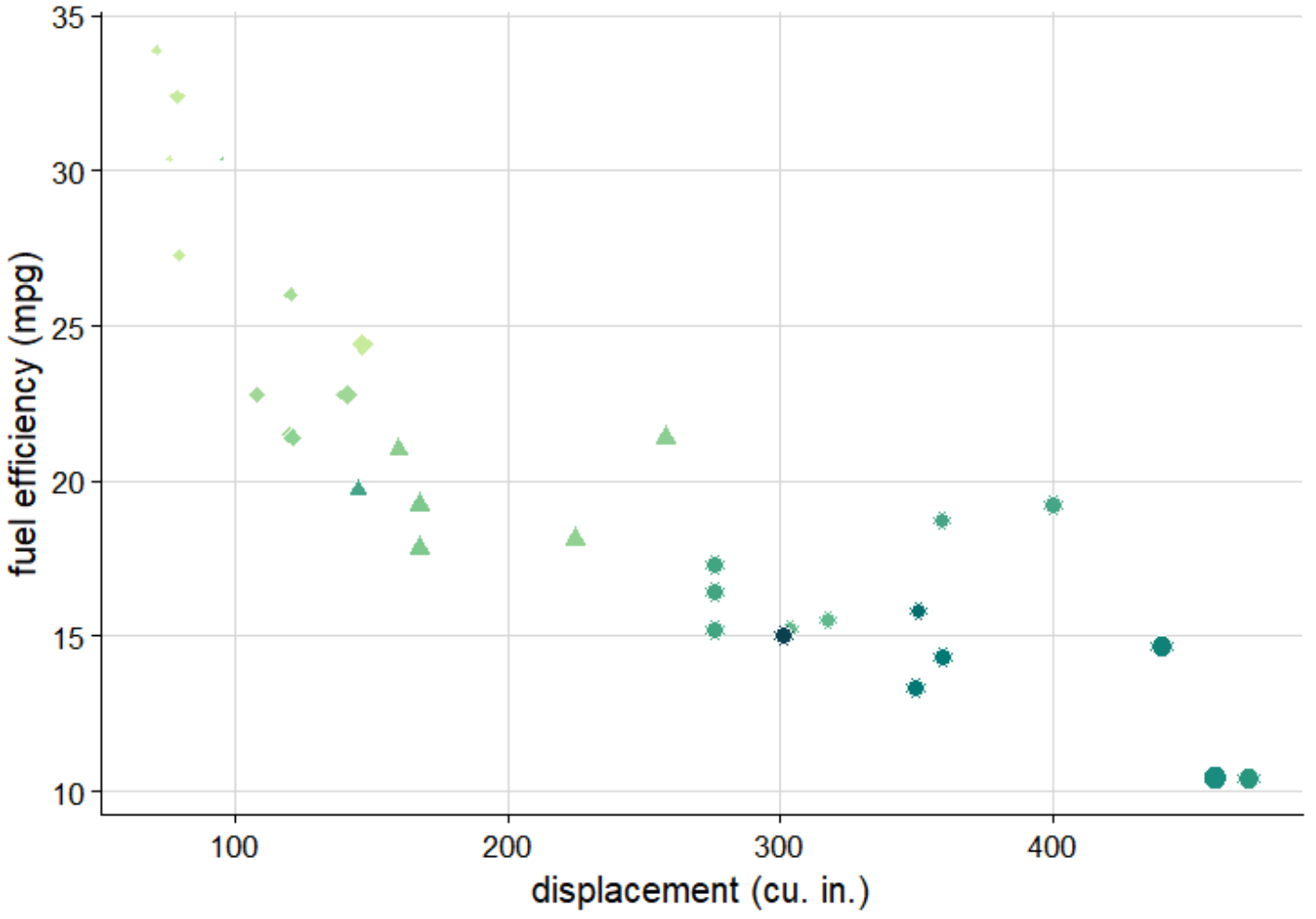
Change the fill

```
temps_long %>%
  group_by(location, month) %>%
  summarize(temp = mean(temperature)) %>%
  ggplot(aes(month, location)) +
  geom_tile(aes(fill = temp),
            color = "white") +
  coord_fixed() +
  scico::scale_fill_scico(palette = "tokyo")
```

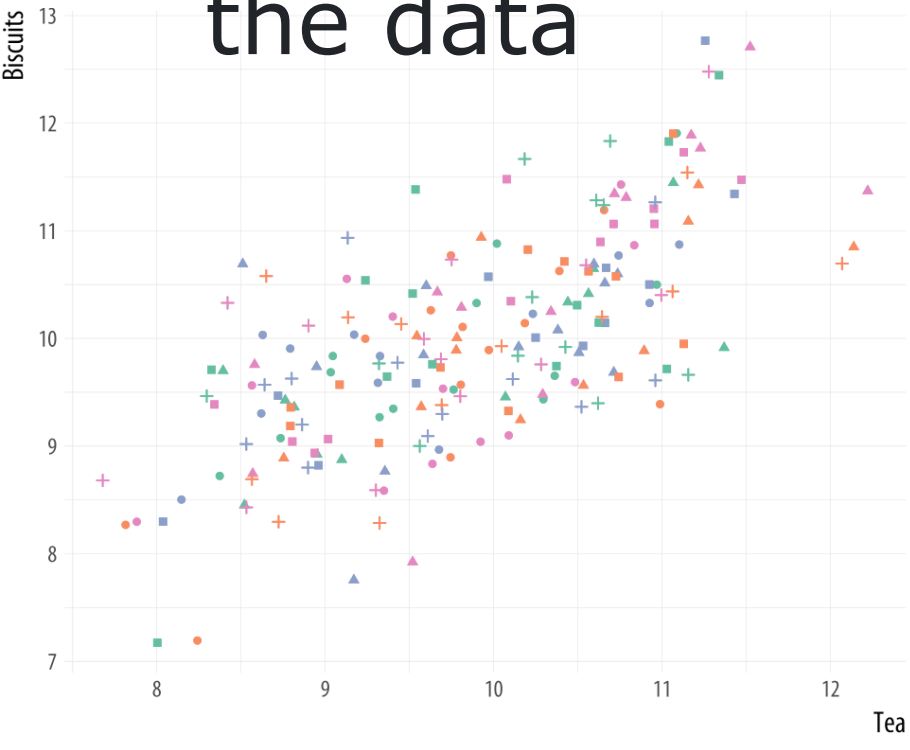


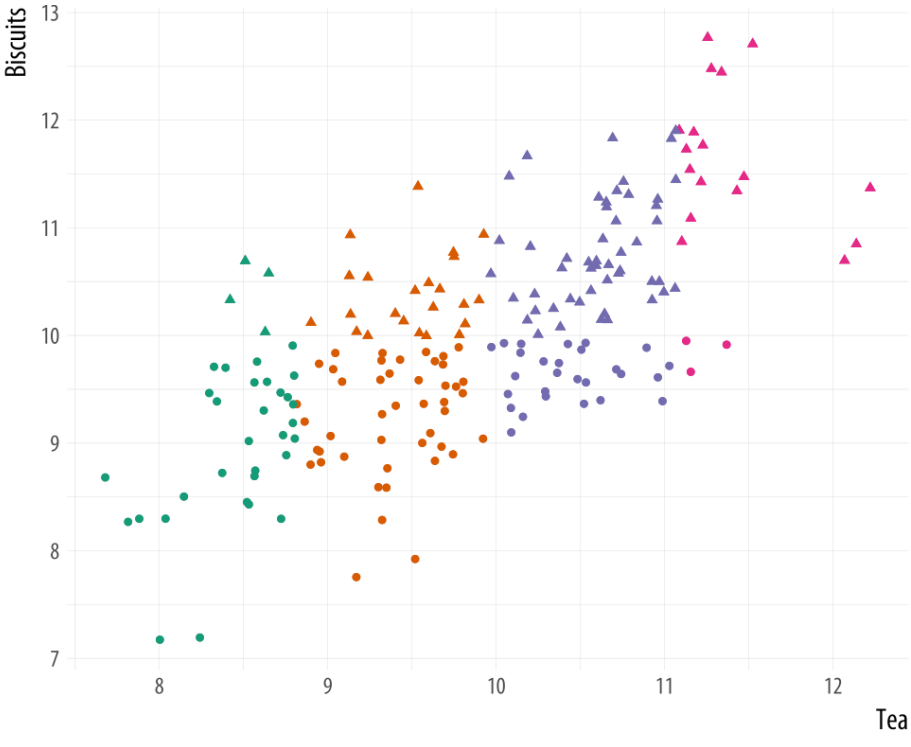
Comparison

- Both represent three scales
  - Two position scales (x/y axis)
  - One color scale (categorical for the first, continuous for the second)
- More scales are possible


35
30
25
20
15
10
fuel efficiency (mpg)
100
200
300
400
displacement (cu. in.)

Additional scales can become lost without high structure in the data





Data ink ratio

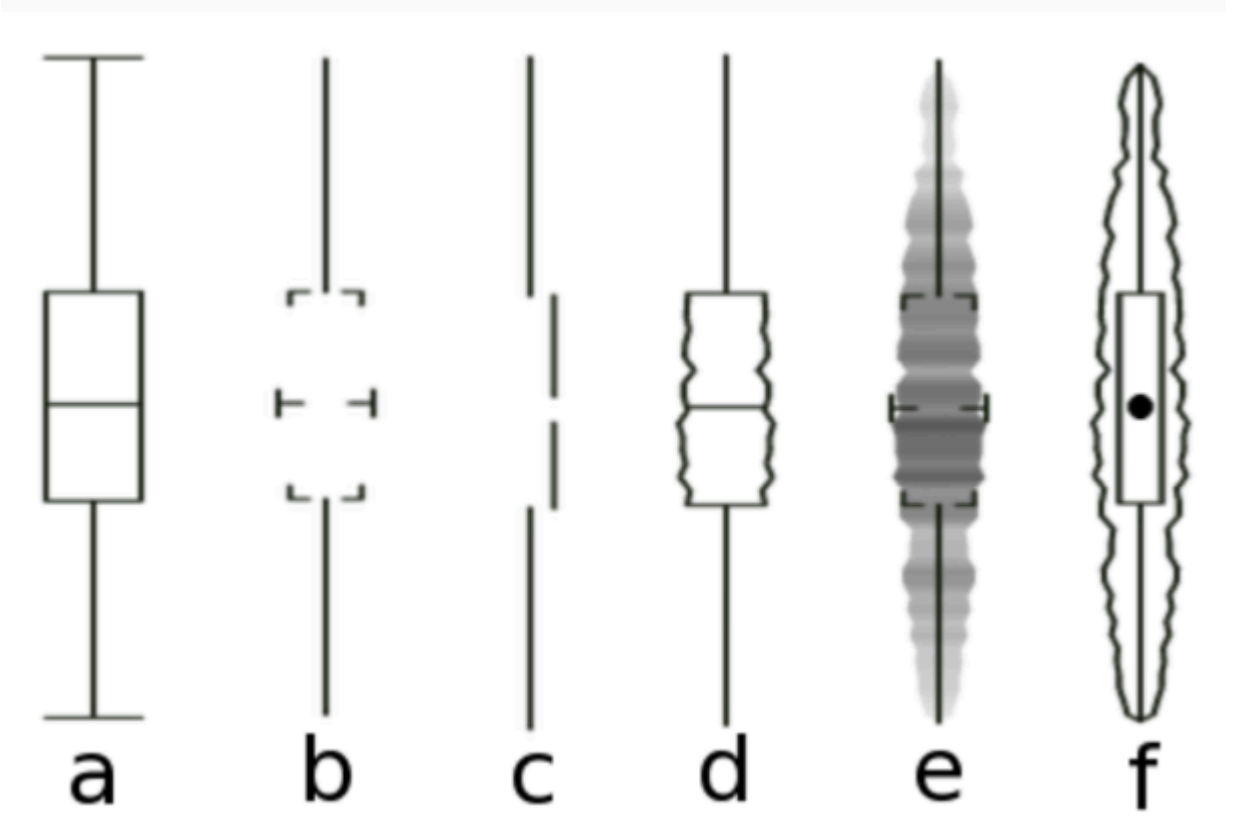
What is it?

> ## Above all else, show the data

-Edward Tufte

- Data-Ink Ratio = Ink devoted to the data / total ink used to produce the figure
- Common goal: Maximize the data-ink ratio

Example



- First thought might be - Cool!

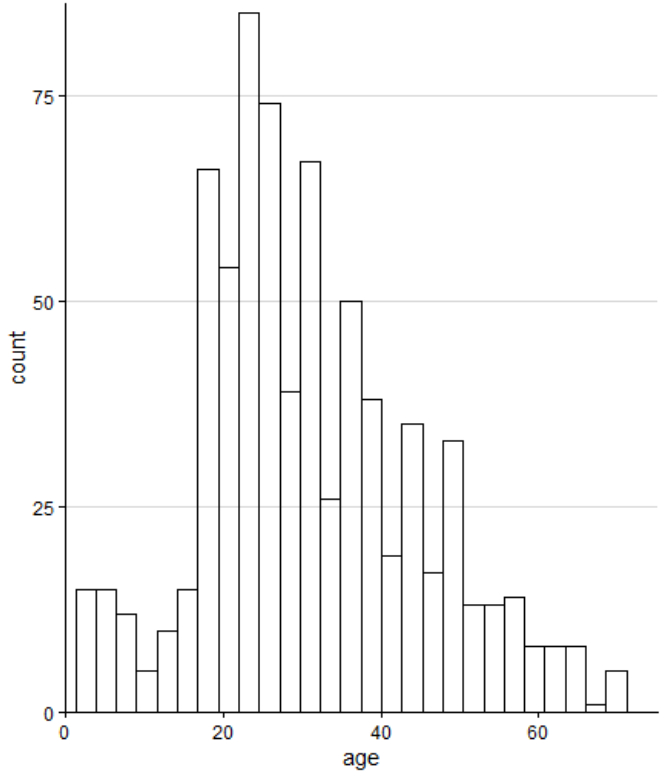

NOT SO FAST, MY FRIEND!

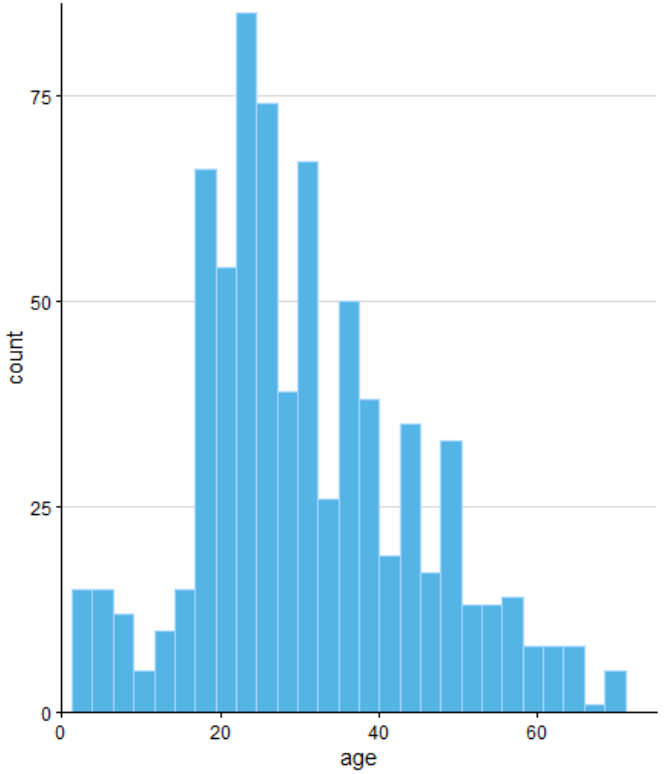
Minimize cognitive load

- Empirically, Tufte's plot was **the most difficult** for viewers to interpret.
- Visual cues (labels, gridlines) *reduce* the data-ink ratio, but can also reduce cognitive load.

Another example

Which do you prefer?

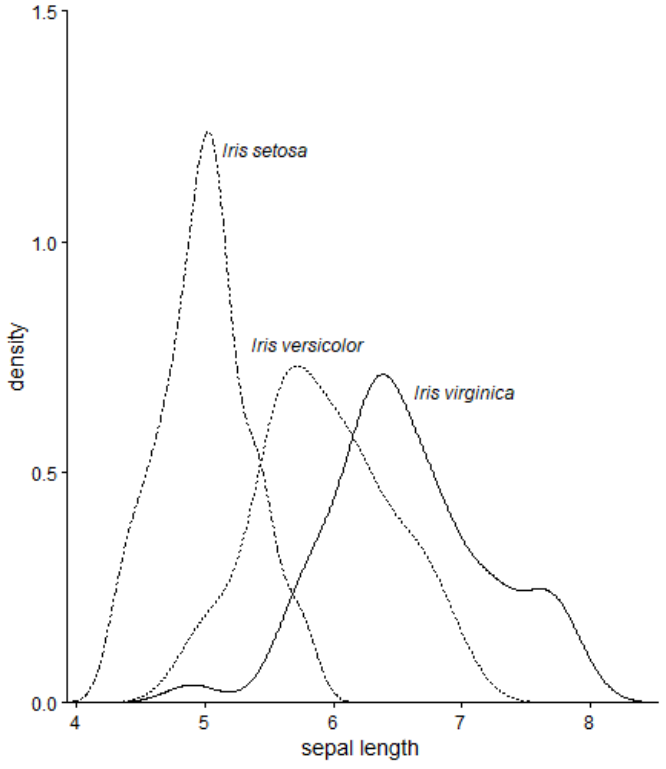


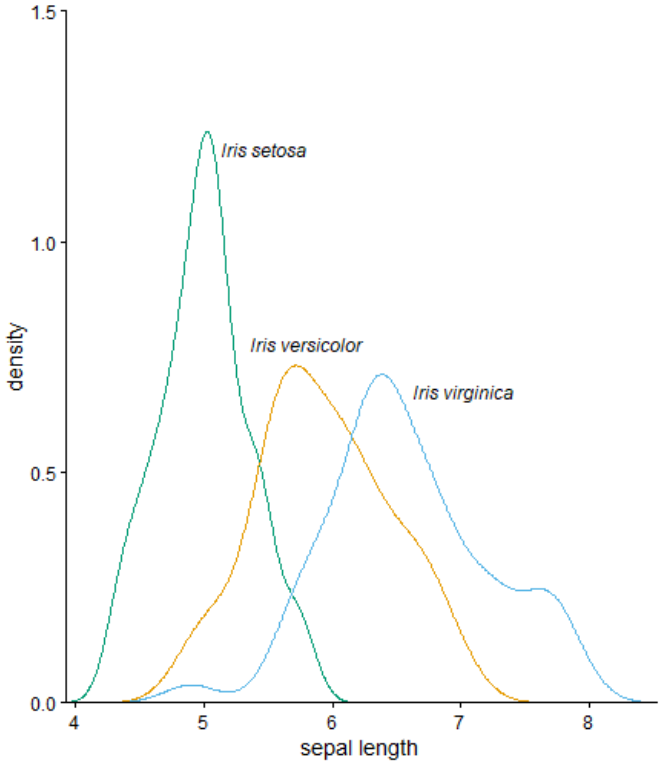


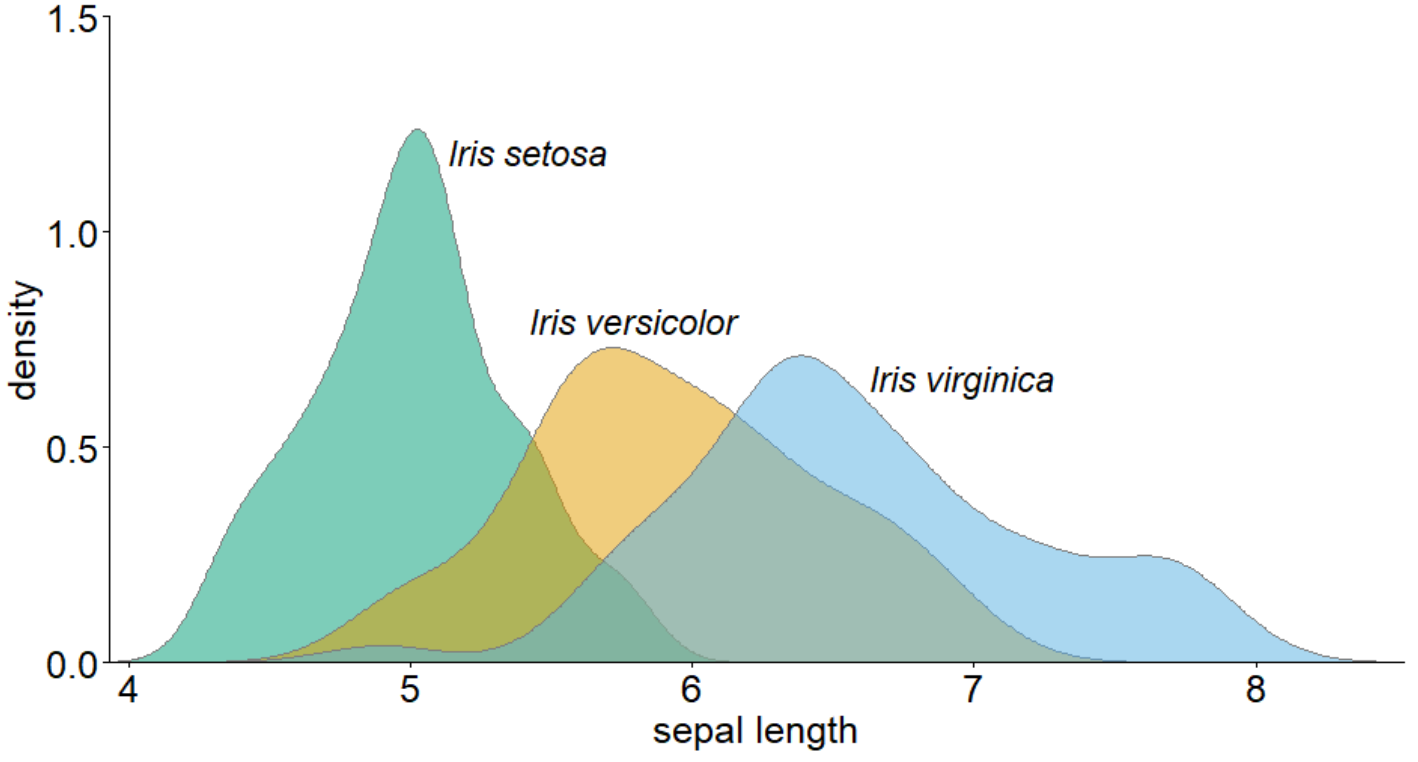
Advice from Wilke

> Whenever possible, visualize your data with solid, colored shapes rather than with lines that outline those shapes. Solid shapes are more easily perceived, are less likely to create visual artifacts or optical illusions, and do more immediately convey amounts than do outlines.

Another example






1.5
1.0
0.5
0.0
density
Iris setosa
Iris versicolor
Iris virginica
4
5
6
7
8
sepal length


MONSTROUS COSTS
Total House and Senate campaign expenditures, in millions
$300
250
200
150
100
50
1972
'74
'76
'78
'80
'82 est.
VOTE

This?

The takeaway?

- It can often be helpful to remove "chart junk"
  - Remove background
  - Unnecessary frills
  - Certainly don't use 3D when it's not clearly warranted

But...

- Infographics can often be more memorable

Compromise?

In some cases, it may be easy and more memorable to use glyphs instead of points or squares

- Install packages

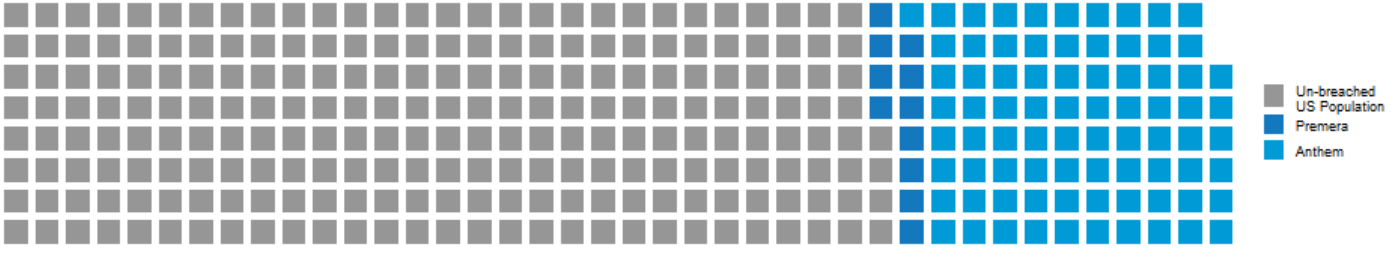
```
install.packages("extrafont")
remotes::install_github("wch/extrafontdb")
remotes::install_github("wch/Rttf2pt1")
remotes::install_github("hrbrmstr/waffle")
```

- Create data

```
parts <- c(`Un-breached\nUS Population` = (318 - 11 - 79),
           `Premera` = 11,
           `Anthem` = 79)
```

Basic plot

```
library(waffle)
waffle(parts,
       rows = 8,
       colors = c("#969696", "#1879bf", "#009bda"))
```



Glyph plot

Doesn't seem to work anymore... 🤷

- Download and install `fontawesome-webfont.ttf` on your machine locally (see here)
- Import new fonts (including glyphs, via font awesome)

```
library(extrafont)
font_import()
loadfonts()
```

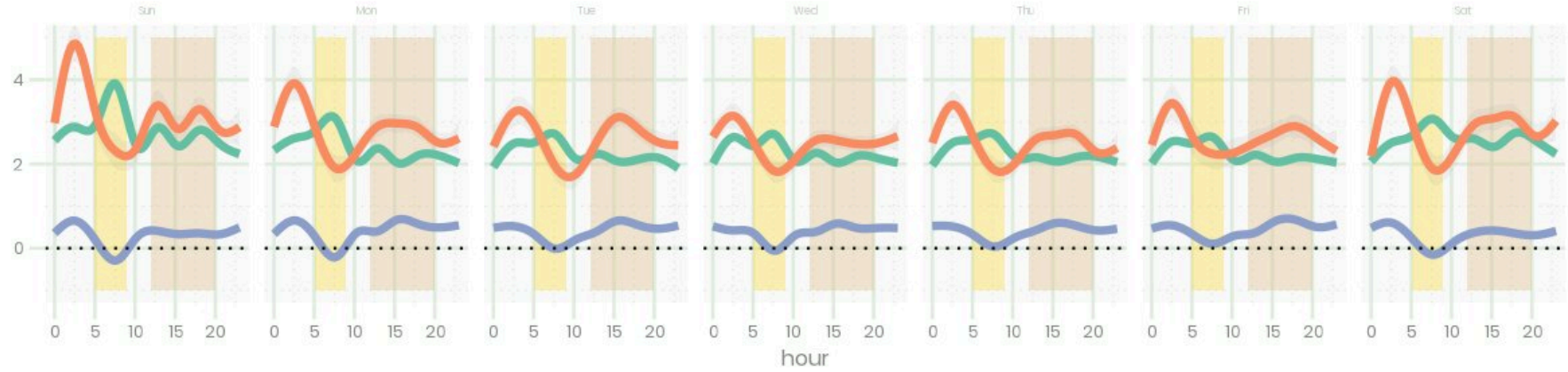
```
waffle(parts/10,
       rows = 3,
       colors = c("#969696", "#1879bf", "#009bda"),
       use_glyph = "medkit",
       size = 8
       ) +
  expand_limits(
    y=c(0,4)
  )
```

Should look like this

Despite glyphs not (easily) working anymore, I still recommend you check it out. It's a neat package and does have some integration with ggplot2 now.

#rstats aren't early morning grinders

the average number of retweets and hashtags per #rstats tweet, by day and time



In general, more hashtags = more retweets. Early morning tweets usually contain more hashtags than tweets later in the day. Fewer people are on twitter at this time, so the apparent effectiveness of hashtags at provoking retweets is reduced. This begs the question: Why do people use more hashtags in the morning?

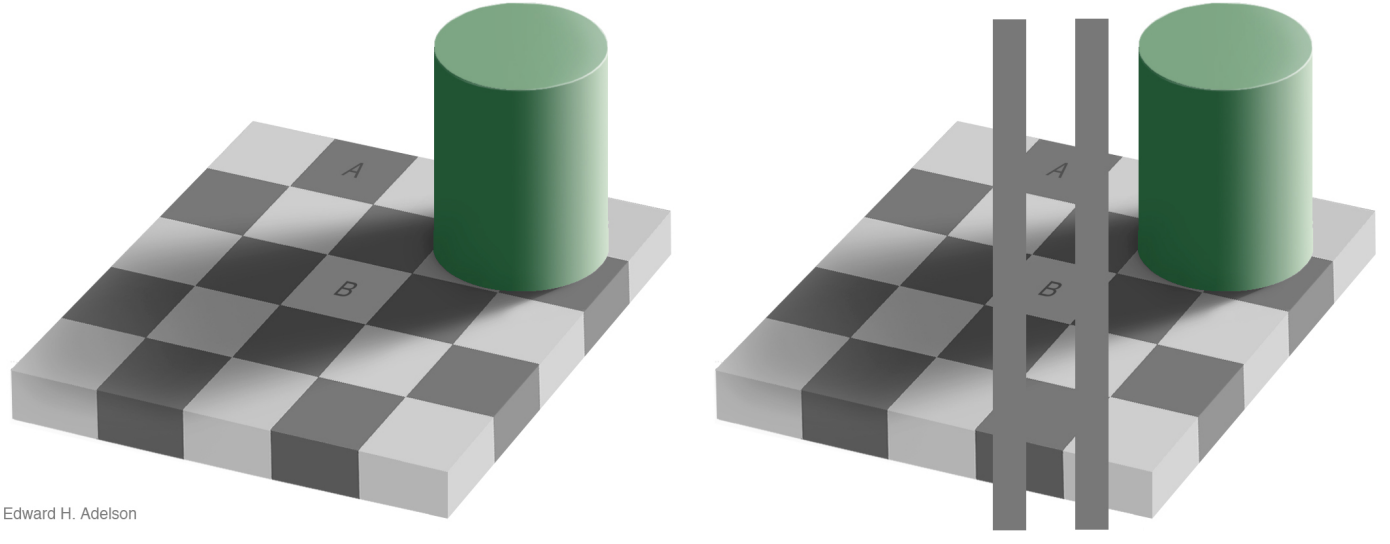
Hashtags used in the mid-afternoon, when most people are on twitter, are almost twice as effective at provoking retweets! Also, while most days see a single pronounced increase in tweets, Sunday afternoon sees two distinct bumps in #rstats tweets.

A #TidyTuesday adventure
Data from rtweet.info
Analysis @Tanner Koomar
Design @Tanner Koomar
https://github.com/tkoomar/tidytuesday/blob/master/work/2019-01-01.md

You can create them!

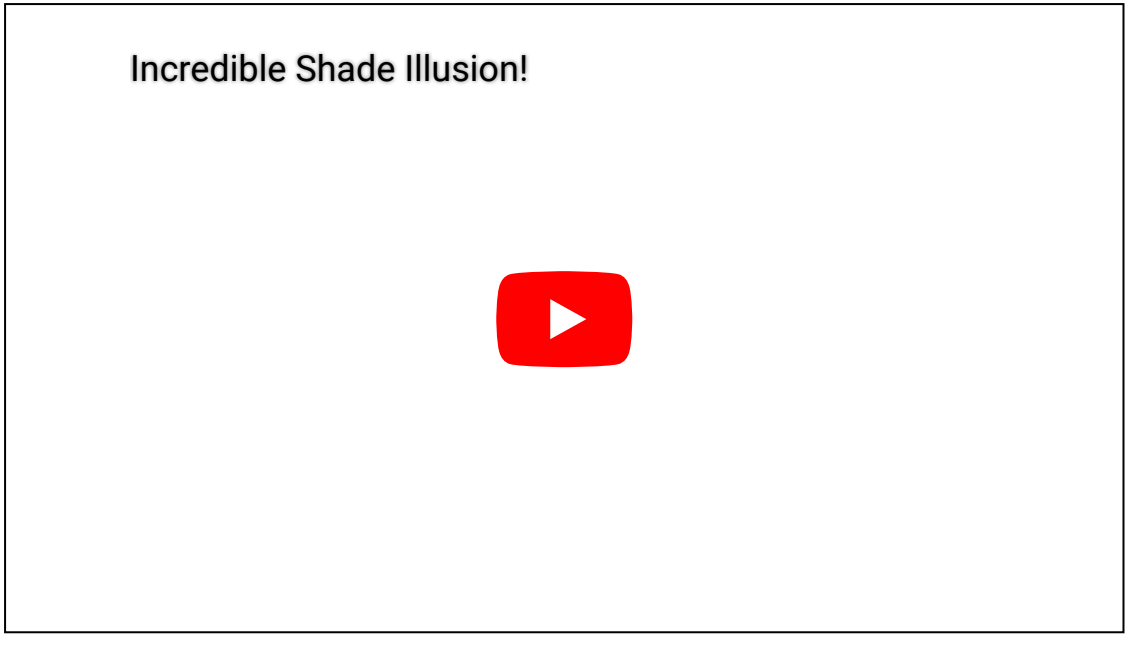
- Create plots
- Use illustrator or similar to put them together
- Add some annotations
- Consider using glyphs for greater memory
- You can do a lot in R without going to illustrator etc. by just using **{patchwork}** or **{cowplot}**

More visual properties

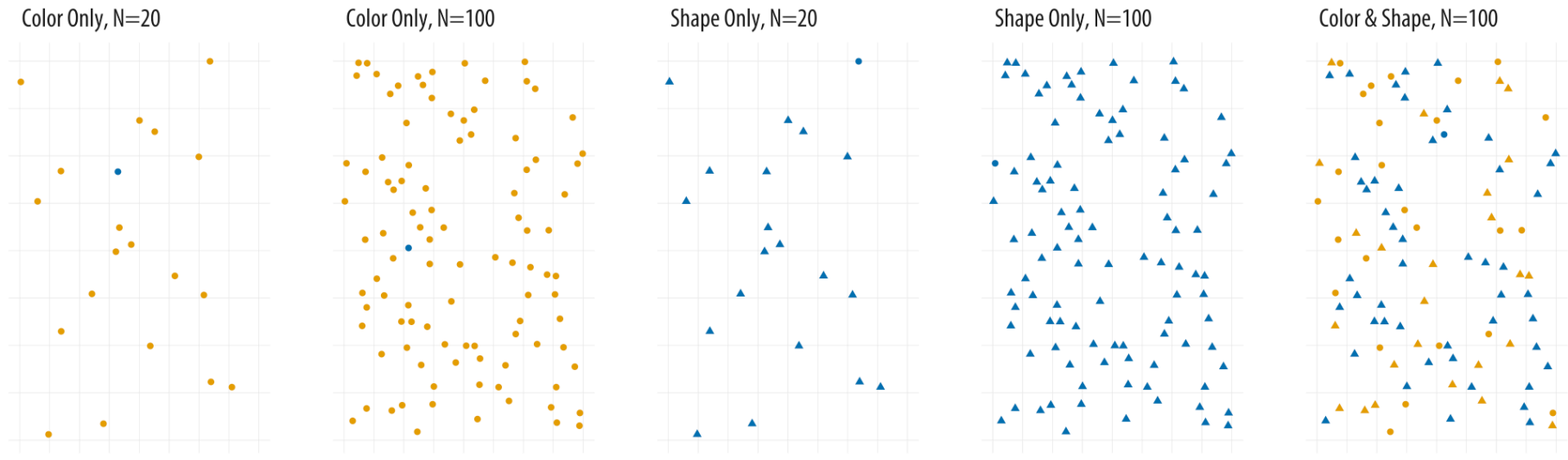


Edward H. Adelson

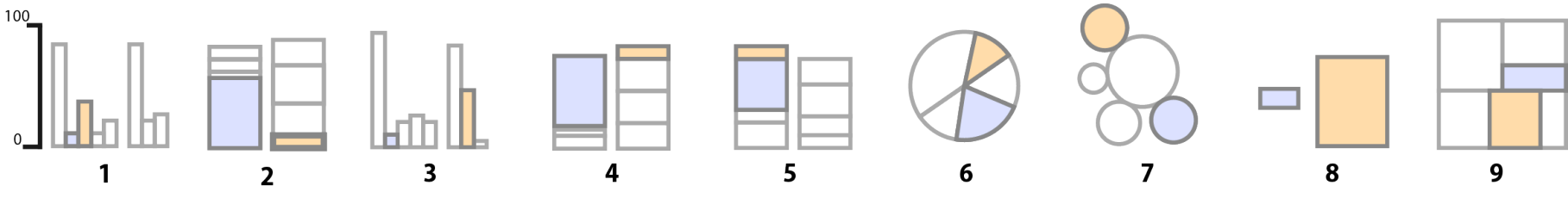
Or in real life

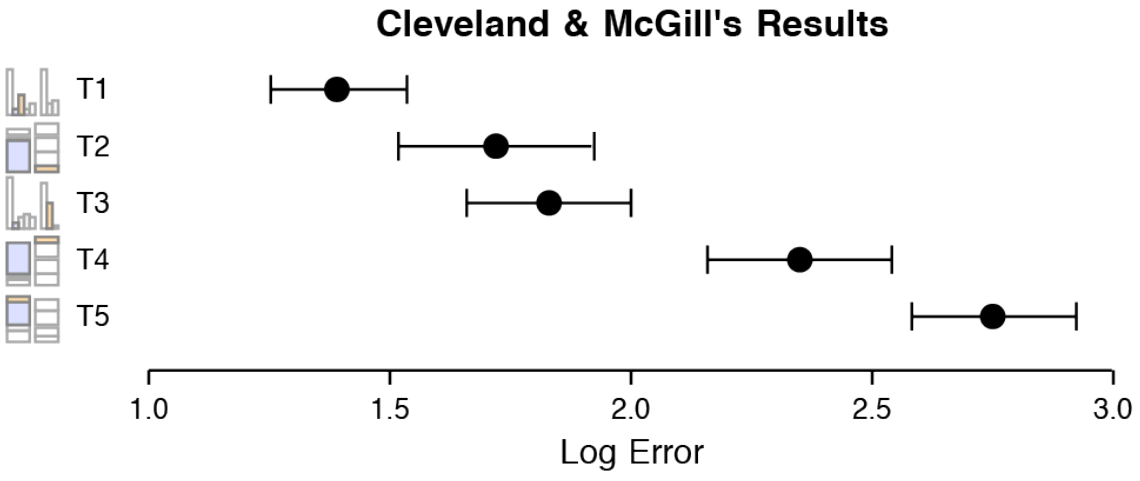


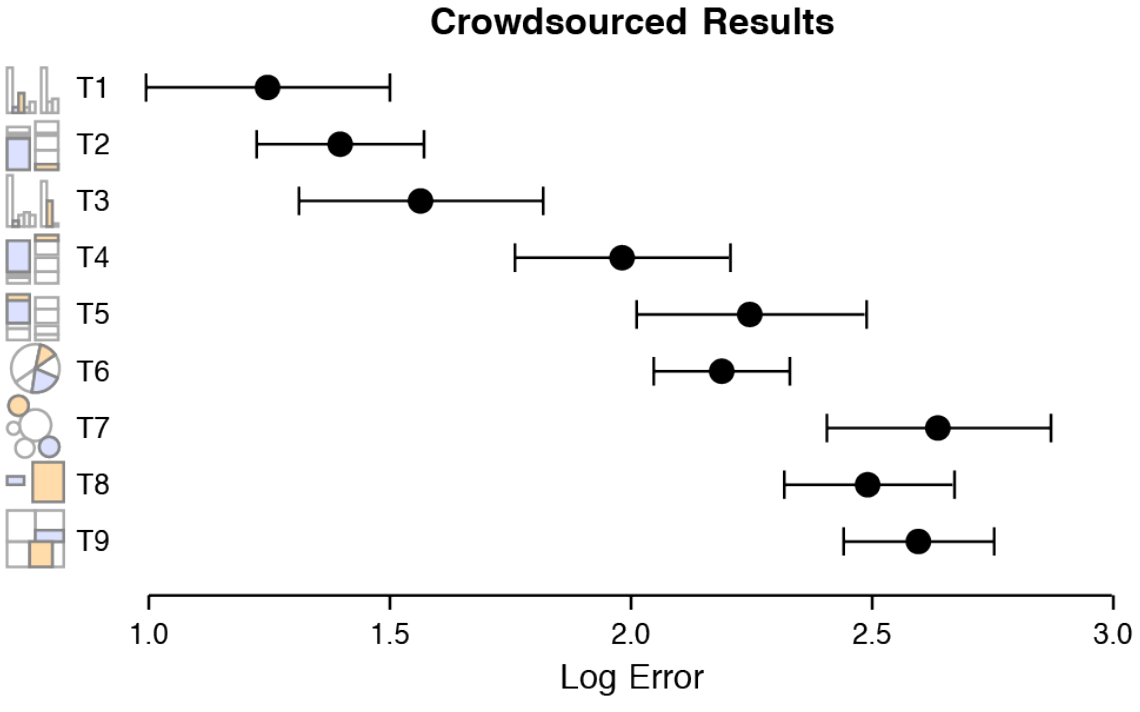
Where's the blue circle in each plot?



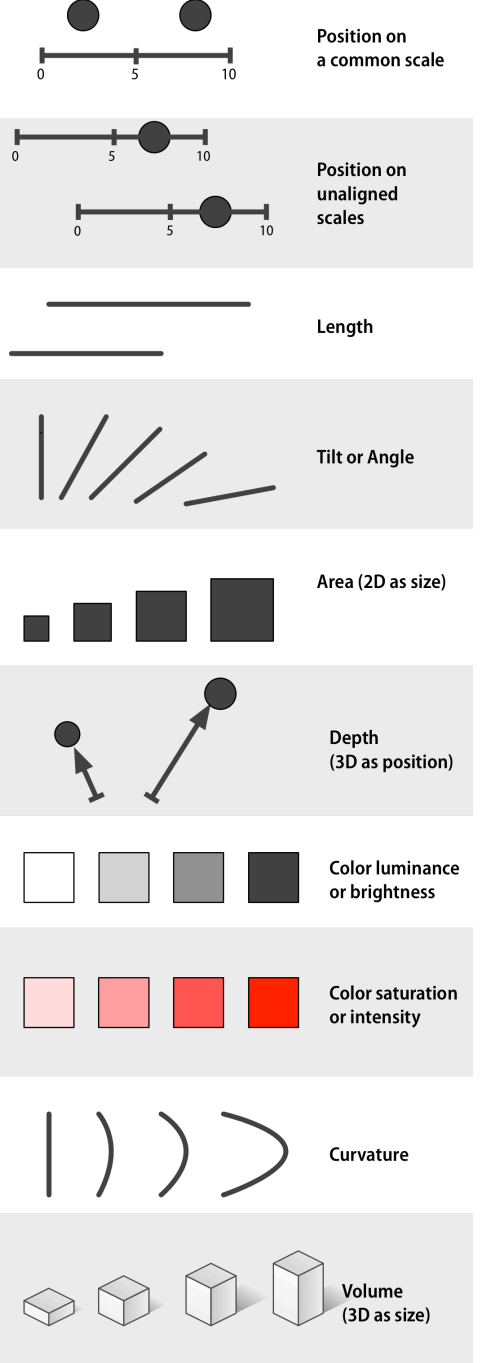
What are we good at perceiving?



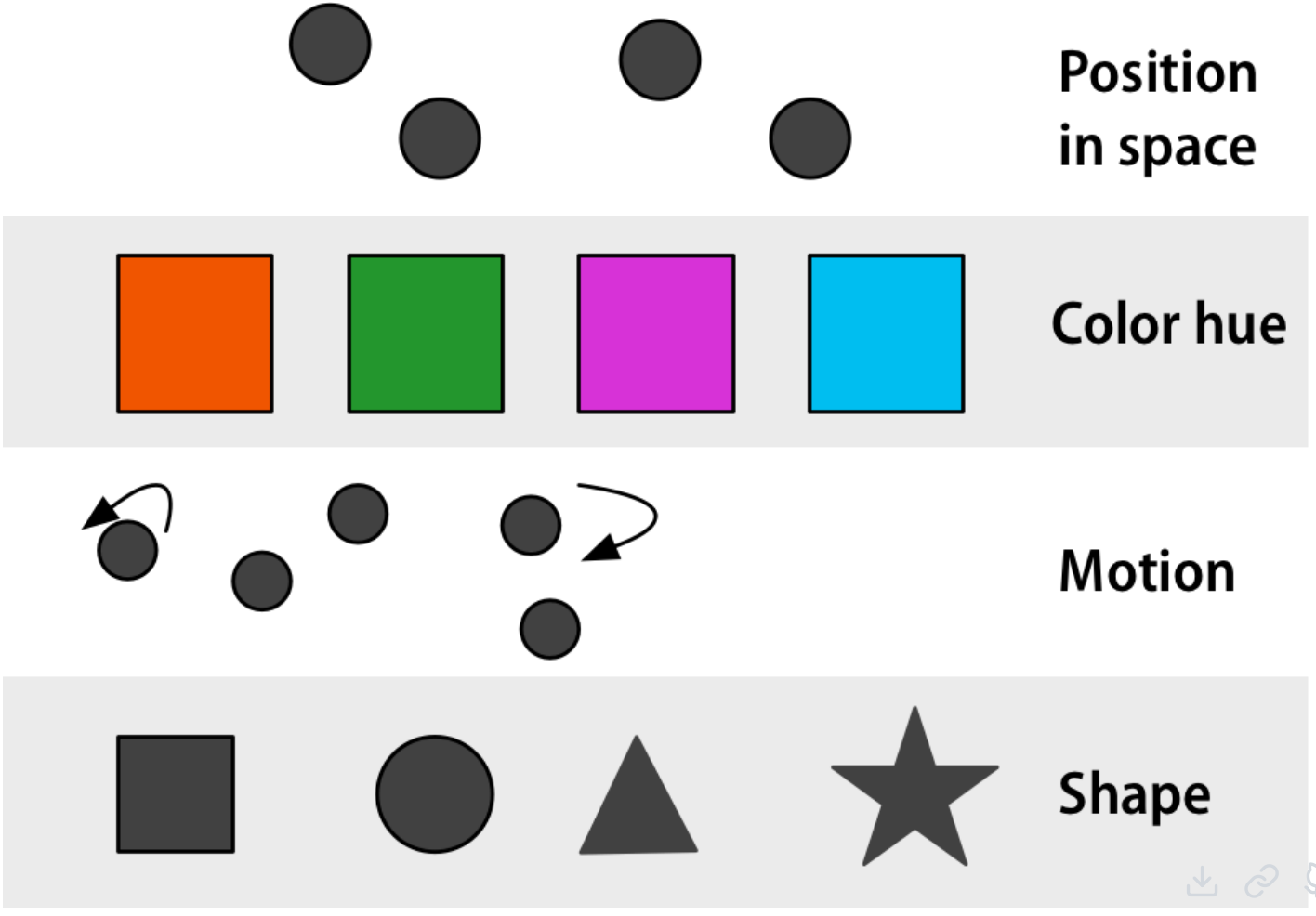

Cleveland & McGill's Results
T1
T2
T3
T4
T5
1.0
1.5
2.0
2.5
3.0
Log Error


Crowdsourced Results
T1
T2
T3
T4
T5
T6
T7
T8
T9
1.0
1.5
2.0
2.5
3.0
Log Error

Ordered data mappings: Ranked



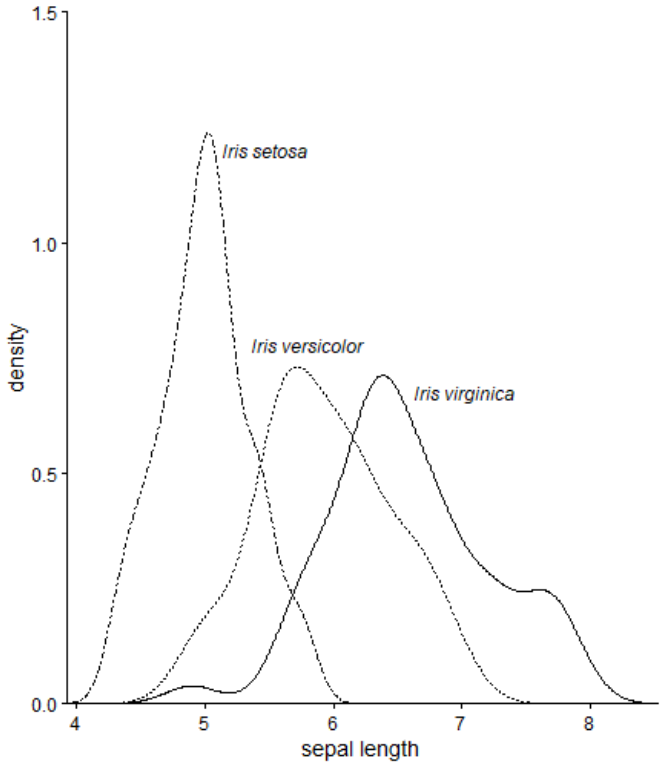
Unordered data mappings

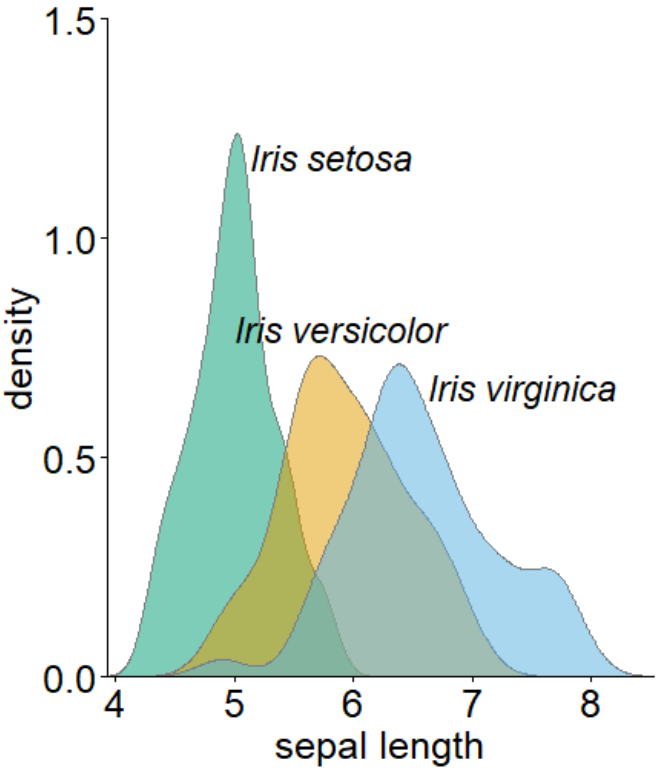


Some things to avoid

Line drawings

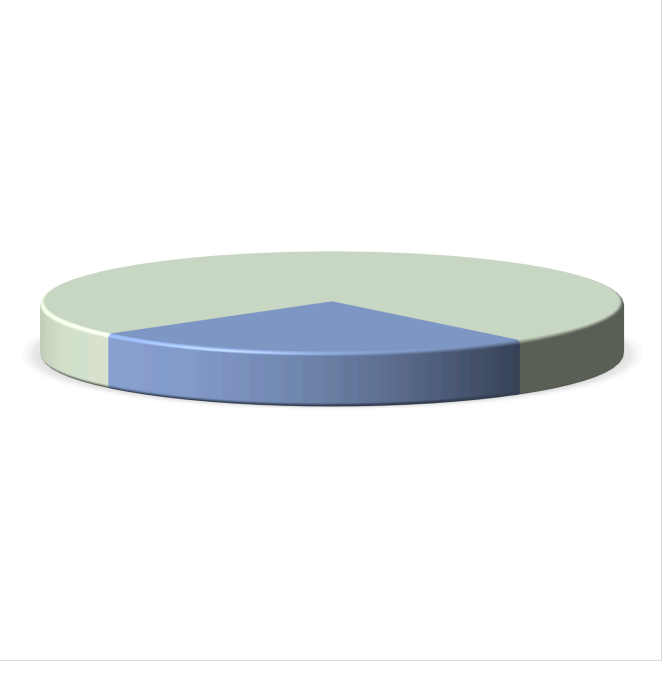
As discussed earlier

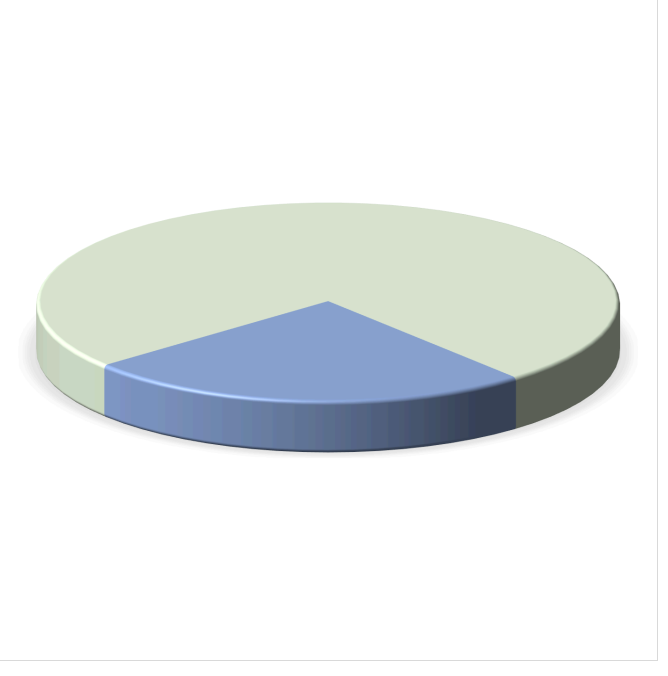




Much worse

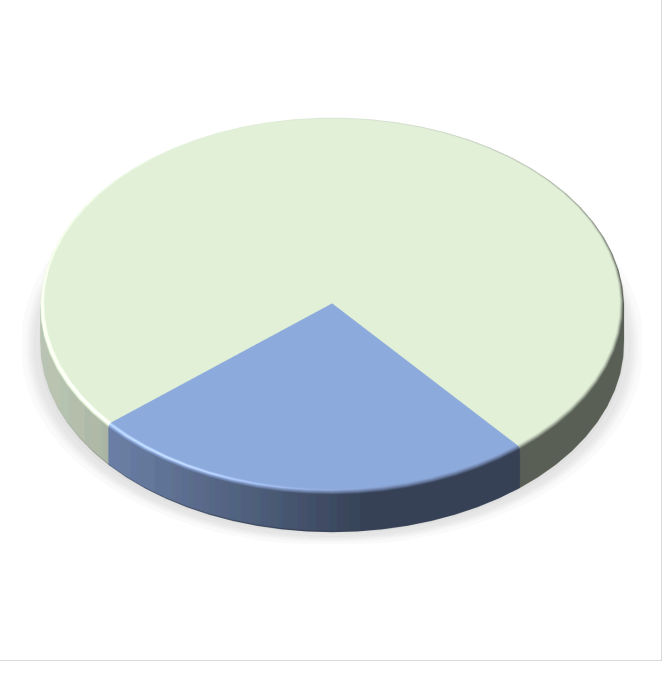
Unnecessary 3D

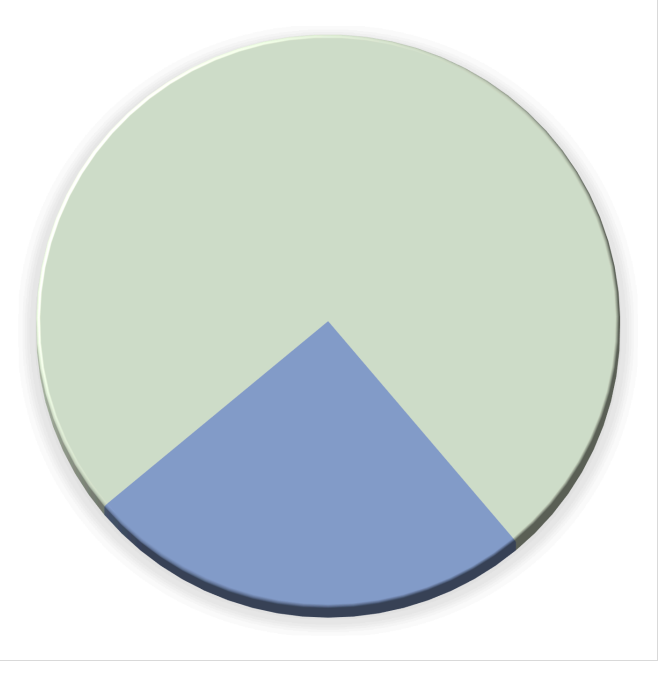




Much worse

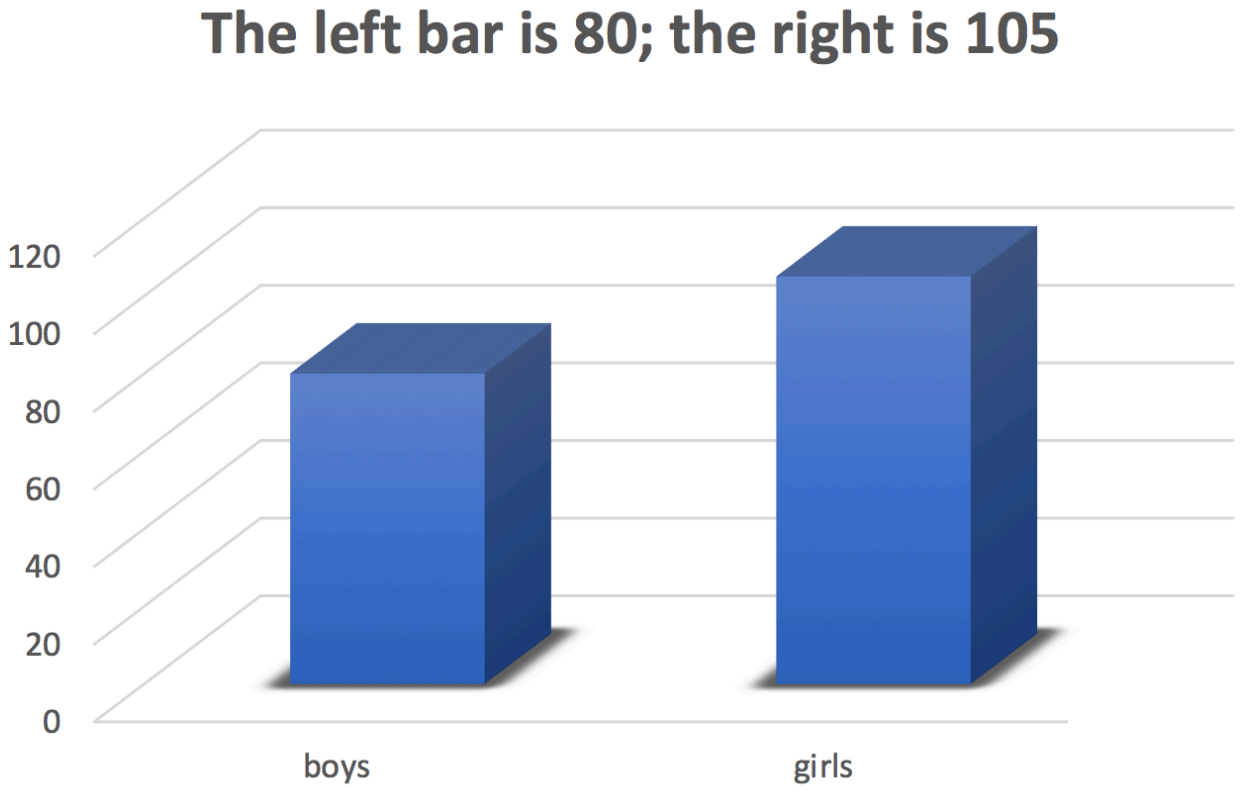
Unnecessary 3D



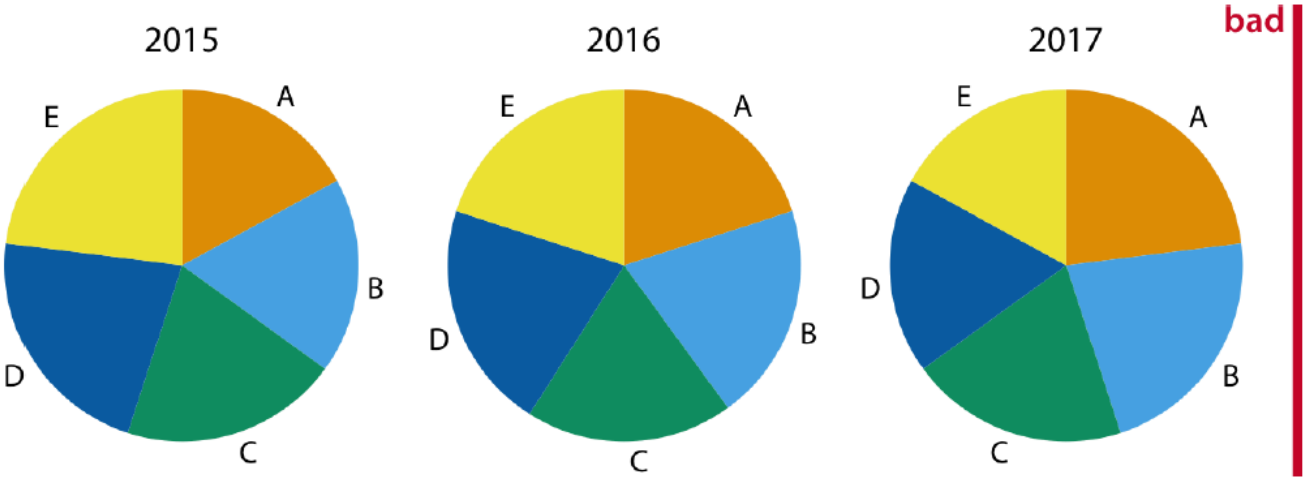


Horrid example

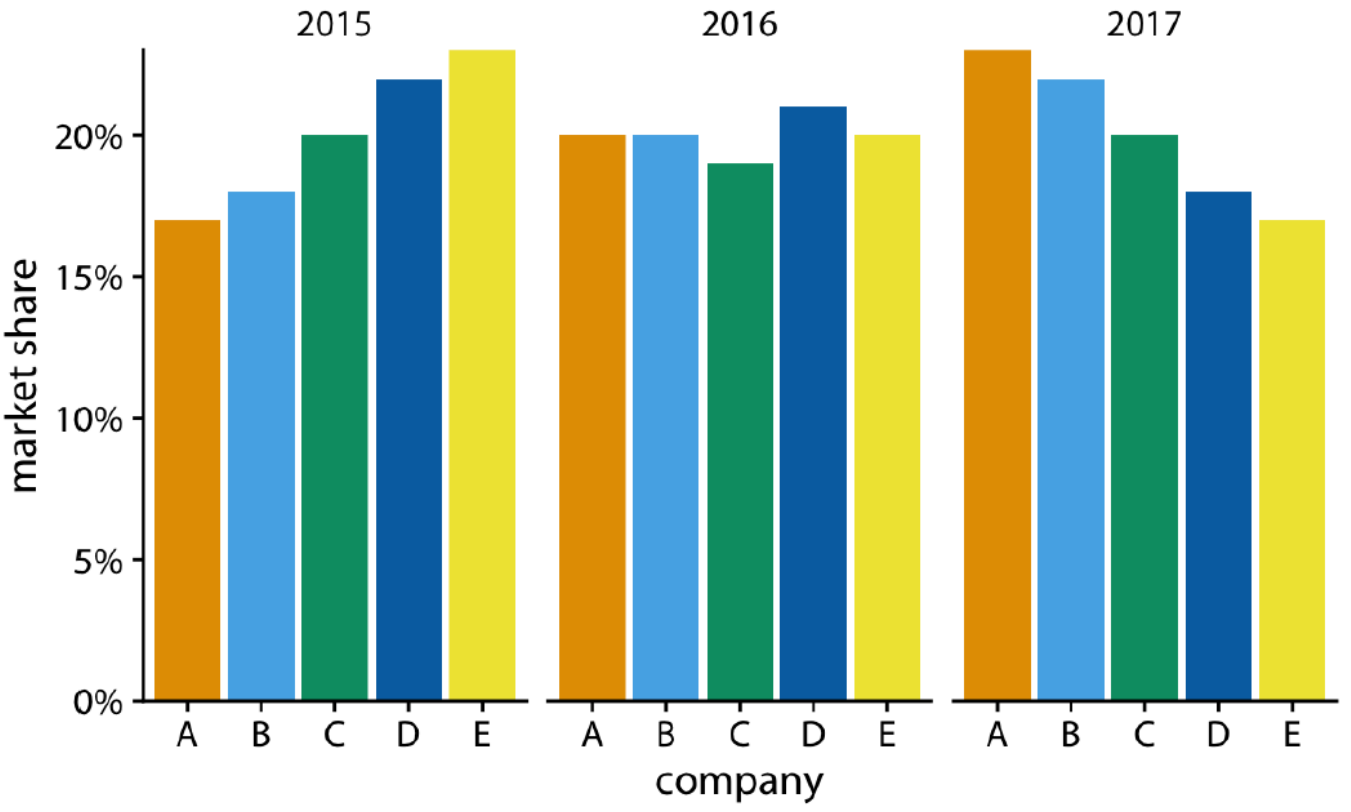
Used relatively regularly



Pie charts w/lots of categories

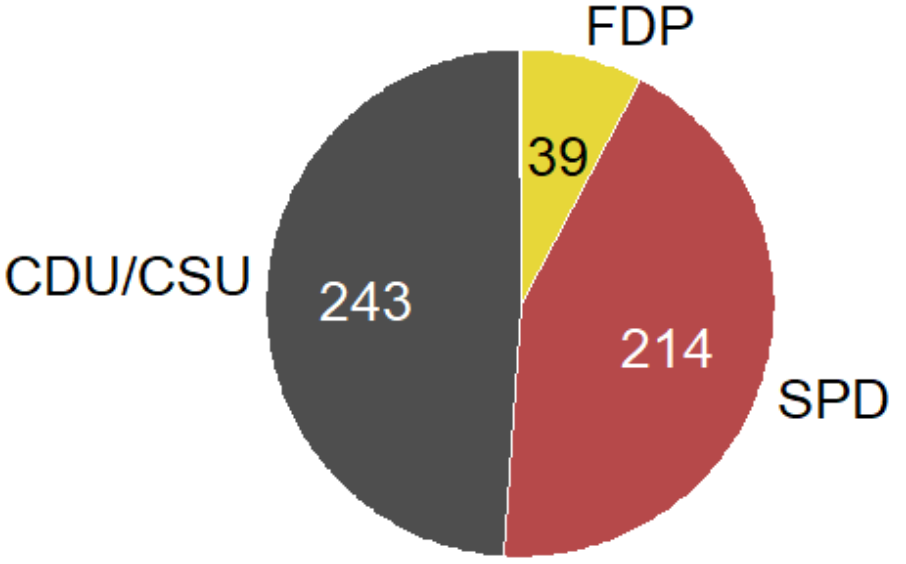


Alternative representation



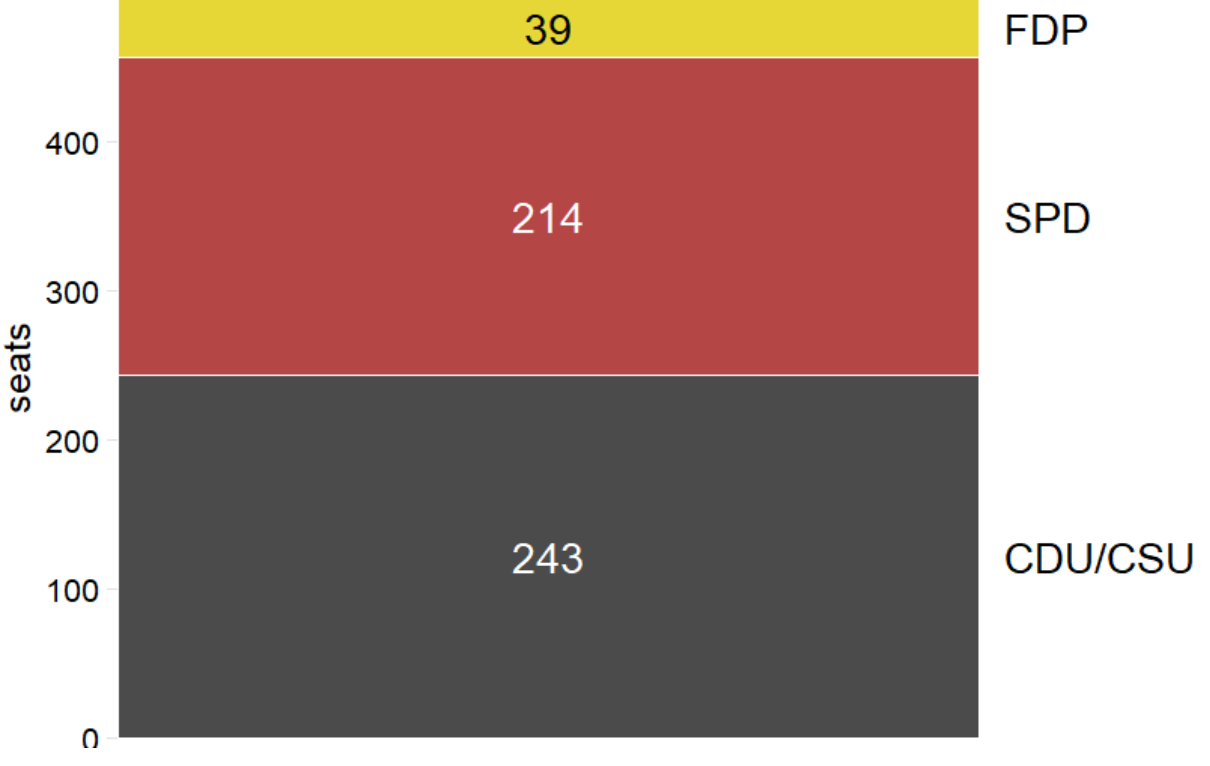
A case for pie charts

- n categories low,
- differences are relatively large
- familiar for some audiences

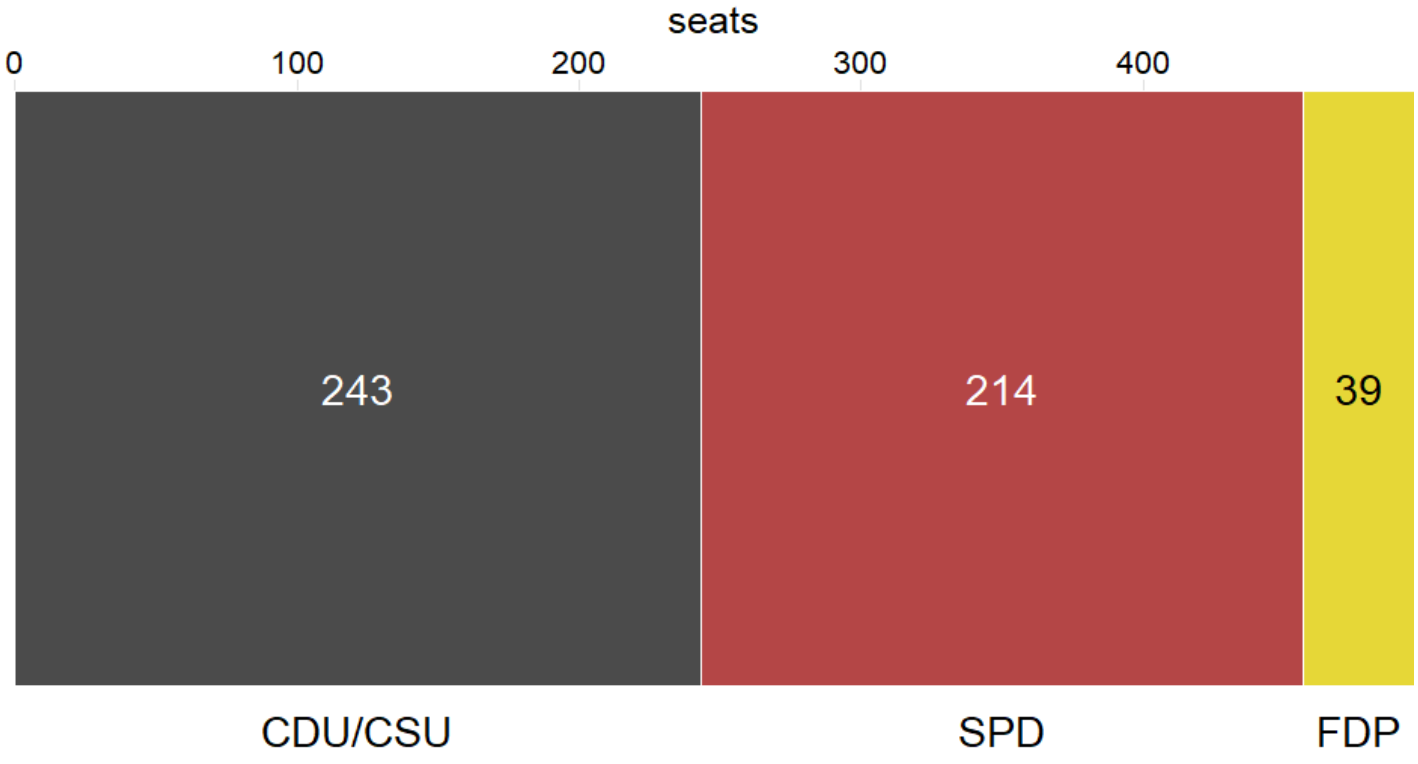


The anatomy of a pie chart

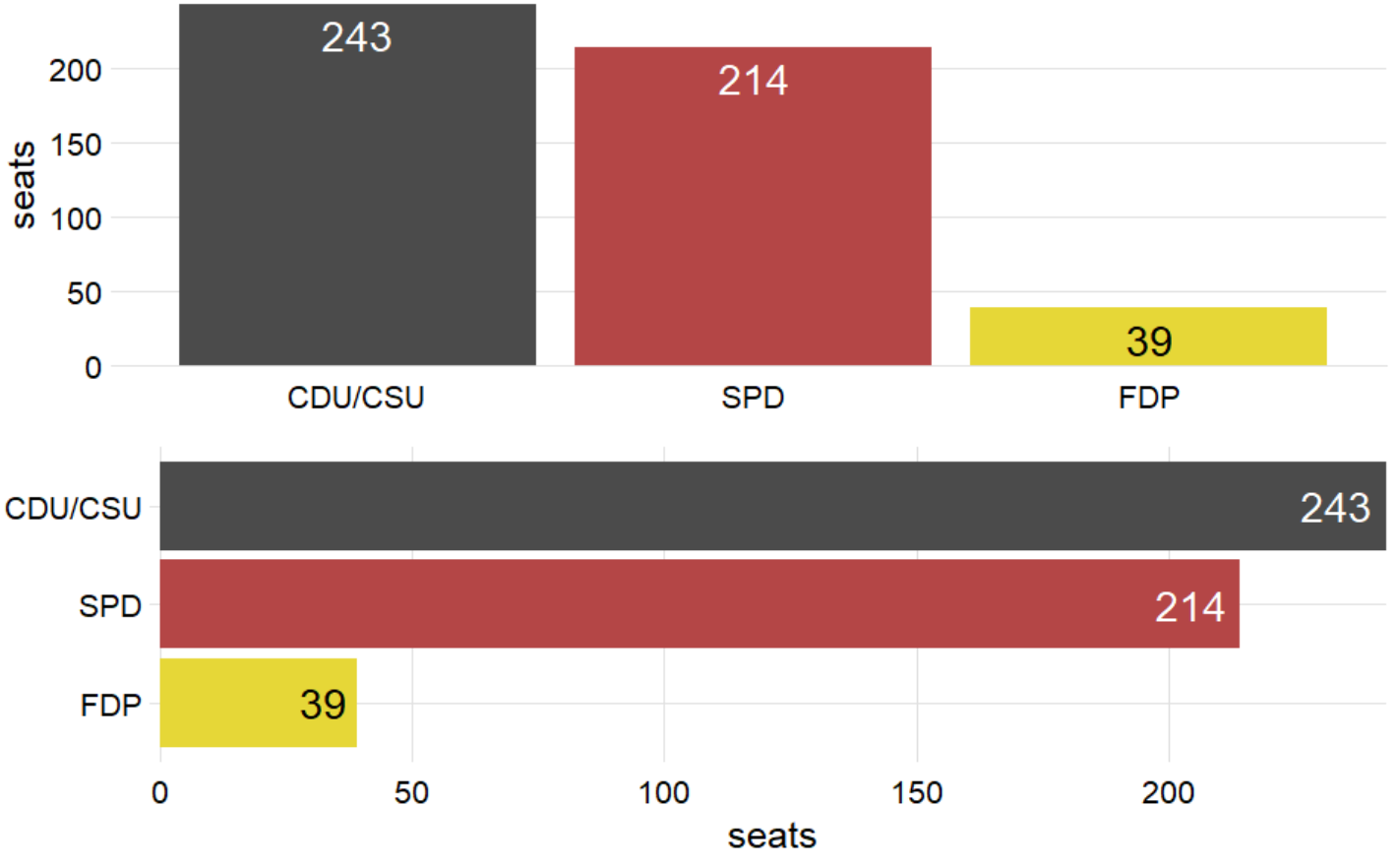
Pie charts are just stacked bar charts with a radial coordinate system



Alternative represenation

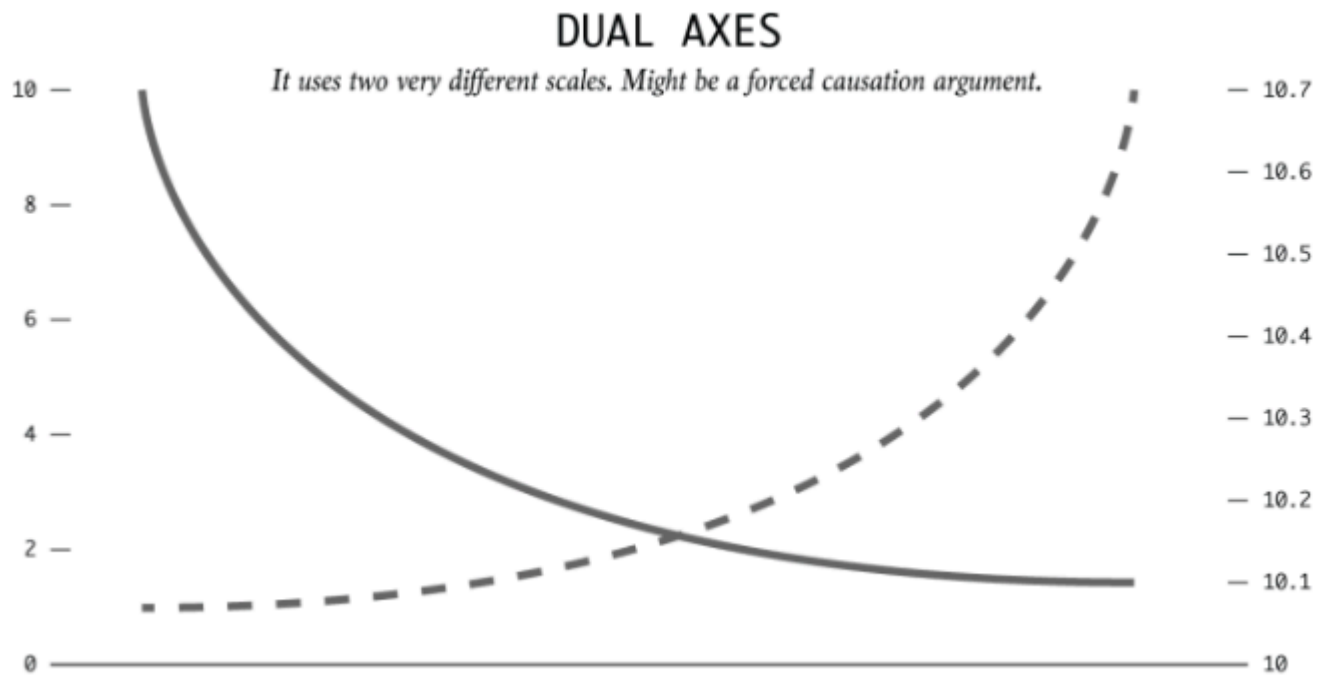


Or one of these

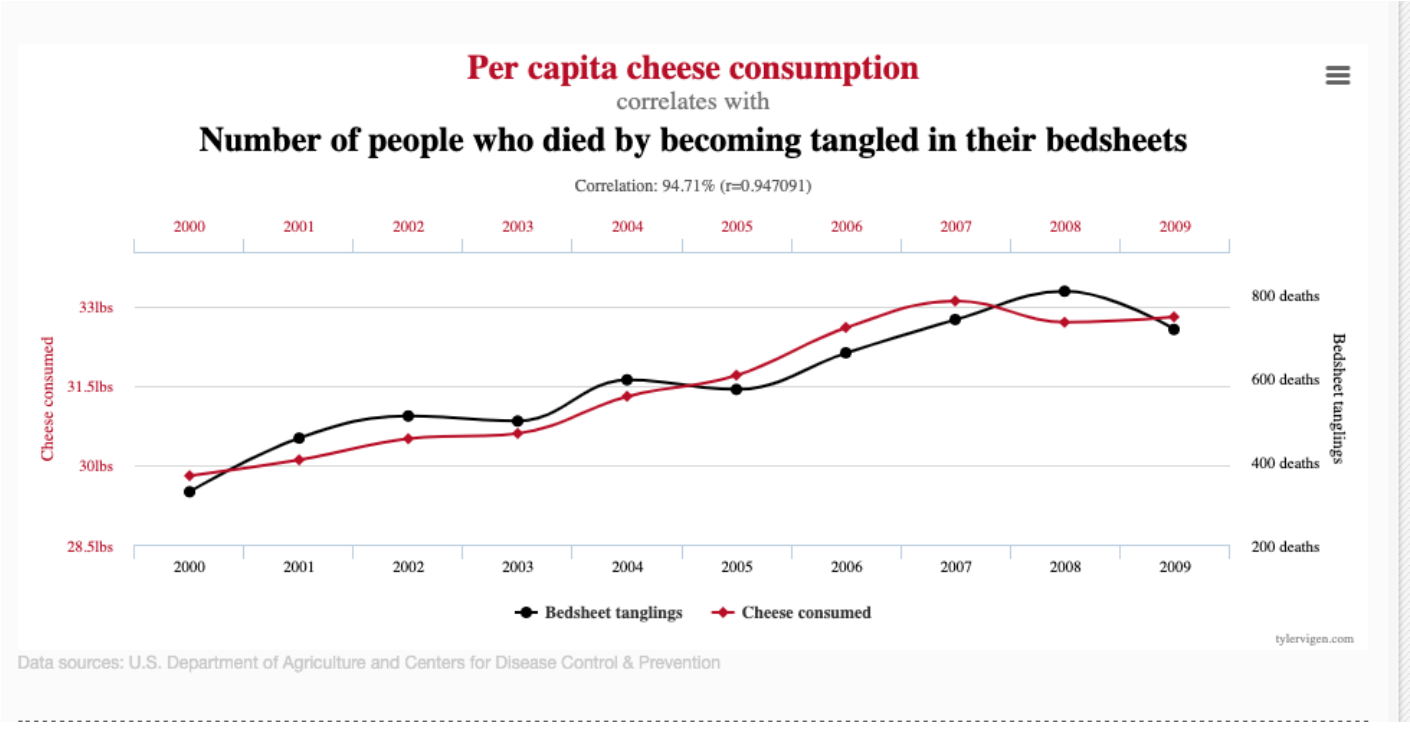


Dual axes

- One exception - if second axis is a direct transformation of the first
  - e.g., Miles/Kilometers, Fahrenheit/Celsius

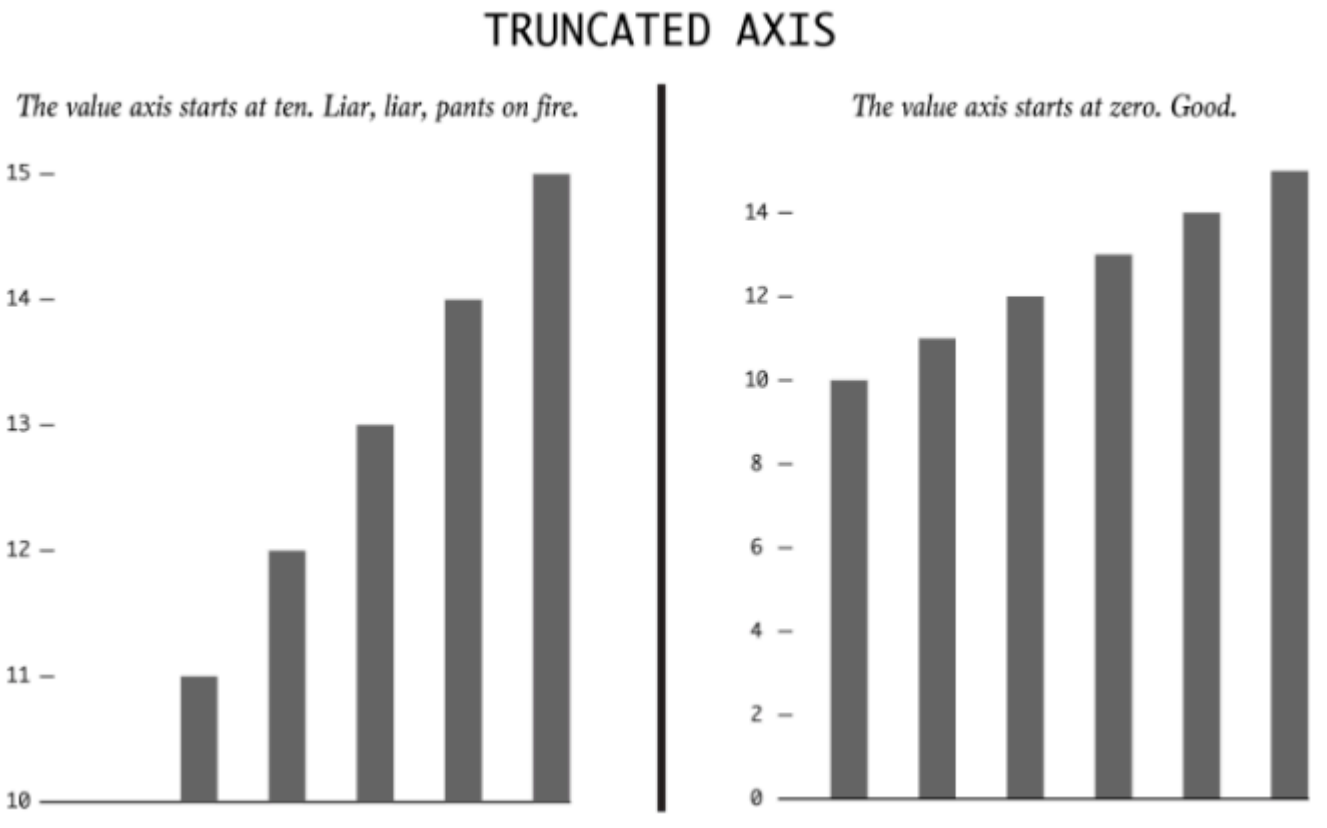


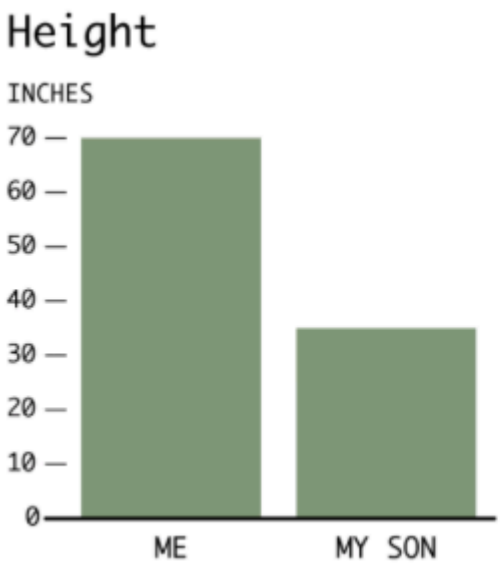
Another example



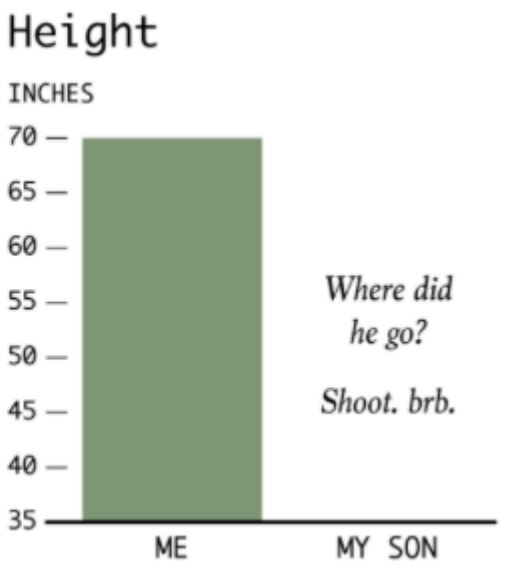
See more examples here

Truncated axes




Height
INCHES
70
60
50
40
30
20
10
0
ME
MY SON

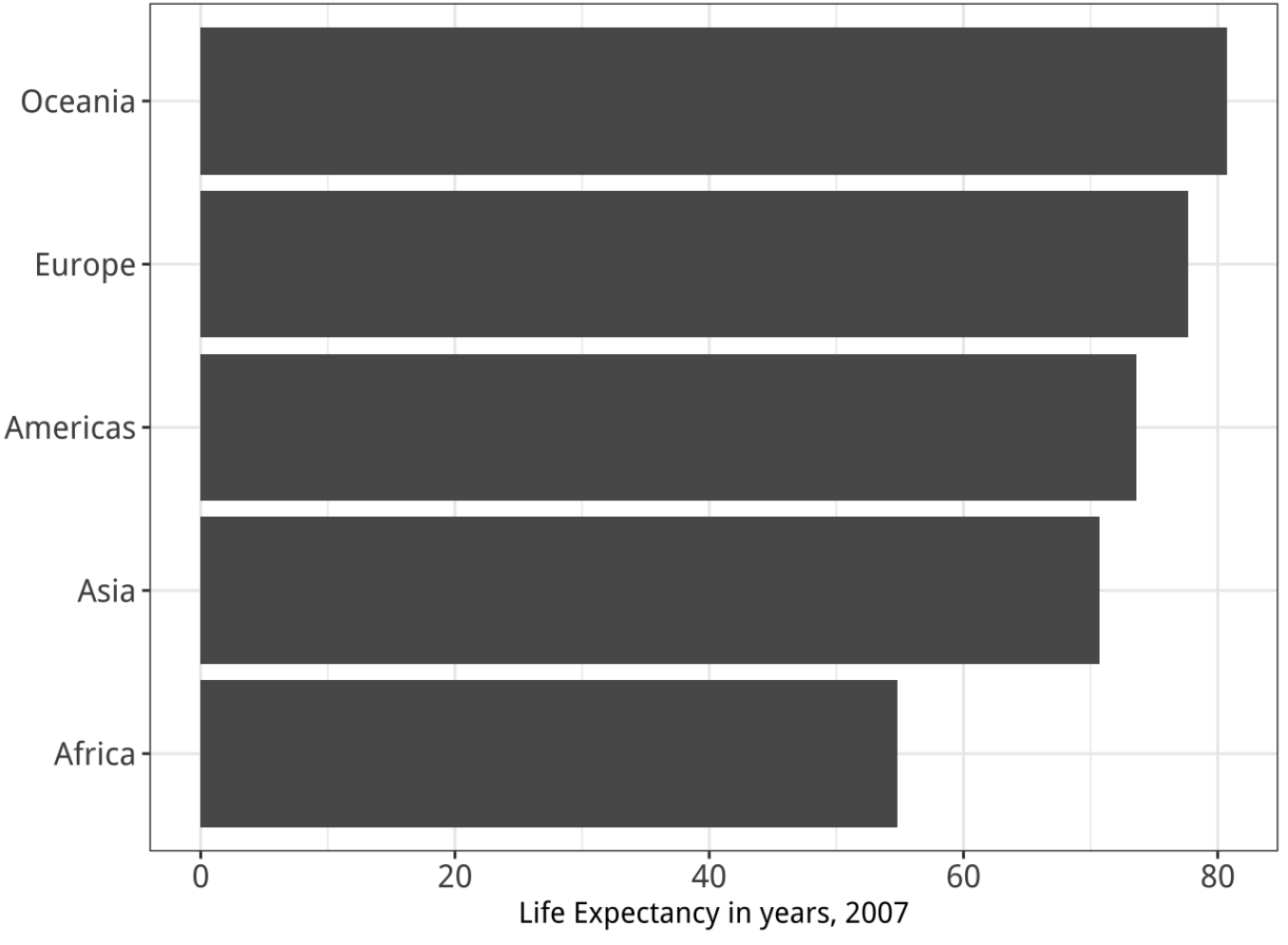
vs.


Height
INCHES
70
65
60
55
50
45
40
35
ME
MY SON
Where did he go?
Shoot. brb.

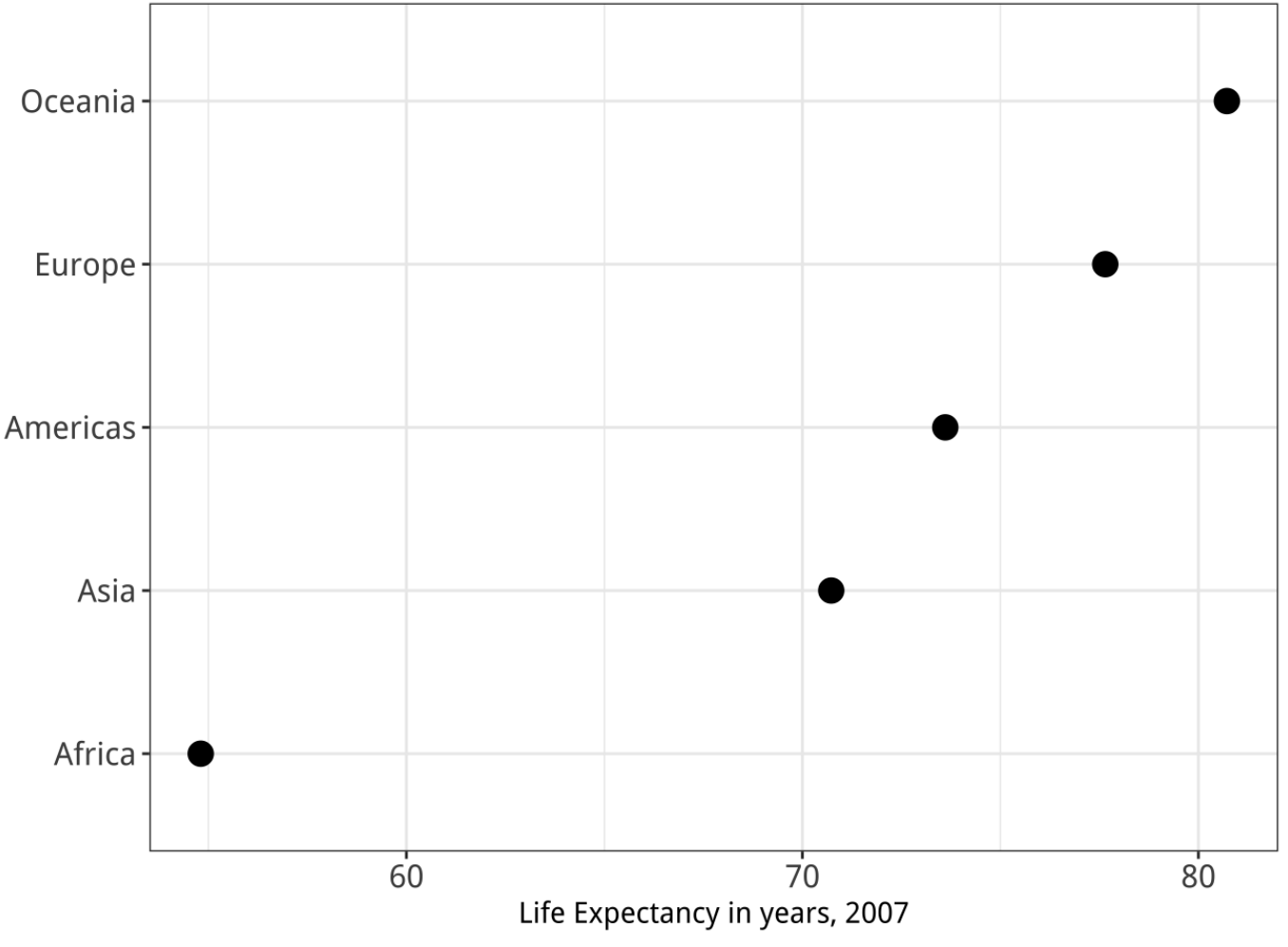
Not always a bad thing

> It is tempting to lay down inflexible rules about what to do in terms of producing your graphs, and to dismiss people who don't follow them as producing junk charts or lying with statistics. But **being honest with your data is a bigger problem than can be solved by rules of thumb** about making graphs. In this case there is a moderate level of agreement that bar charts should generally include a zero baseline (or equivalent) given that bars encode their variables as lengths. But it would be a mistake to think that a dot plot was by the same token deliberately misleading, just because it kept itself to the range of the data instead.

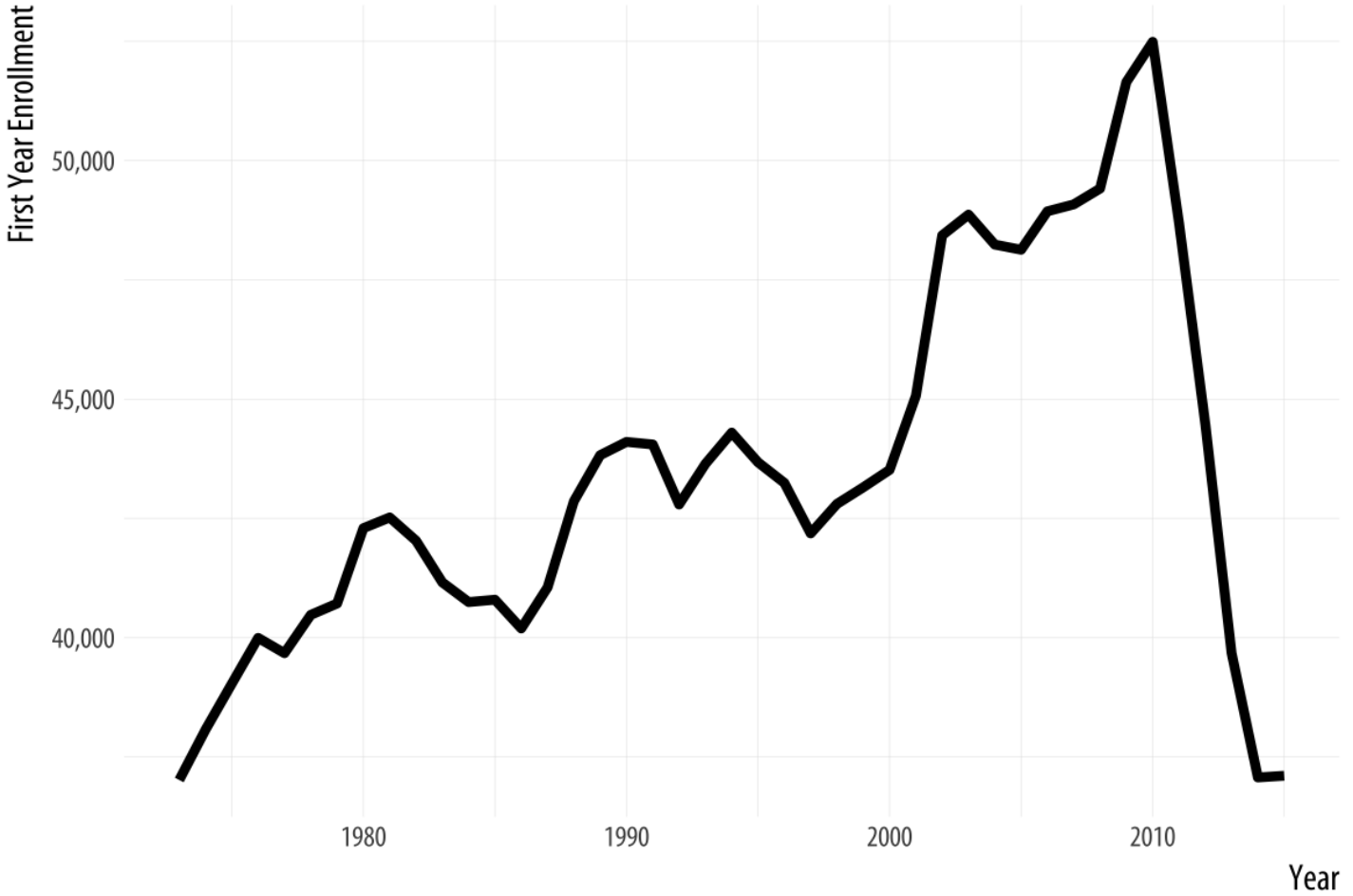
Bars


Oceania
Europe
Americas
Asia
Africa
0
20
40
60
80
Life Expectancy in years, 2007

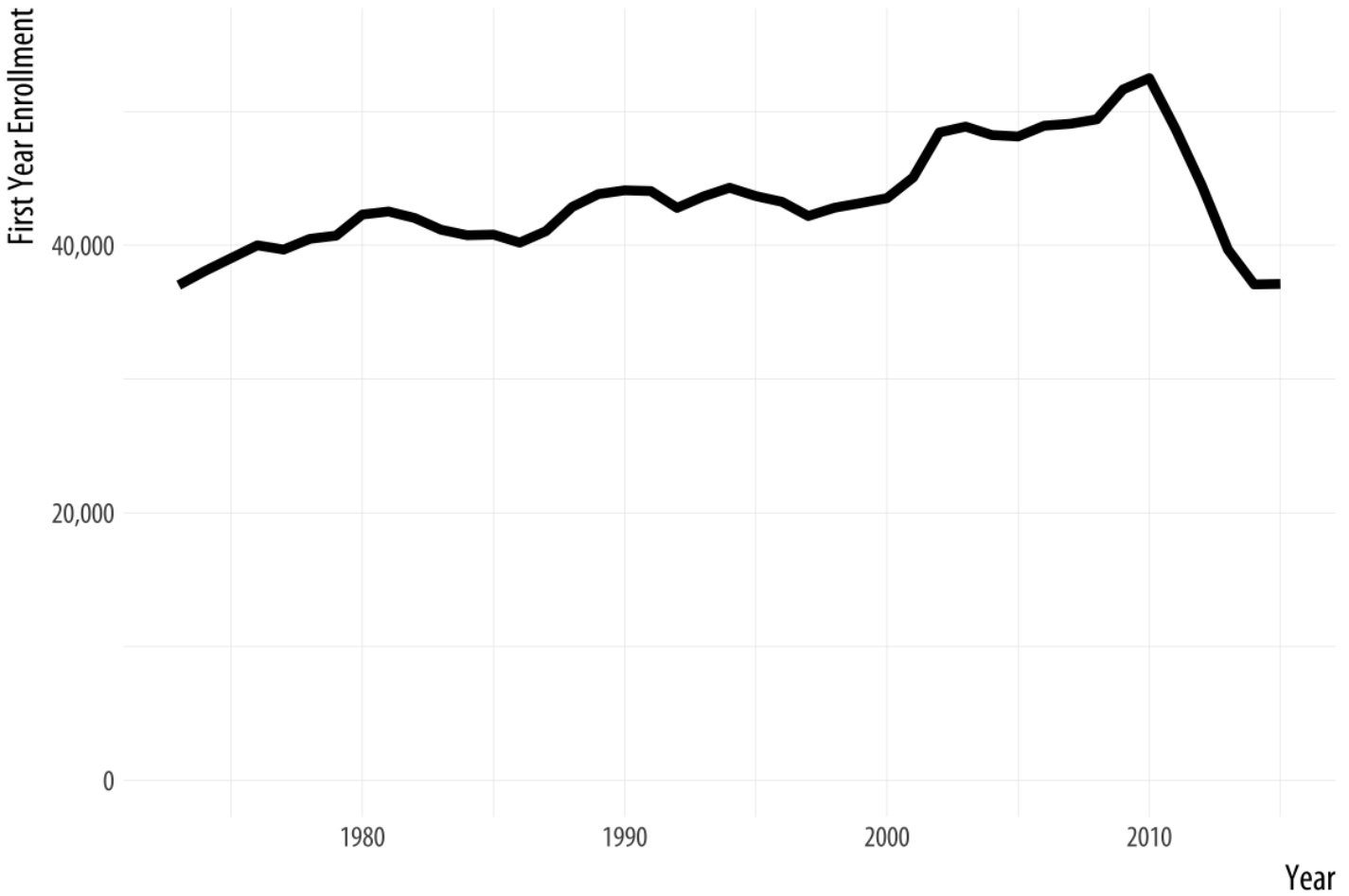
Points


Oceania
Europe
Americas
Asia
Africa
60
70
80
Life Expectancy in years, 2007

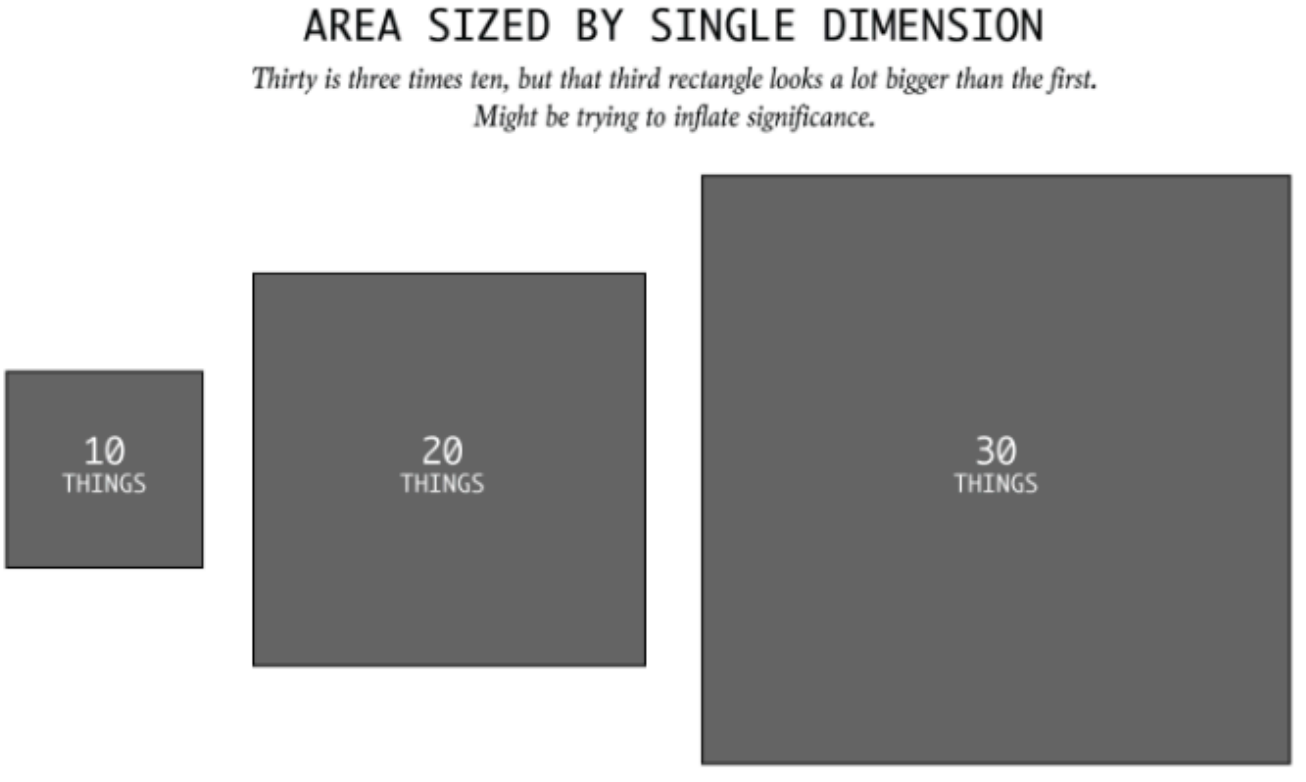
Law school enrollments



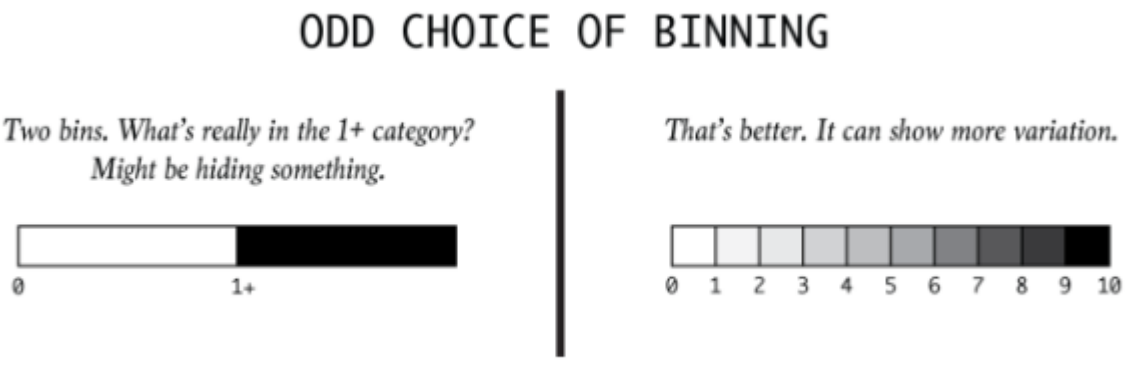
Start at zero



Scaling issues



Poor binning choices



Conclusions

Essentially never

- Use dual axes (unles they are direct transformations, just produce separate plots instead)
- Use 3D unnecessarily

Be wary of

- Truncated axes

Do

- Minimize cognitive load
- Be as clear as possible

Let's wrap Lab 2 and then move onto Lab PS-1 and

Review Lab 2

Jump onto Lab PS1

Lab 3 - if time permits