Time

Week 9

AEM 2850: R for Business Analytics

Cornell Dyson

Spring 2022

Acknowledgements: Andrew Heiss

Announcements

Reminders:

- Mini Project 1 due April 1 (link)

Questions before we get started?

Plan for today

Prologue: Axis issues

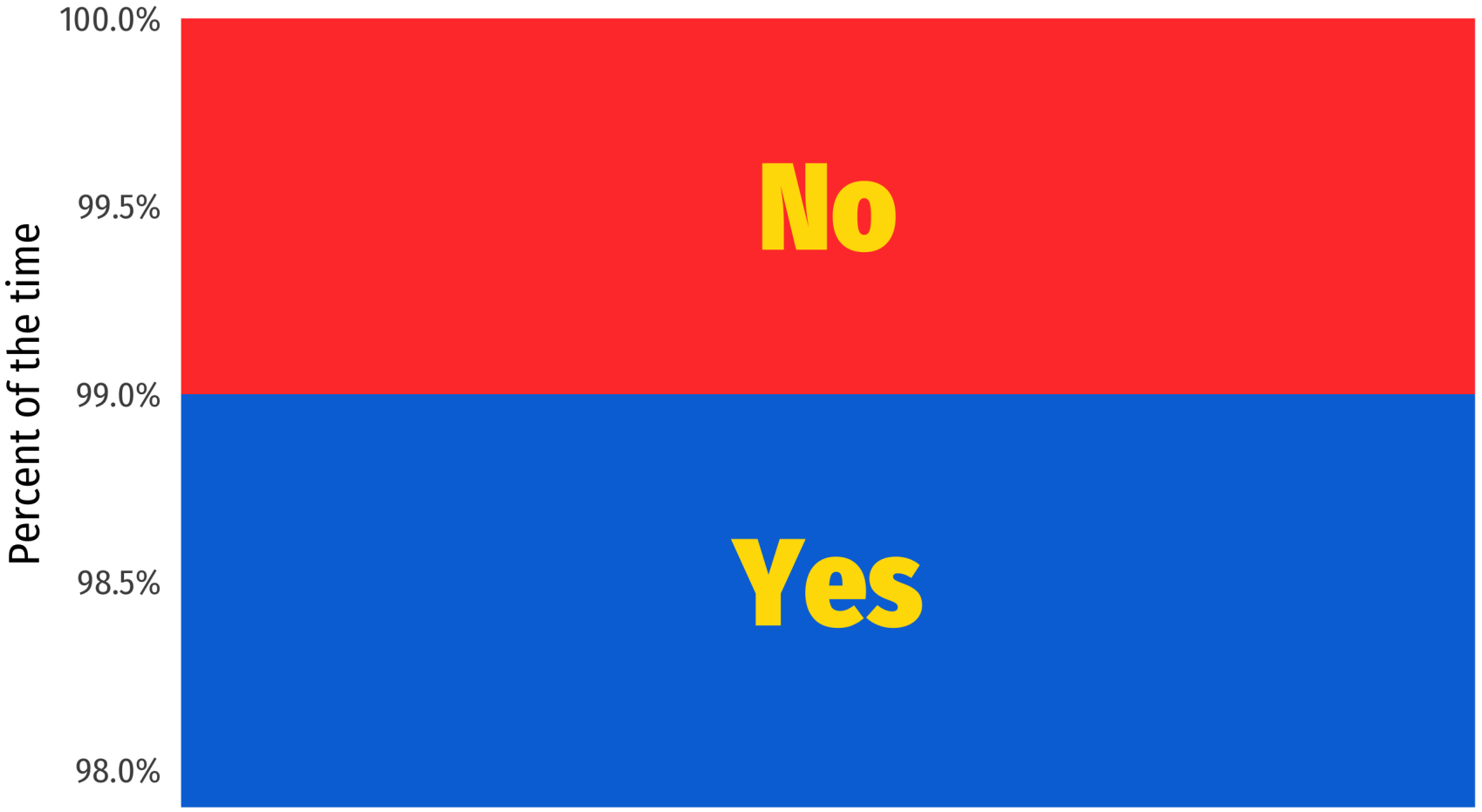
Visualizing time

Starting, ending, and decomposing time

Dates and times in R

Prologue: Axis issues

Is truncating the y-axis misleading?



Don't be too extreme!

It is actually more legal to truncate the y-axis than you might think

When do you think it is okay to truncate?

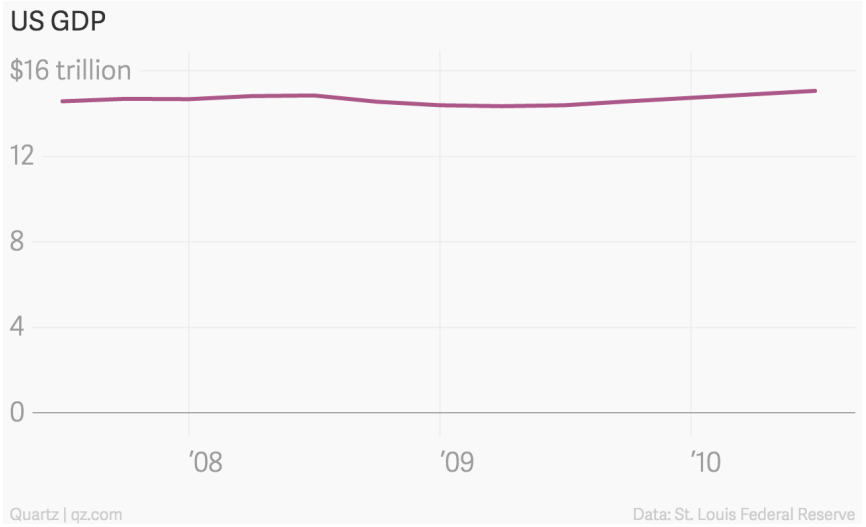
When small movements matter

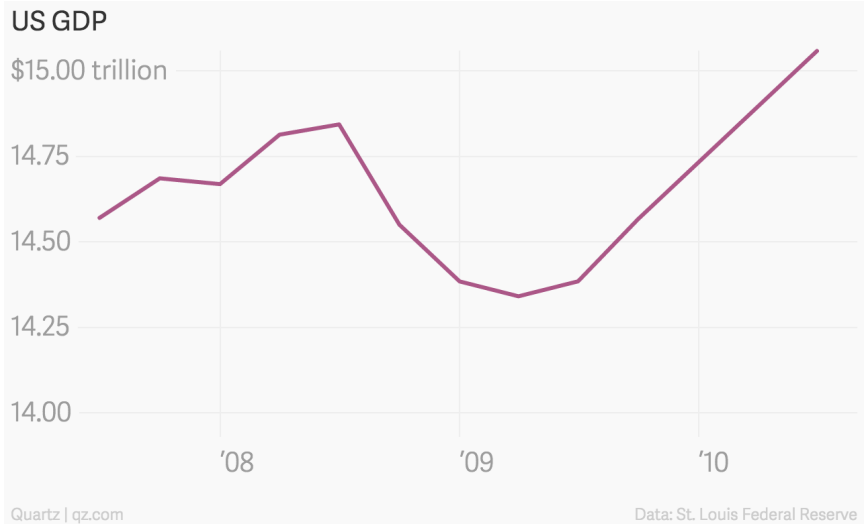
When the scale itself is distorted

When zero values are impossible

When is it okay to truncate?

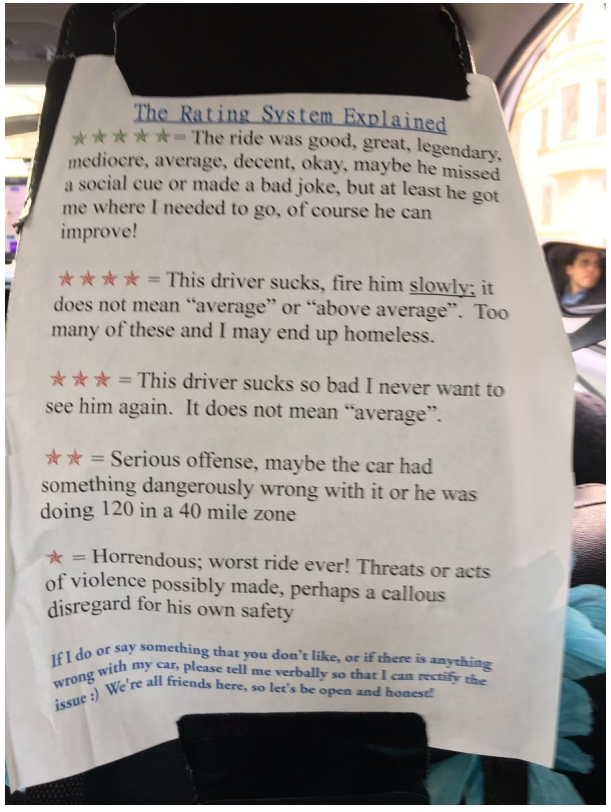
When small movements matter

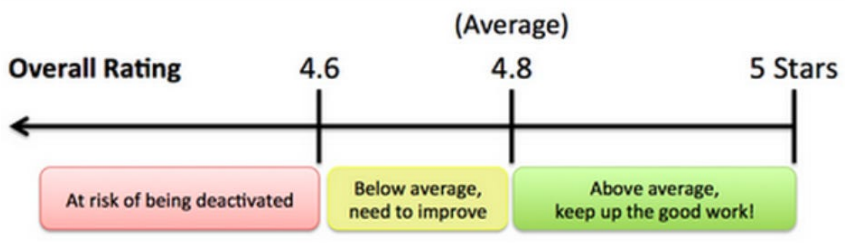




When is it okay to truncate?

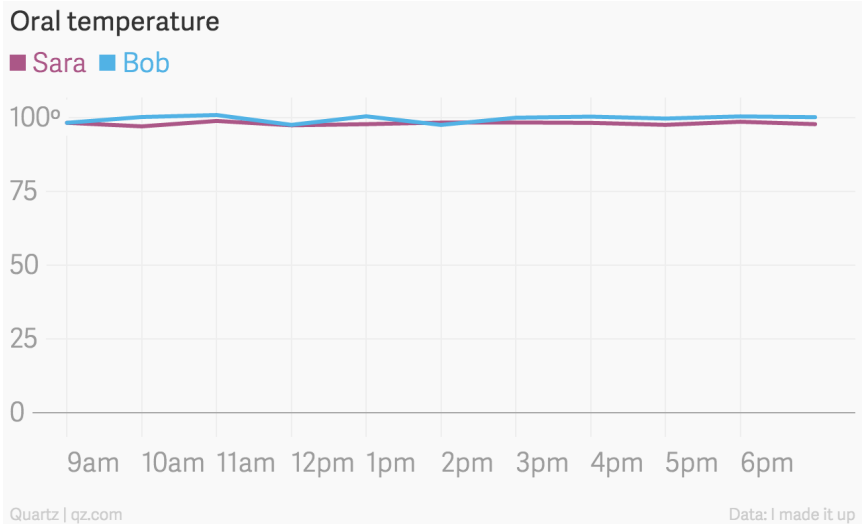
When the scale itself is distorted

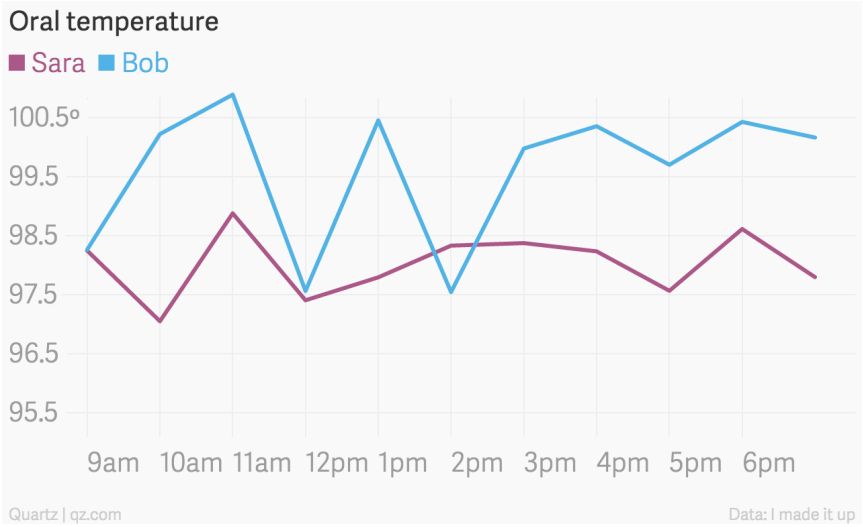




When is it okay to truncate?

When zero values are impossible



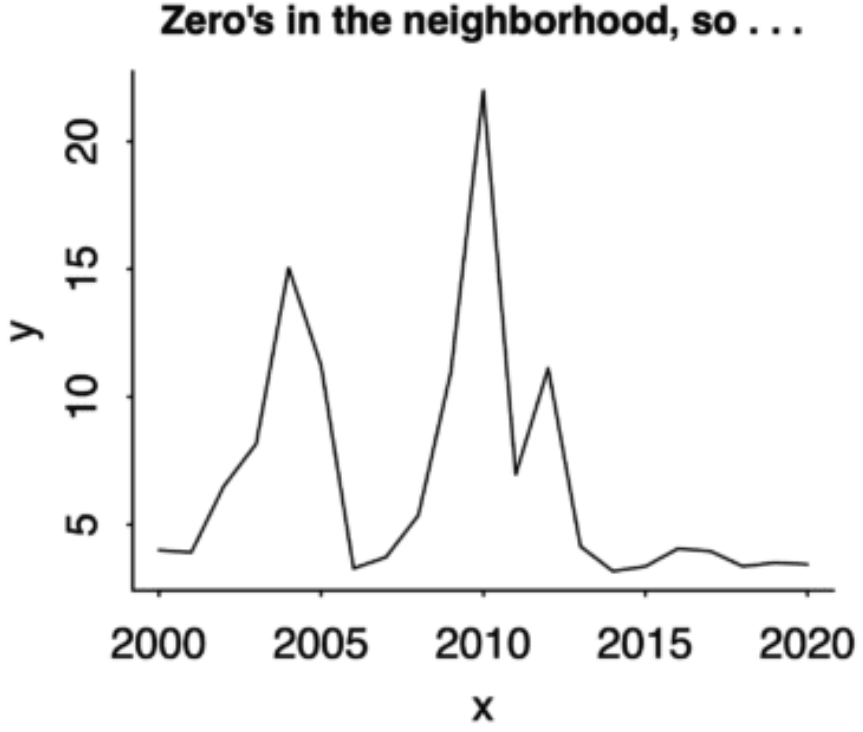


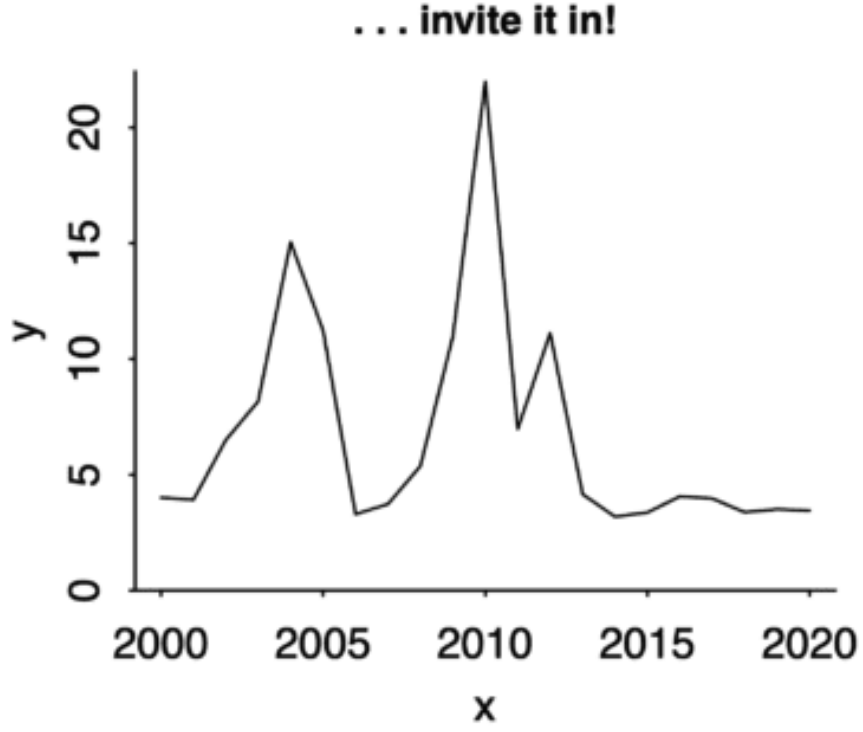
Never on bar charts

Zero is okay too!

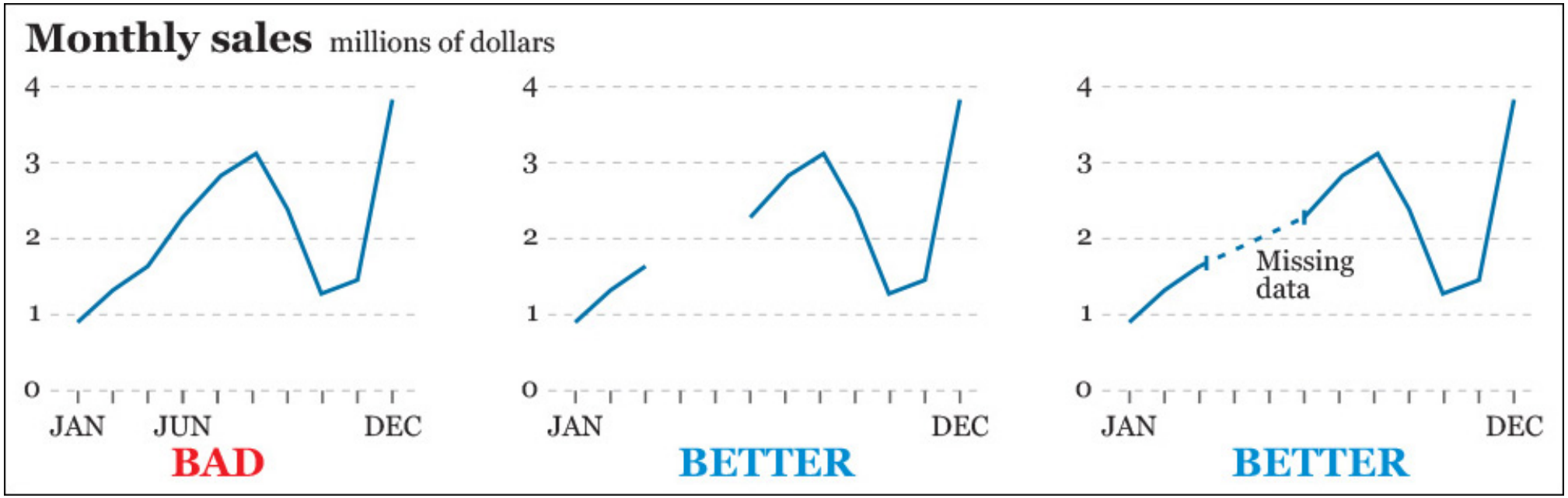
Just because you don't *have to* start at 0 doesn't mean you should *never* start at 0

Andrew Gelman's heuristic: **"If zero is in the neighborhood, invite it in."**

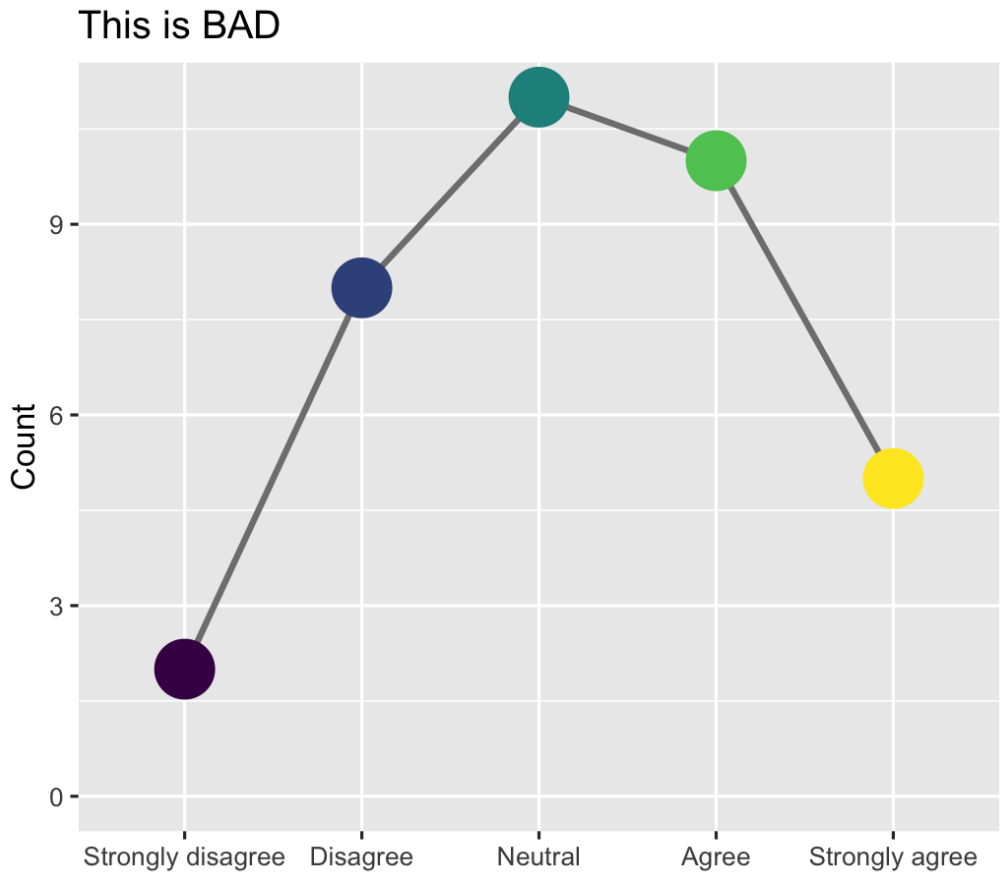


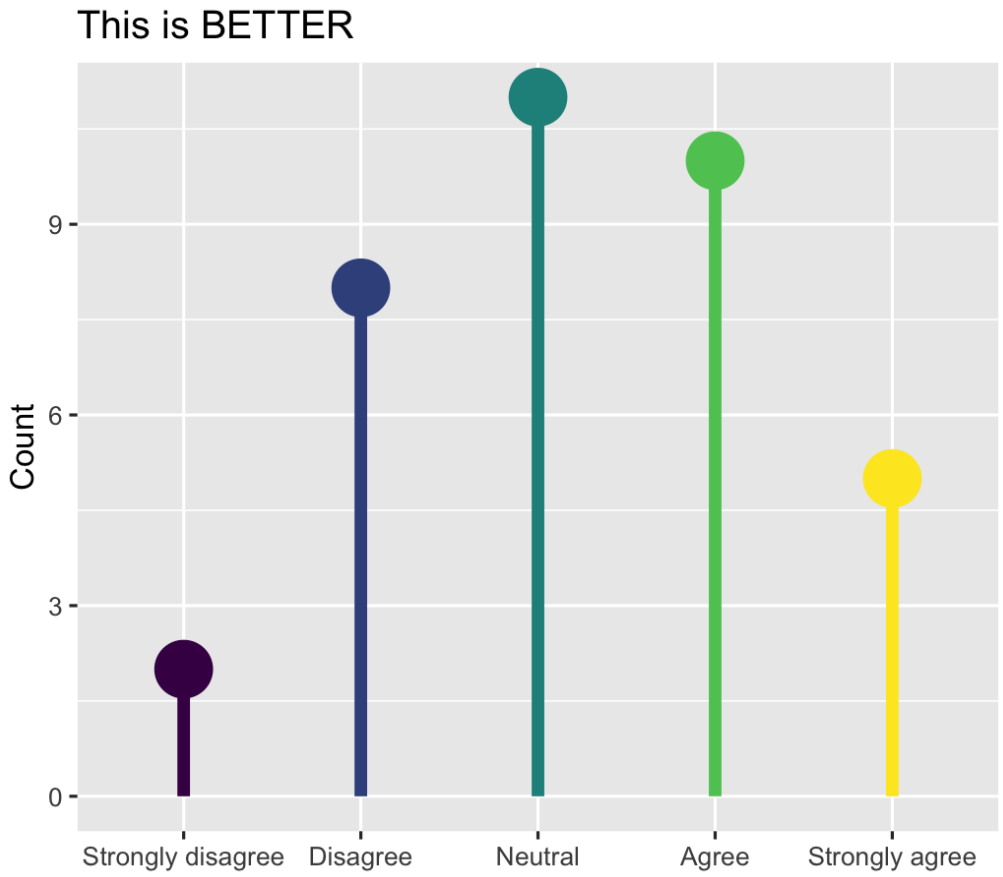


Keep scales consistent and flag missing data



Don’t impute across categories





Visualizing time

Showing changes over time

Time is just a variable that can be mapped to an aesthetic

Can be used as `x`, `y`, `color`, `fill`, `facet`, and even animation

Can use all sorts of `geom`s: lines, columns, points, heatmaps, densities, maps, etc.

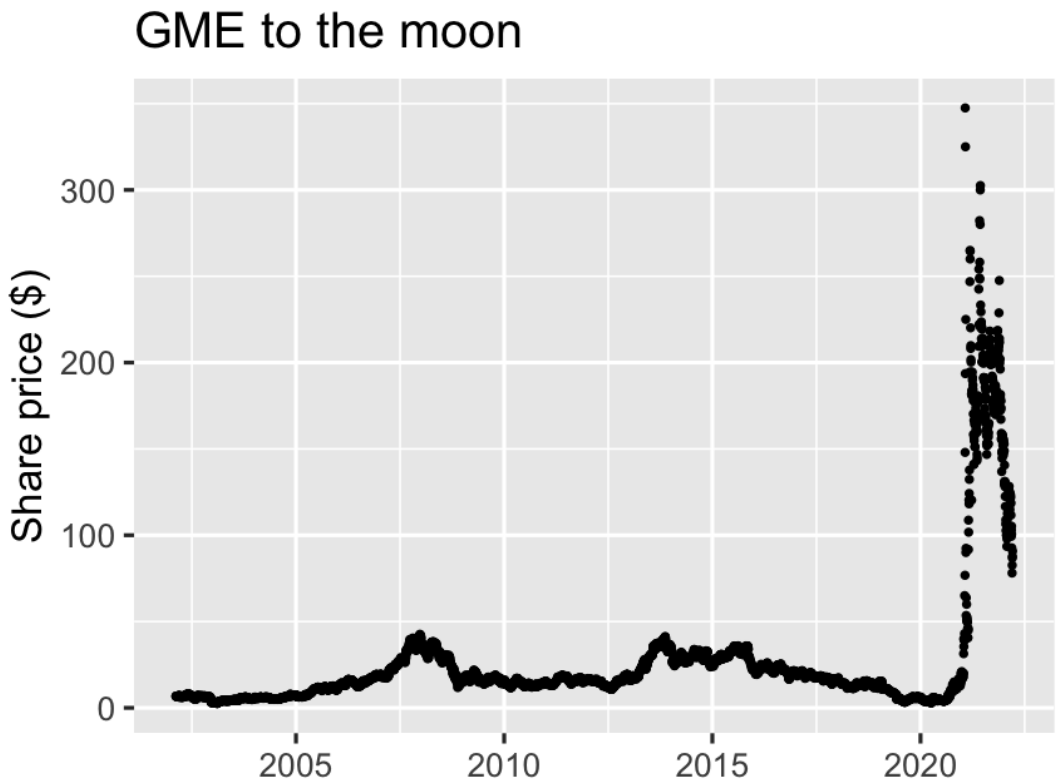
In general, follow reading conventions to show time progression:

→ & ↓

Time on x-axis + `geom_point()`

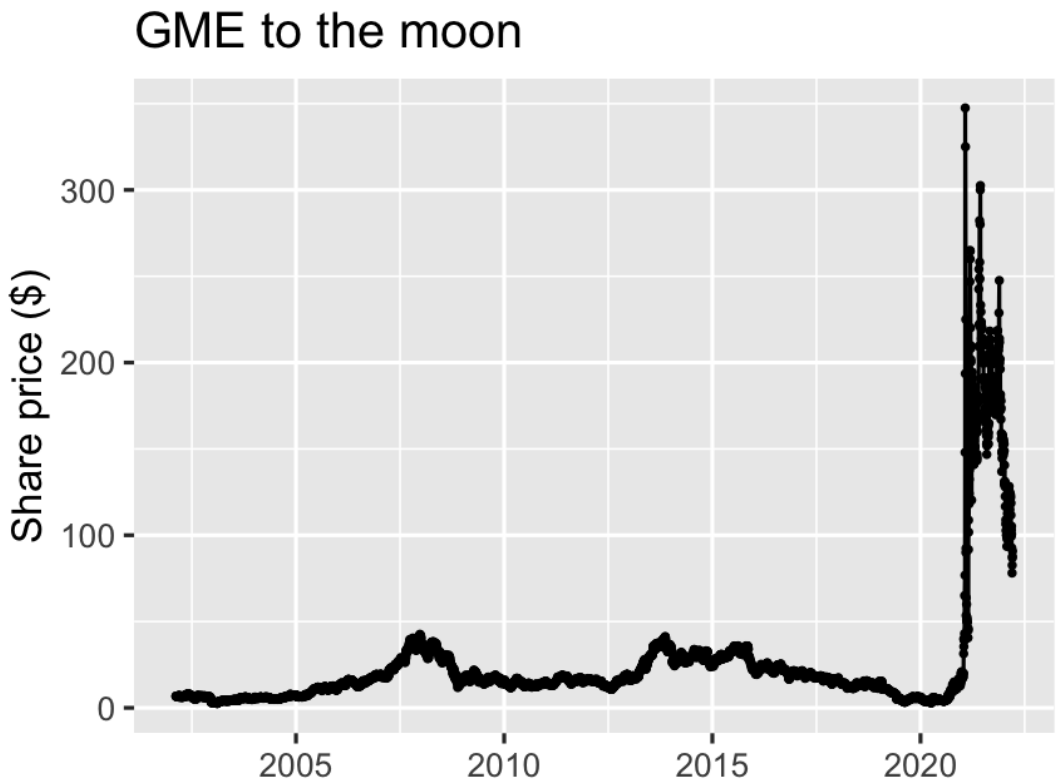
```
gme_prices %>%
  ggplot(aes(x = date, y = adjusted)) +
  geom_point(size = 0.5) +
  labs(x = NULL, y = "Share price ($)",
       title = "GME to the moon")
```

How would you add a line to this?



Time on x-axis + `geom_point()`

```
gme_prices %>%
  ggplot(aes(x = date, y = adjusted)) +
  geom_point(size = 0.5) +
  geom_line() +
  labs(x = NULL, y = "Share price ($)",
       title = "GME to the moon")
```



Time on x-axis: points vs lines

Points emphasize observations

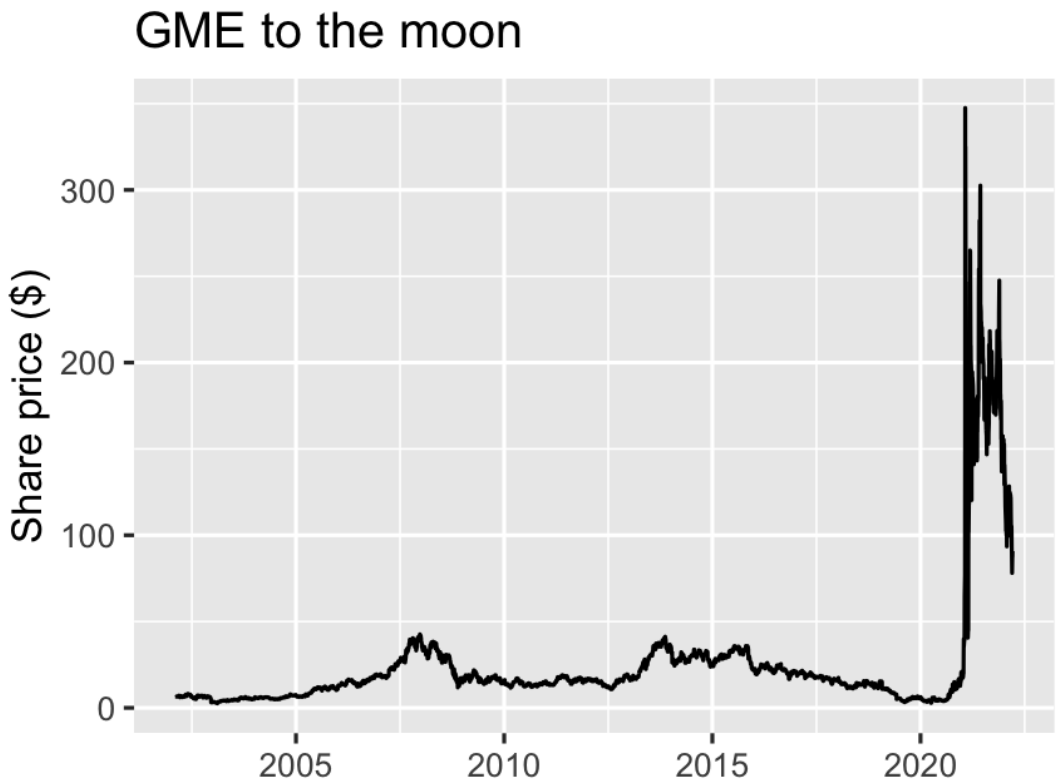
Lines emphasize trends

Using lines for time series is often fine since data are evenly spaced and usually complete

But lines are effectively made-up data, so be careful how you use them!

Time on x-axis + `geom_line/col()`

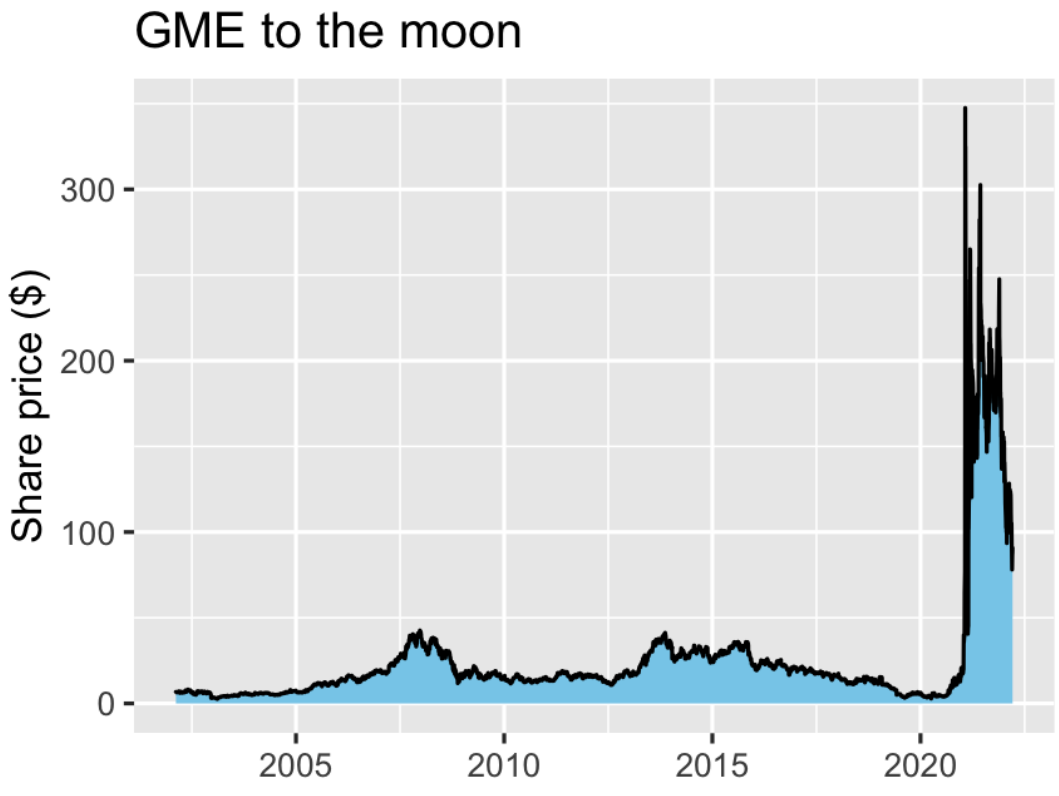
```
gme_prices %>%
  ggplot(aes(x = date, y = adjusted)) +
  geom_line() +
  labs(x = NULL, y = "Share price ($)",
       title = "GME to the moon")
```



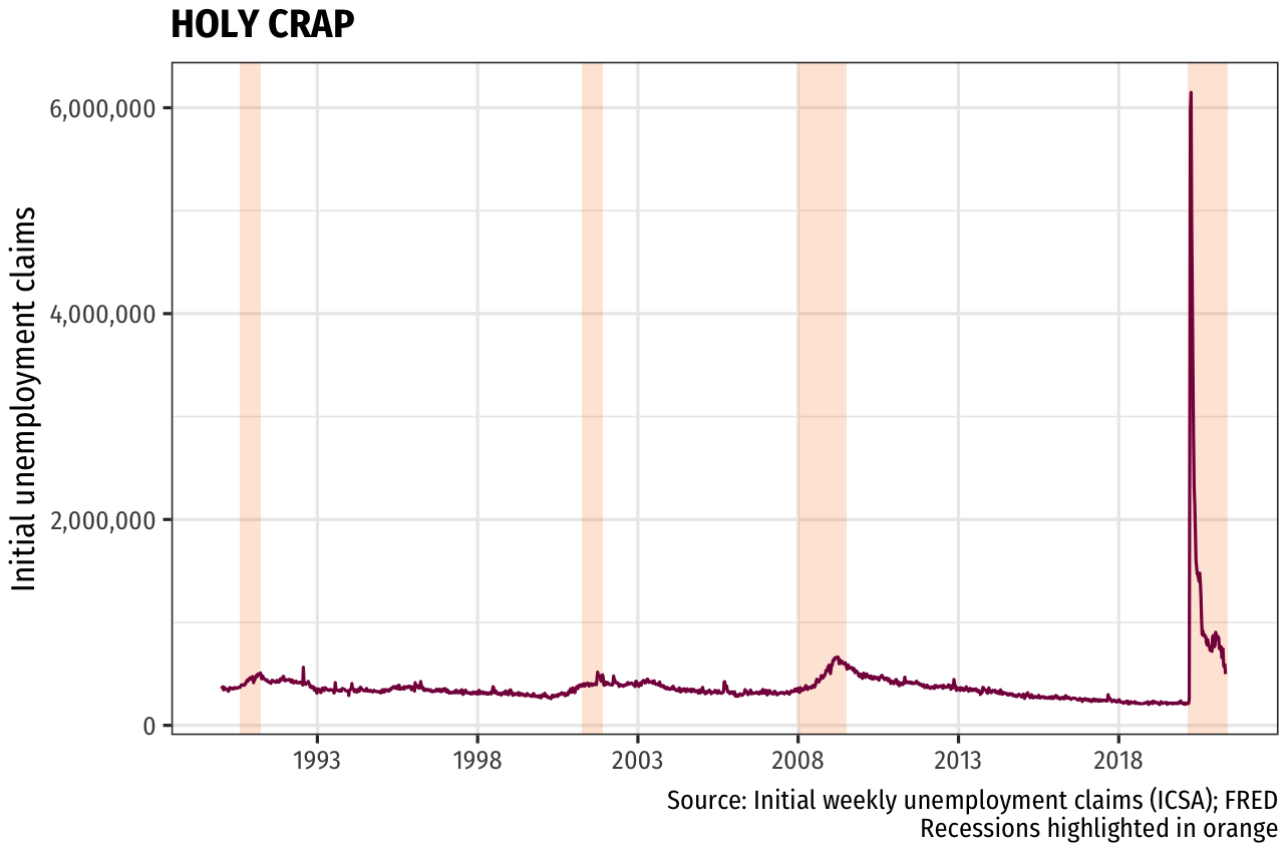
Time on x-axis + `geom_line/col()`

Can also use a fill for the area under a line, **as long as the y-axis starts at zero**

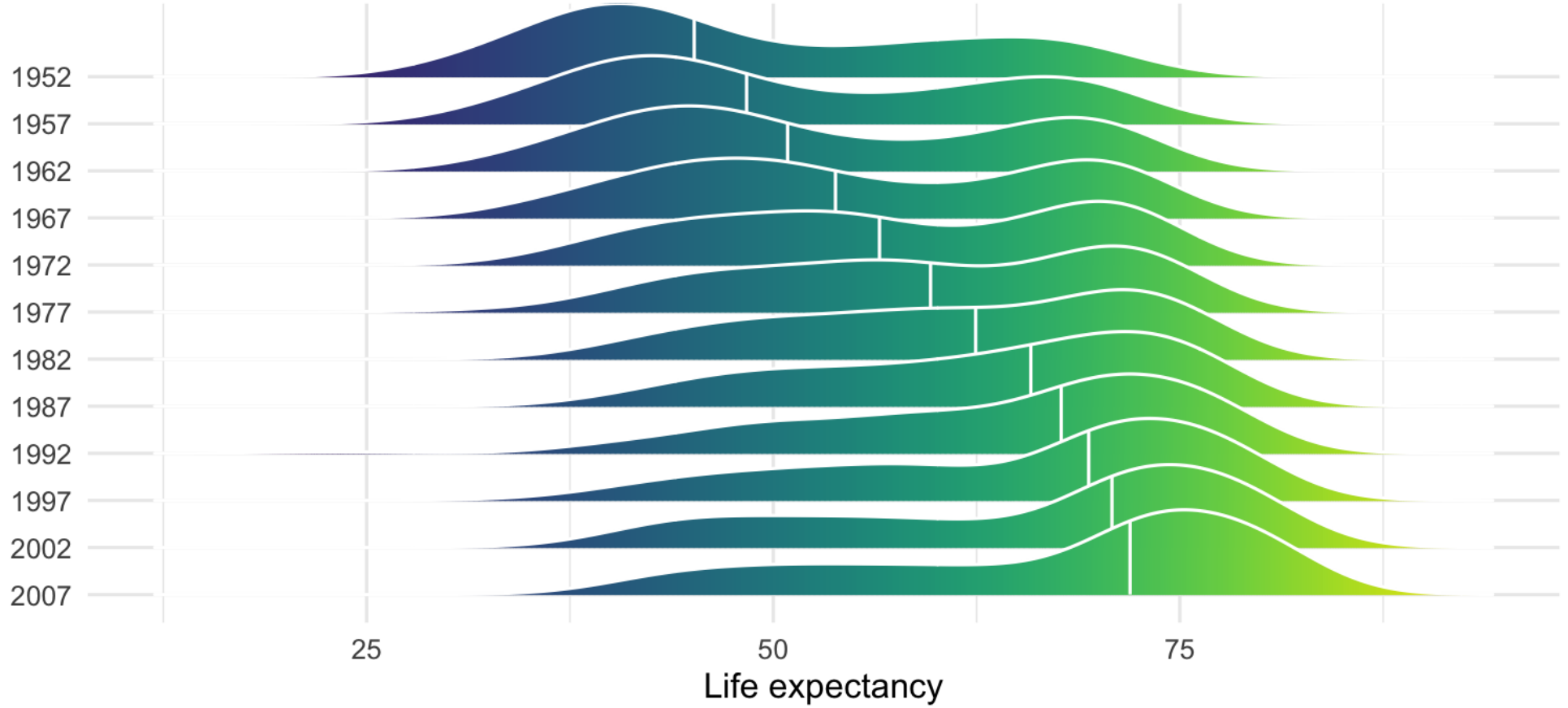
```
gme_prices %>%
  ggplot(aes(x = date, y = adjusted)) +
  geom_area(fill = "skyblue", color = "black") +
  labs(x = NULL, y = "Share price ($)",
       title = "GME to the moon")
```



Line plots don't have to be boring



Time on y-axis + `geom_density()`



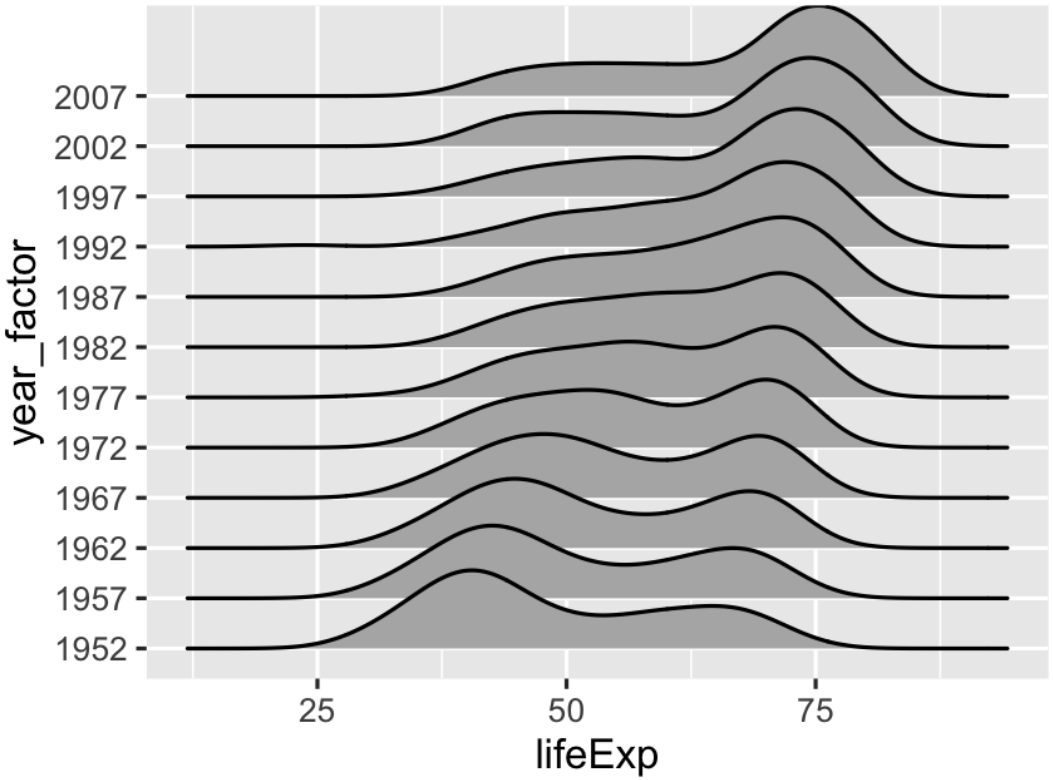
How would you make this plot?

Let's start simple

```
library(ggridges) # geom_density_ridges() package

gapminder %>%
    mutate(year_factor = factor(year)) %>%
    ggplot(
      aes(x = lifeExp,
          y = year_factor) # map time to y
      ) +
    geom_density_ridges() # add geom
```

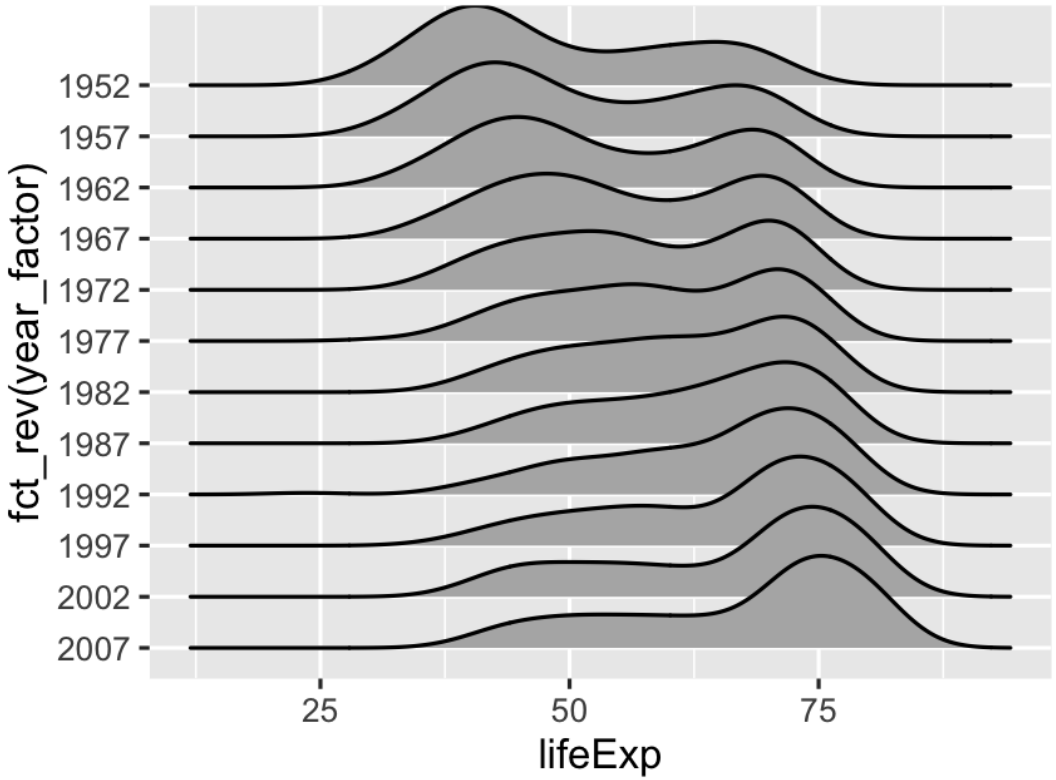
How could we modify this to follow ↓ convention for time?



Follow ↓ convention for time

```
gapminder %>%
    mutate(year_factor = factor(year)) %>%
    ggplot(
      aes(x = lifeExp,
          y = fct_rev(year_factor)) # reverse year
      ) +
    geom_density_ridges()
```

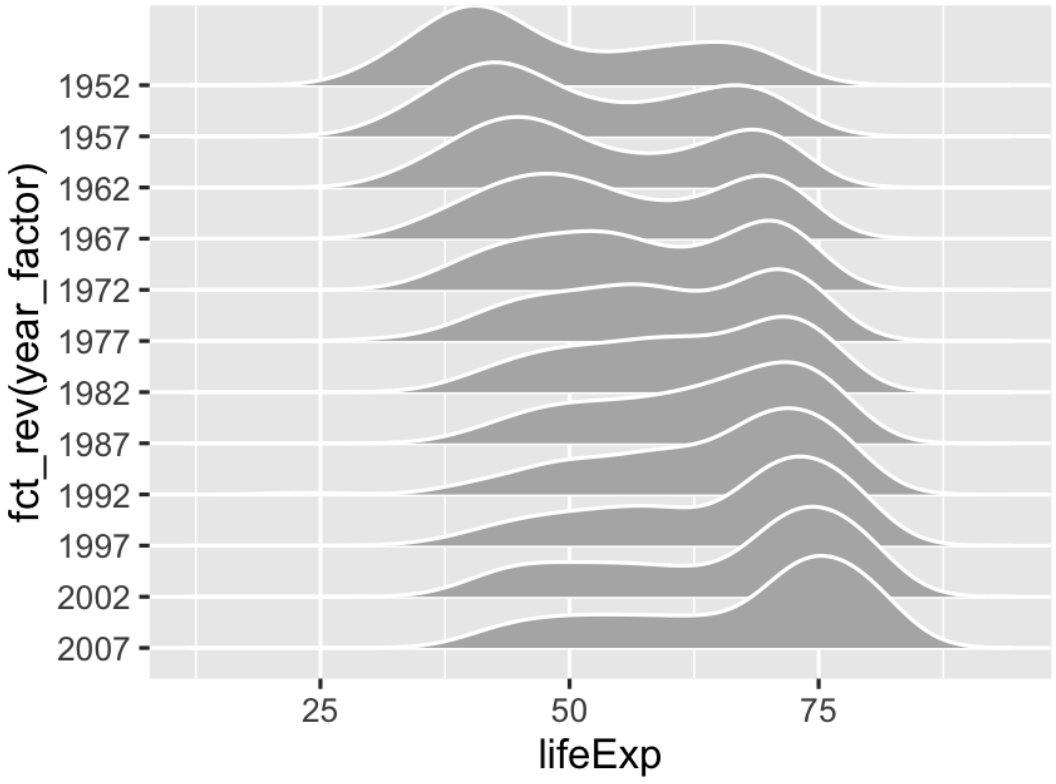
How could we change the black lines to white?



Add color to help visually separate densities

```
gapminder %>%
    mutate(year_factor = factor(year)) %>%
    ggplot(
      aes(x = lifeExp,
          y = fct_rev(year_factor))
      ) +
    geom_density_ridges(
      color = "white"  # separate densities
      )
```

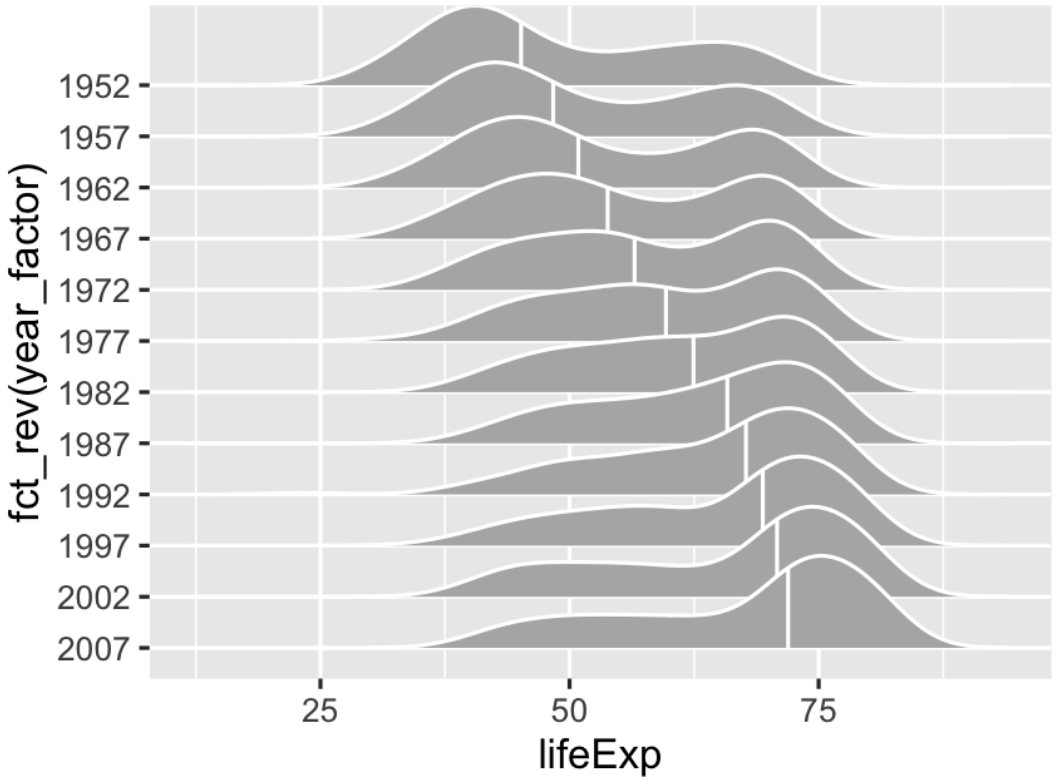
How could we add a line at the median of each density?



Add a line at the median of each density

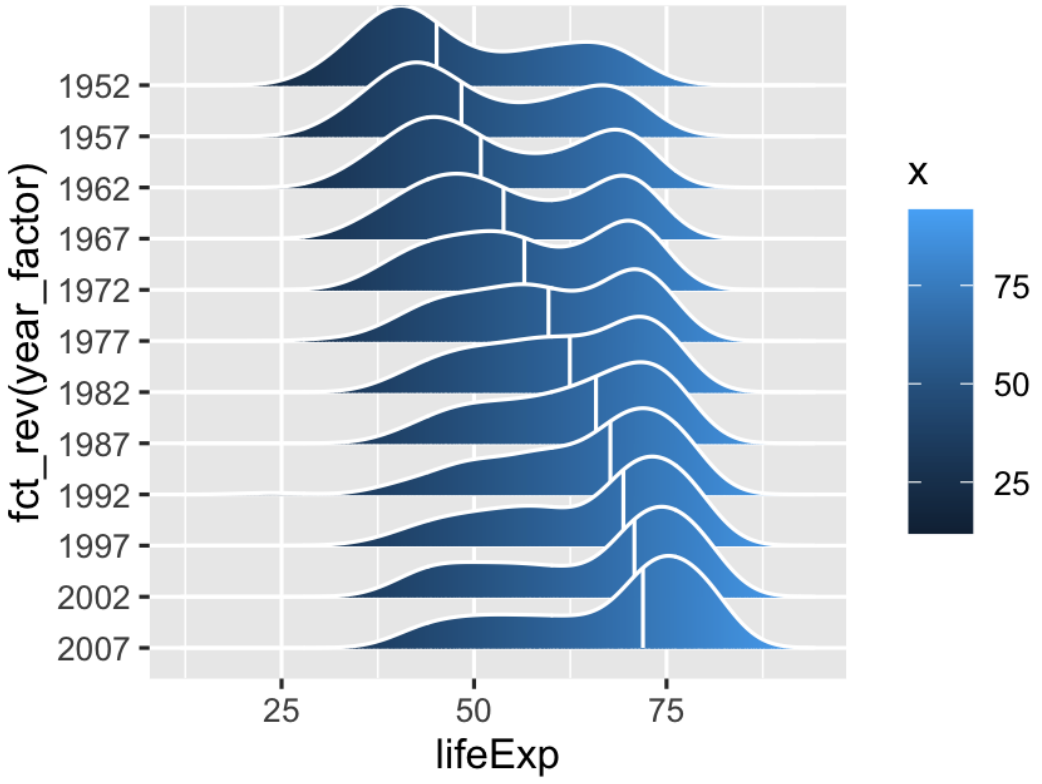
```
gapminder %>%
    mutate(year_factor = factor(year)) %>%
    ggplot(
      aes(x = lifeExp,
          y = fct_rev(year_factor))
      ) +
    geom_density_ridges(
      color = "white",
      quantile_lines = TRUE, # add quantile lines
      quantiles = 2          # median
      )
```

How could we add a color gradient to the densities?



Fill the densities along the x axis

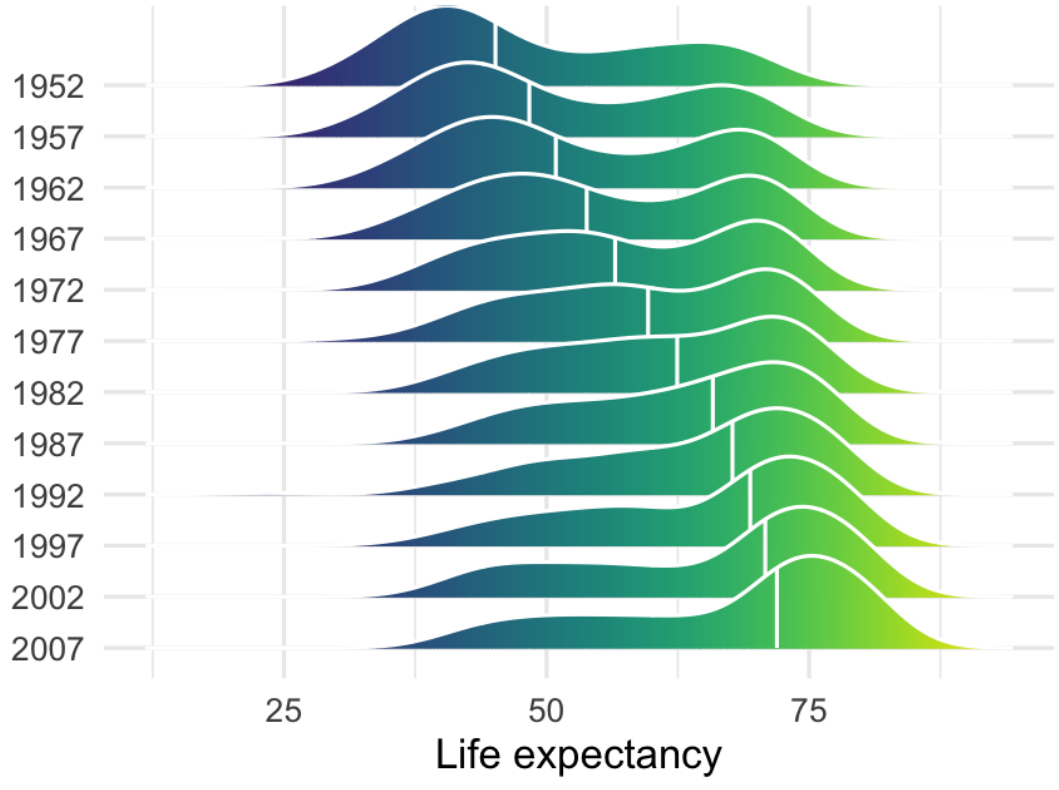
```
gapminder %>%
    mutate(year_factor = factor(year)) %>%
    ggplot(
      aes(x = lifeExp,
          y = fct_rev(year_factor),
          fill = after_stat(x)) # fill by x value
      ) +
    geom_density_ridges_gradient(
      # this is a special geom to fill along x
      color = "white",
      quantile_lines = TRUE,
      quantiles = 2
      )
```



Use **after_stat** to shade the *densities*, not the data itself

Finish by cleaning the plot up

```
gapminder %>%
    mutate(year_factor = factor(year)) %>%
    ggplot(
      aes(x = lifeExp,
          y = fct_rev(year_factor),
          fill = after_stat(x))
      ) +
    geom_density_ridges_gradient(
      color = "white",
      quantile_lines = TRUE,
      quantiles = 2
      ) +
    guides(fill = "none") +          # omit legend
    scale_fill_viridis() +           # nicer shading
    labs(x = "Life expectancy",      # modify label
         y = NULL, fill = NULL) +    # omit labels
    theme_minimal()                  # cleaner theme
```

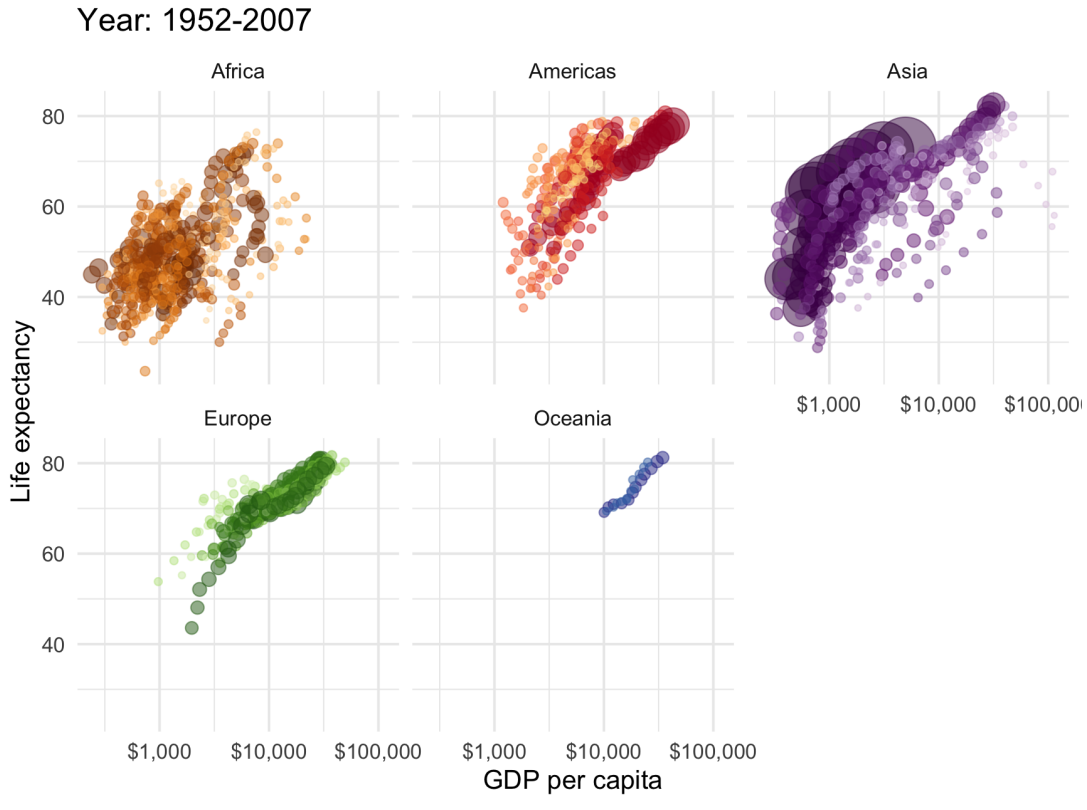


Time in animation + `geom_point()`

How can we make this animation in R?

First, how would we make a static plot of the data?

```
gapminder %>%
  ggplot(aes(gdpPercap, lifeExp,
             size = pop, color = country)) +
  geom_point(alpha = 0.5, show.legend = FALSE) +
  facet_wrap(~continent) +
  scale_color_manual(
    values = gapminder::country_colors
    ) +
  scale_size_continuous(range = c(1, 15)) +
  scale_x_log10(labels = scales::dollar) +
  theme_minimal(base_size = 14) +
  labs(x = "GDP per capita",
       y = "Life expectancy",
       title = "Year: 1952-2007")
```

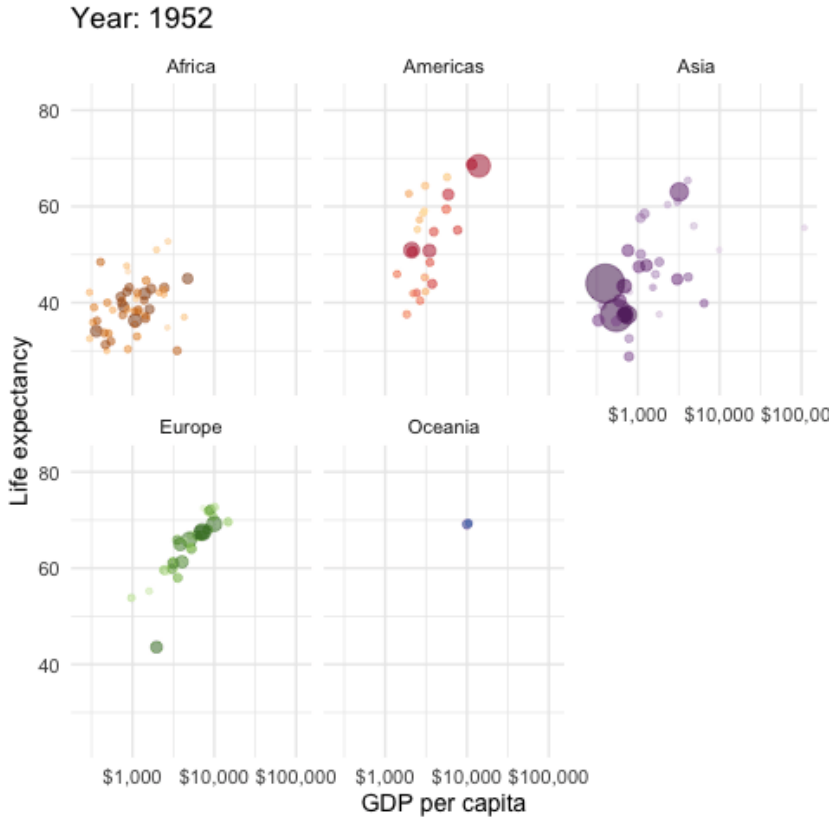


Converting a static plot to an animation

Second, let's use `gganimate` to visualize changes over time

```
library(gganimate) # plot animation package

gapminder %>%
  ggplot(aes(gdpPercap, lifeExp,
             size = pop, color = country)) +
  geom_point(alpha = 0.5, show.legend = FALSE) +
  facet_wrap(~continent) +
  scale_color_manual(
    values = gapminder::country_colors
    ) +
  scale_size_continuous(range = c(1, 15)) +
  scale_x_log10(labels = scales::dollar) +
  theme_minimal(base_size = 14) +
  labs(x = "GDP per capita",
       y = "Life expectancy",
       title = "Year: {frame_time}") +
  transition_time(year) +
  ease_aes('linear') # default progression
```



Saving gifs

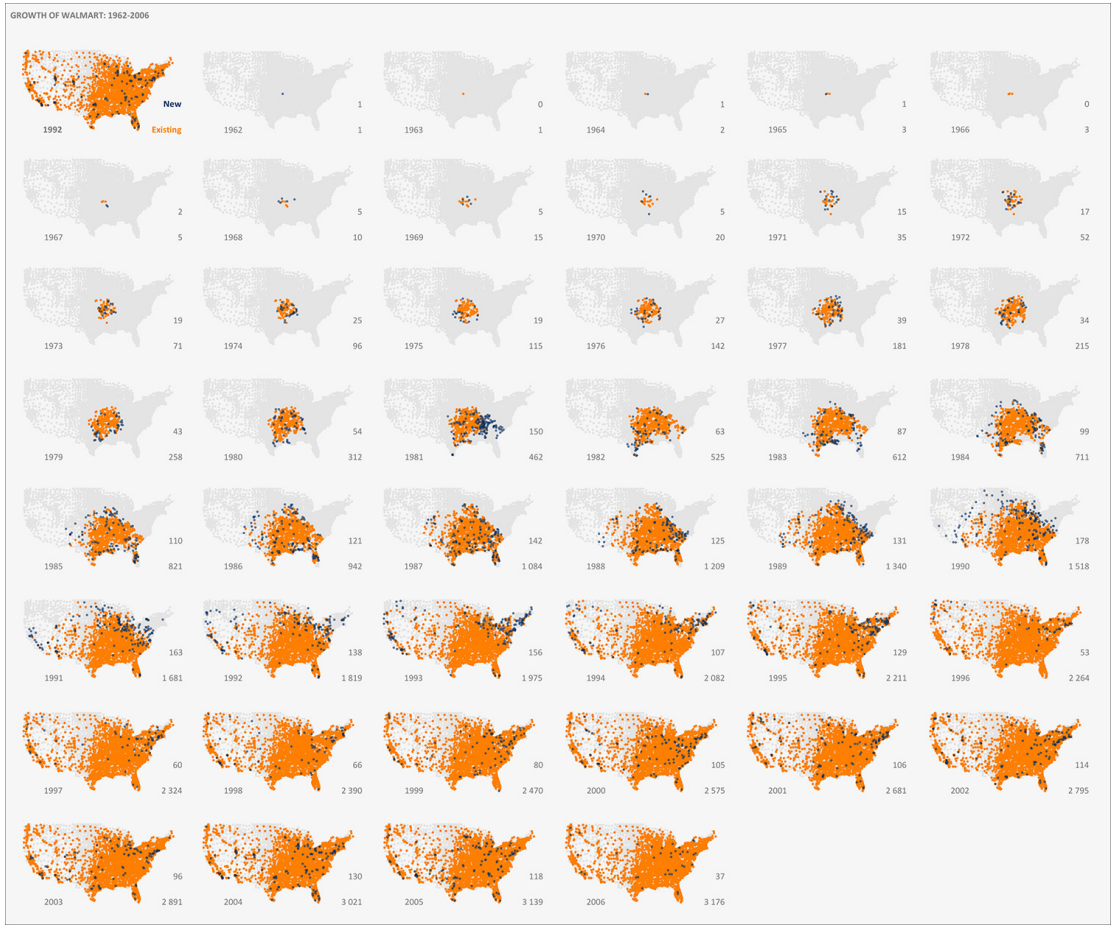
Third, use `anim_save()` to write a `.gif` that you can text to your mom

```
library(gganimate) # plot animation package

my_gapminder_animation <- gapminder %>%
  ggplot(aes(gdpPercap, lifeExp, size = pop, color = country)) +
  geom_point(alpha = 0.5, show.legend = FALSE) +
  scale_color_manual(values = gapminder::country_colors) +
  scale_size_continuous(range = c(1, 15)) +
  scale_x_log10(labels = scales::dollar) +
  facet_wrap(~continent) +
  theme_minimal(base_size = 14) +
  labs(x = "GDP per capita",
       y = "Life expectancy",
       title = "Year: {frame_time}") +
  transition_time(year) +
  ease_aes('linear')

anim_save("content/slides/img/09/my-gapminder-animation.gif",
          my_gapminder_animation)
```

Time in maps



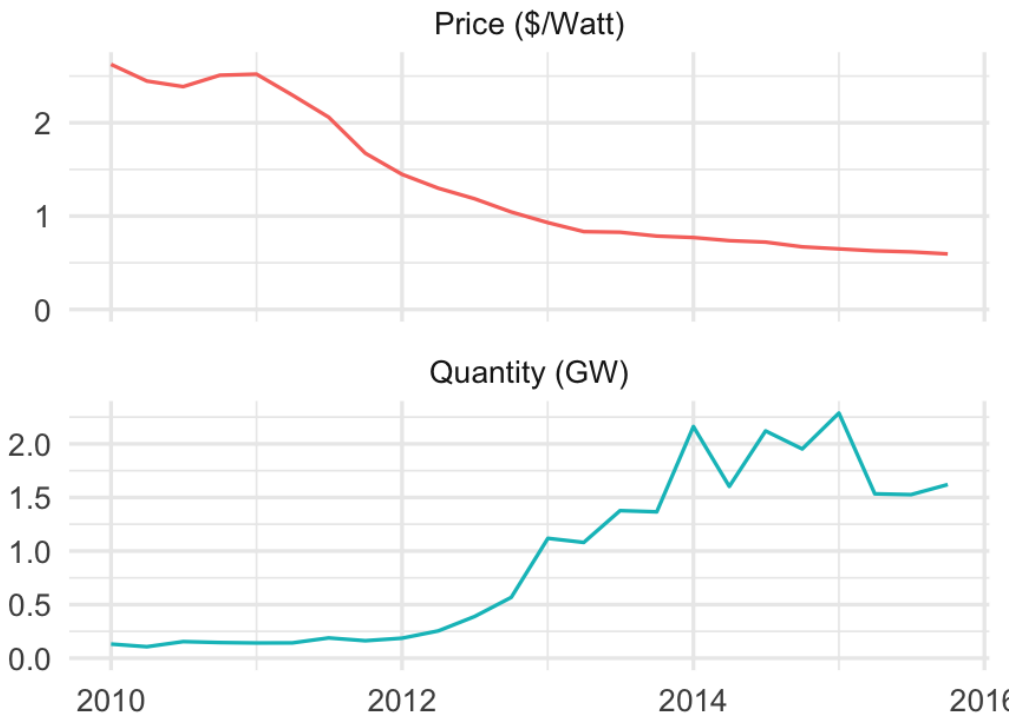
Map of the spread of Walmart by Jorge Camões

Connected scatter plots

Sometimes connected scatter plots of time series data make sense

What is a connected scatter plot?

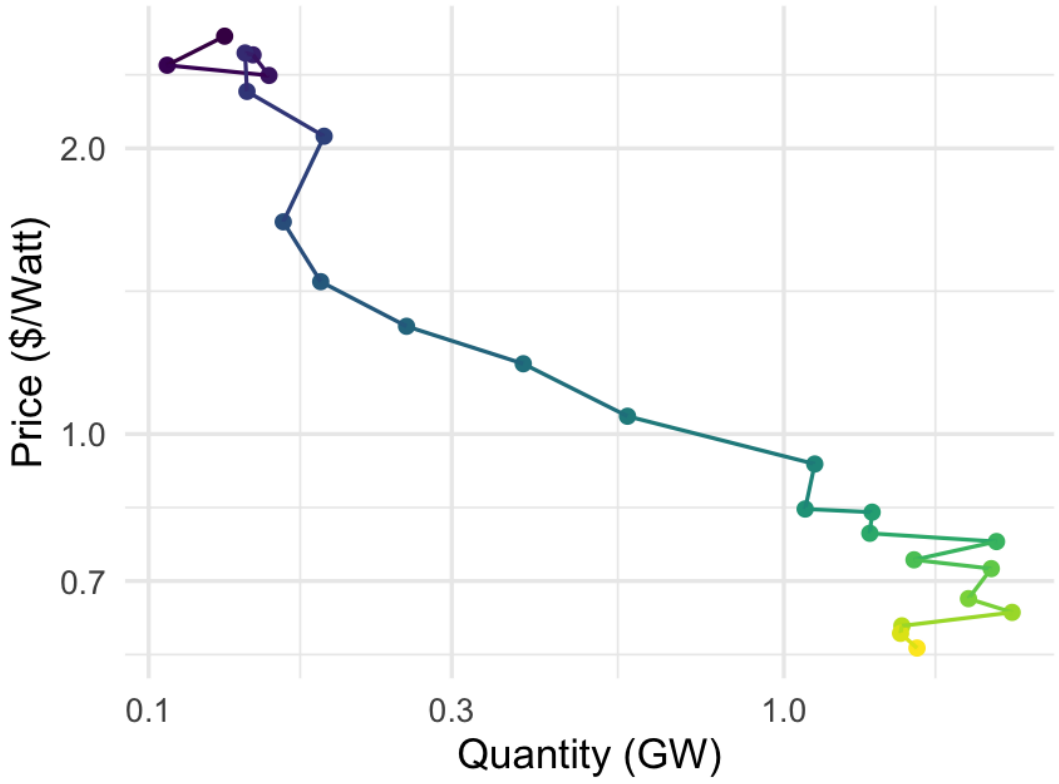
How would you make one using these data?



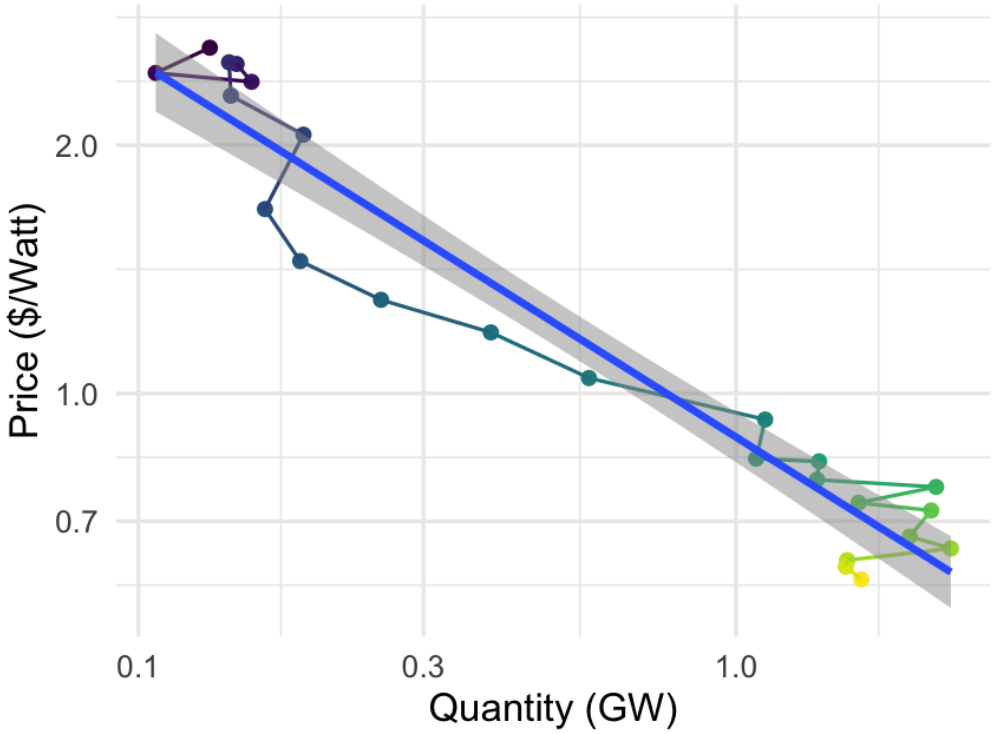
Connected scatter plots

```
solar_data %>%
  ggplot(aes(x = `Quantity (GW)`,  # not time
             y = `Price ($/Watt)`, # not time
             color = date)) +      # time!
  geom_point() +
  geom_path() + # connect by time, not x
  scale_x_log10() +
  scale_y_log10() +
  scale_color_viridis() +
  guides(color = "none") +
  theme_minimal()
```

Note the log axes



Is this a good use case?



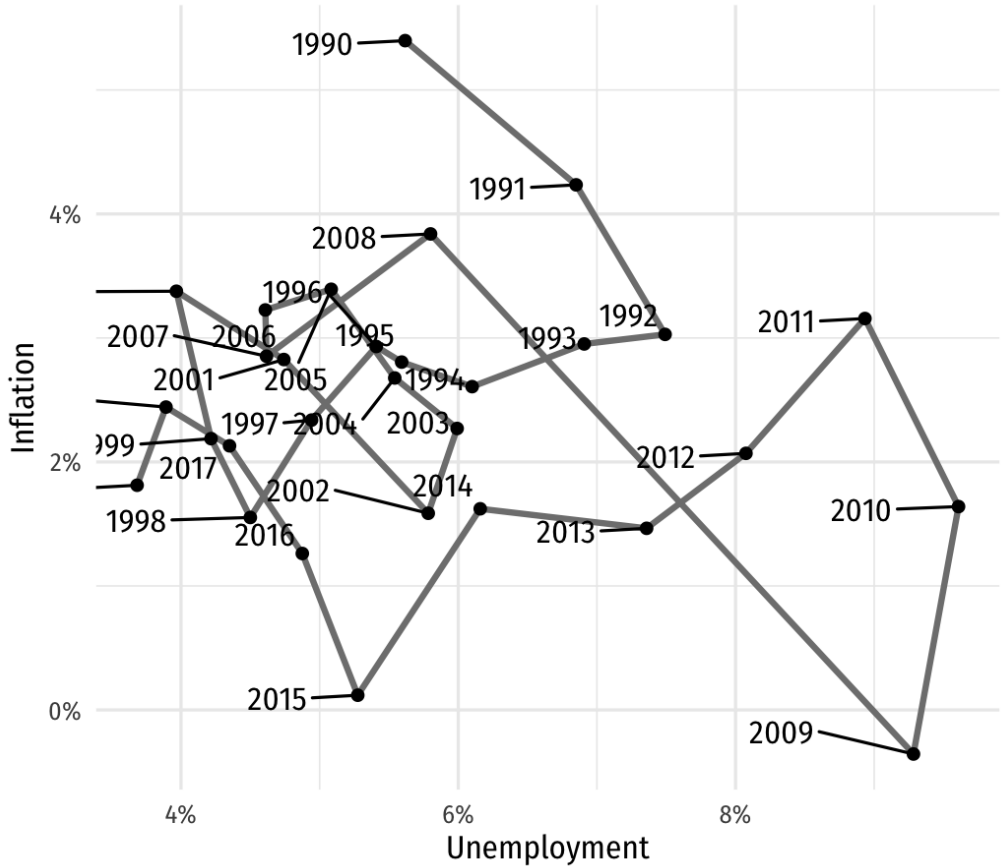
Looks a lot like a demand curve!

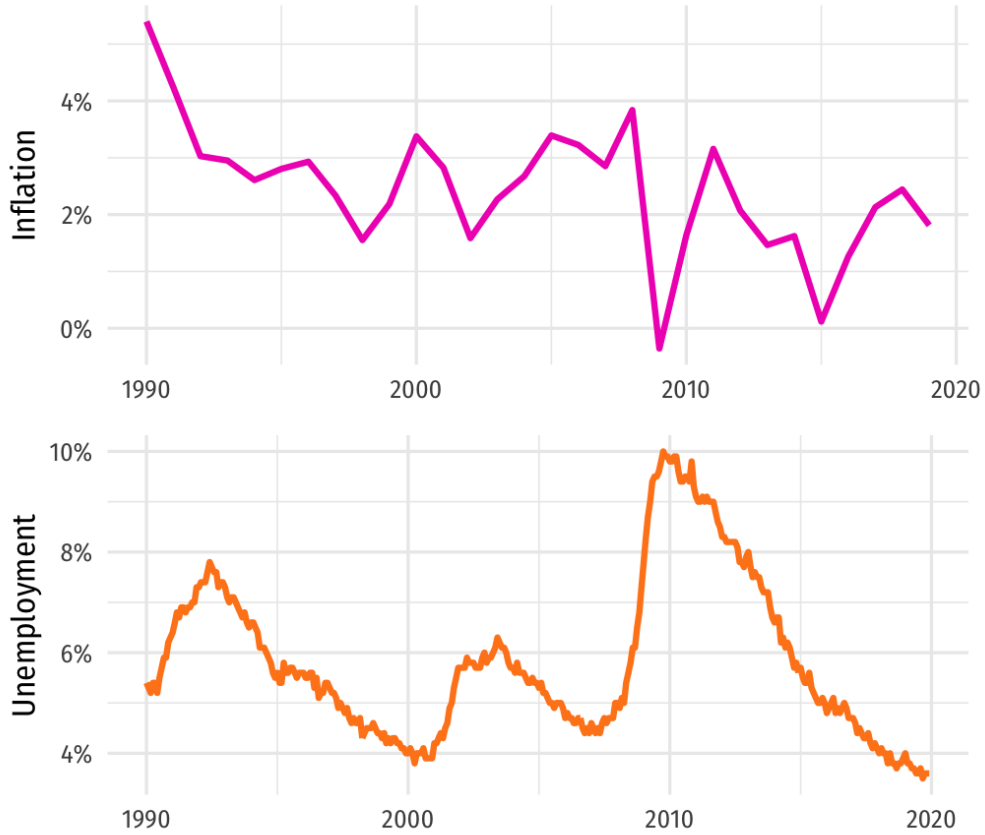
$$log(Q) = \alpha + \beta log(P) + \varepsilon$$

We could use OLS to estimate β

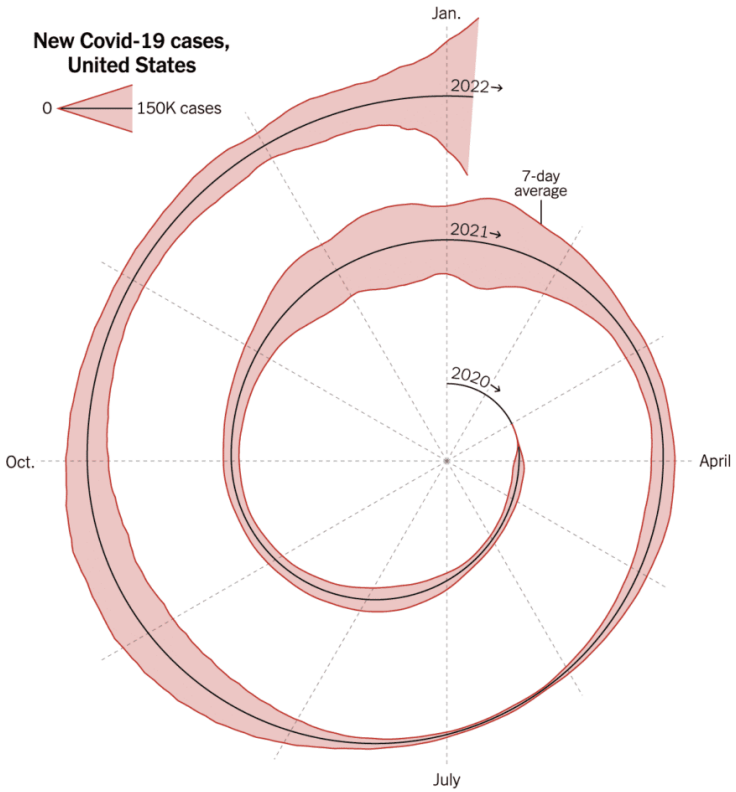
How might we interpret β?

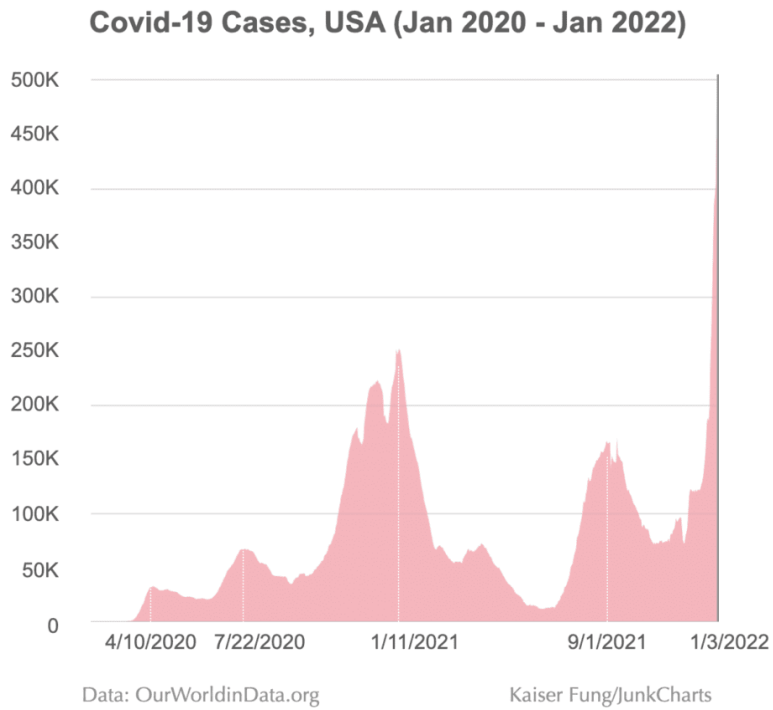
Often it's better to use multiple plots





Don't go wild with time mapping!





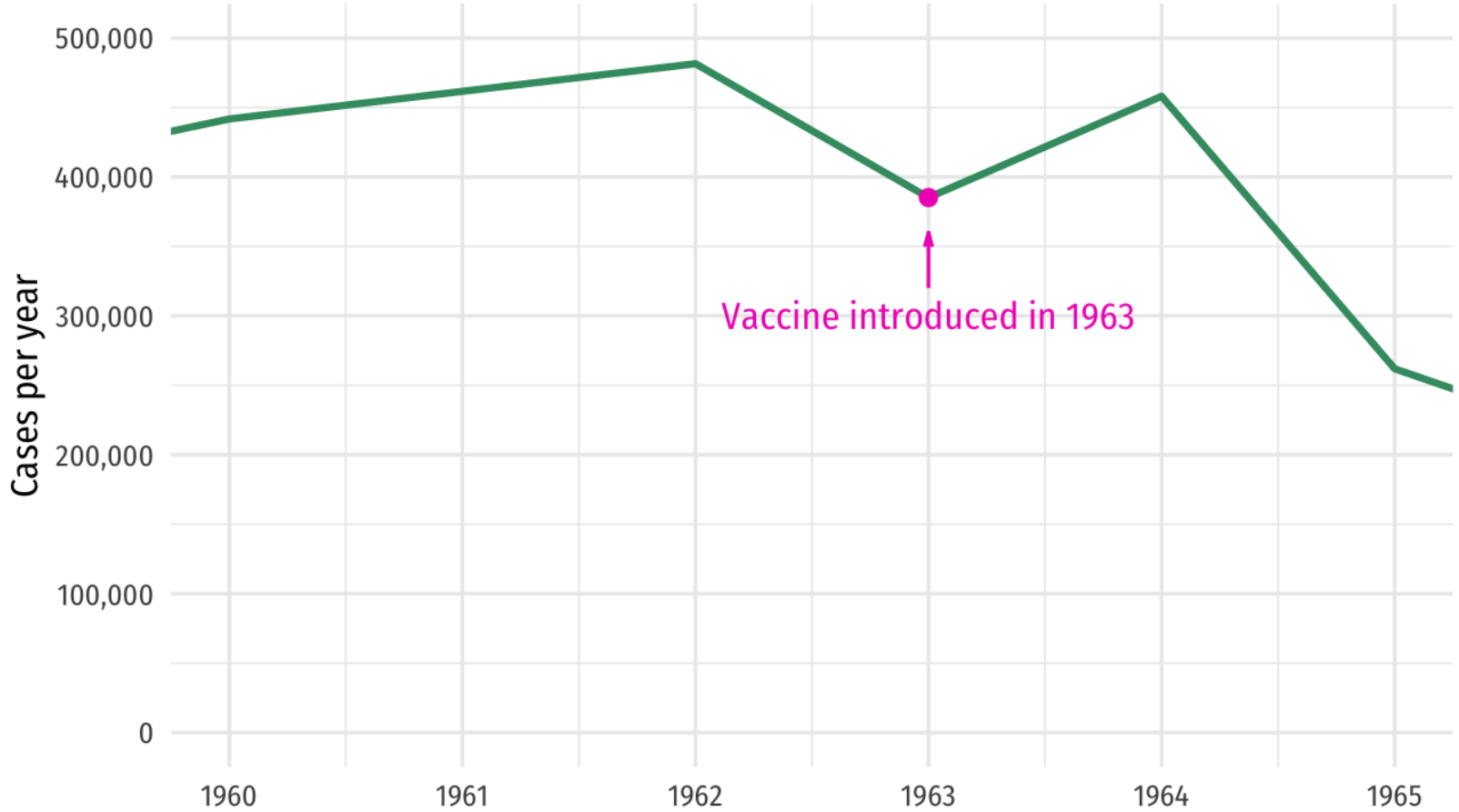
Starting, ending, and decomposing time

You have to start (and end) somewhere

You always have to choose start and end points

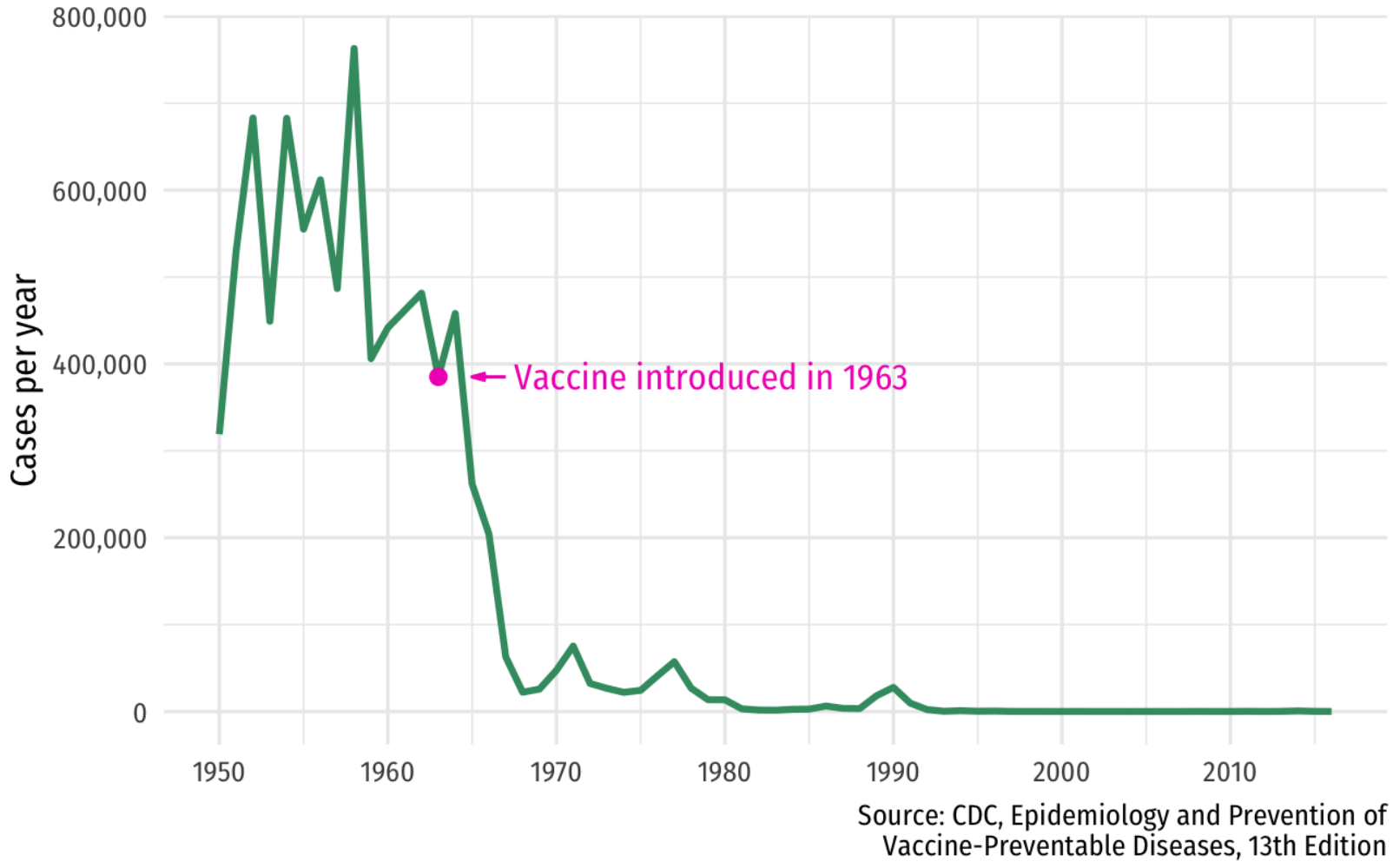
Start and end at reasonable times that help maintain the context of your story

Measles vaccine was pretty effective



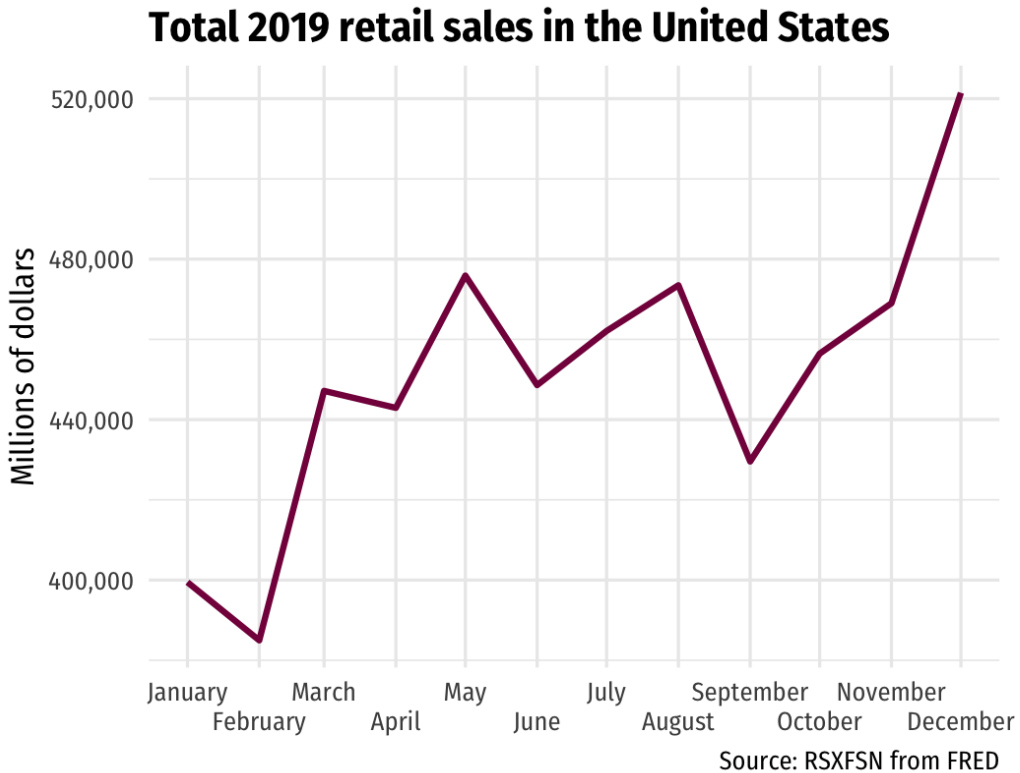
Source: CDC, Epidemiology and Prevention of Vaccine-Preventable Diseases, 13th Edition

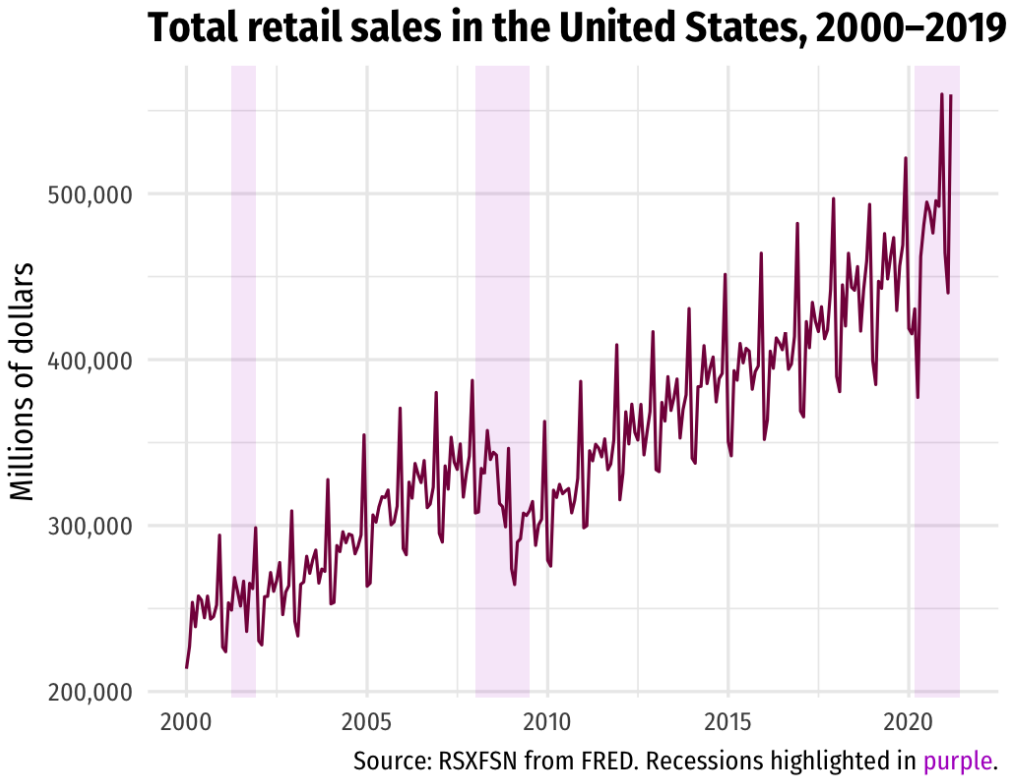
Measles vaccine was *incredible*!



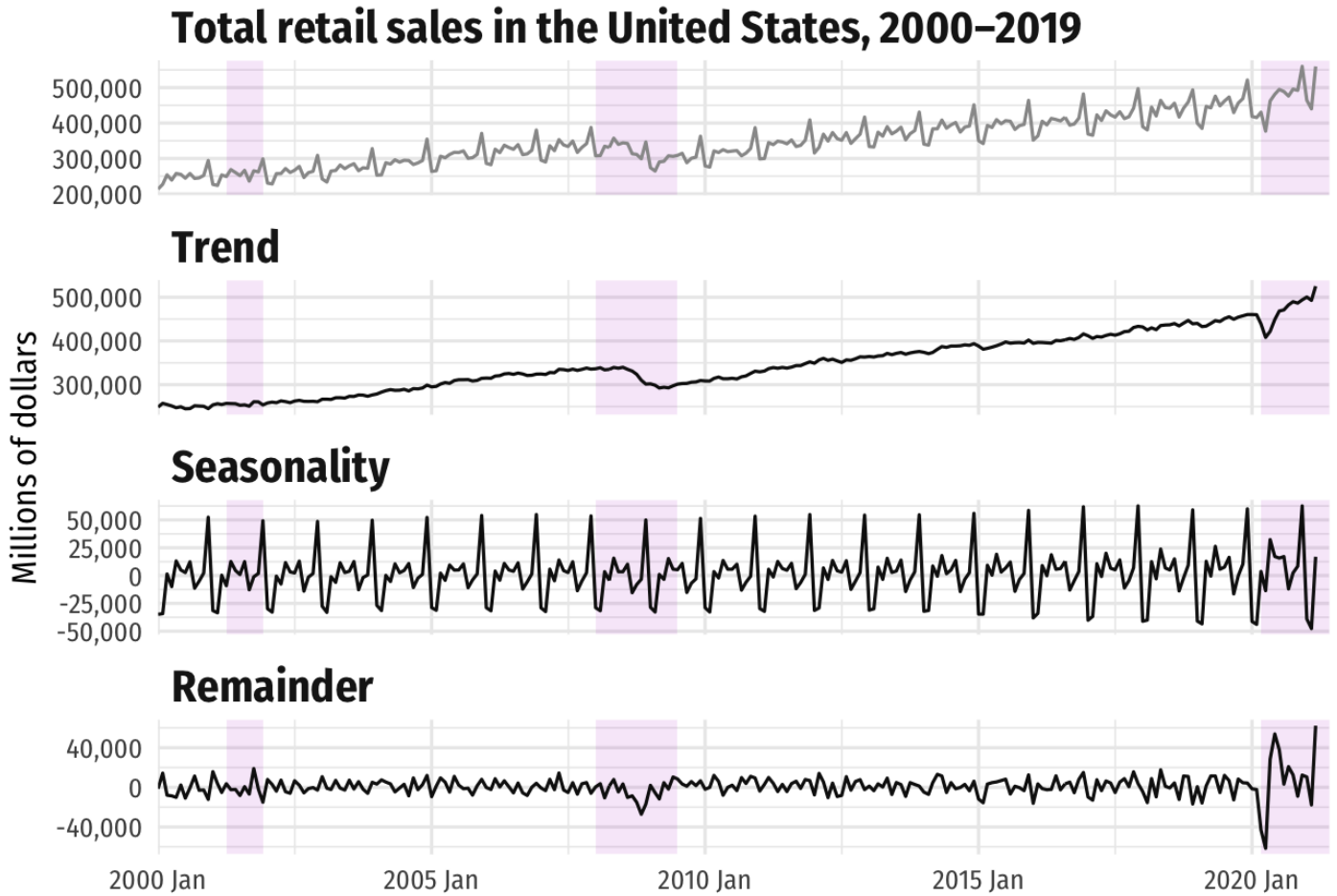
Seasonality

Don't mistake seasonality for actual trends



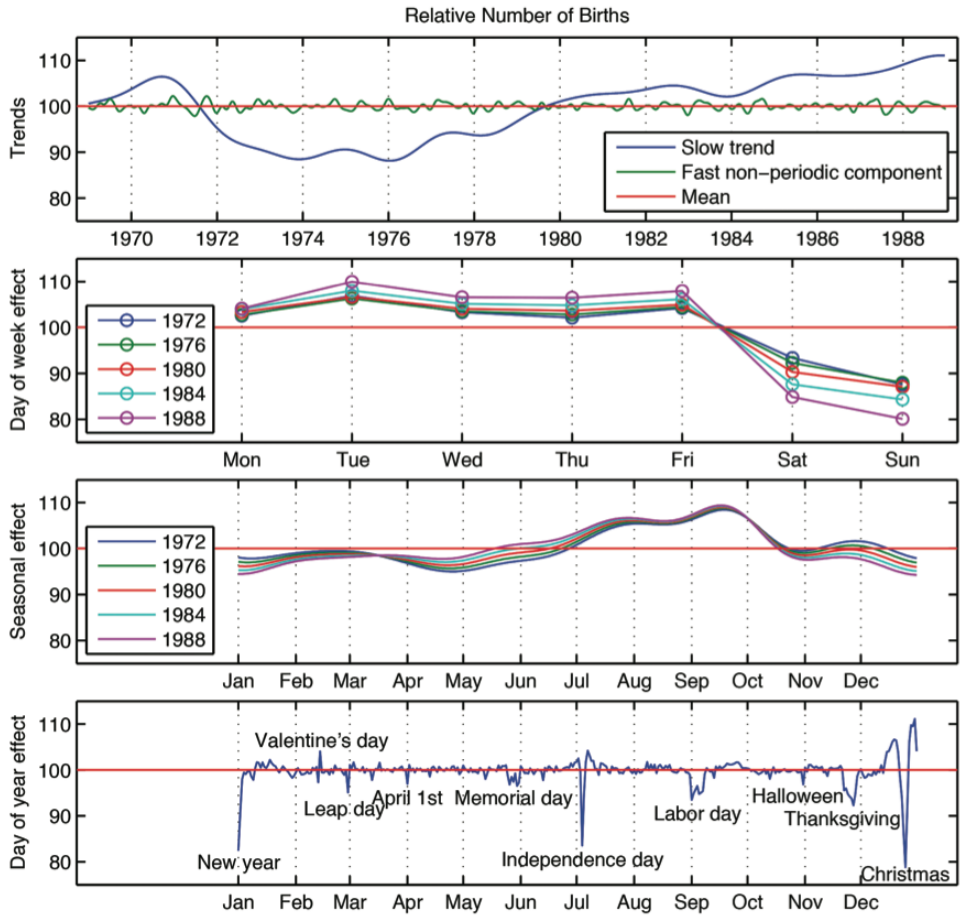


Seasonal adjustment



Source: RSXFSN from FRED. Recessions highlighted in purple.

Birthday decomposition



Cover of Andrew Gelman, et al., *Bayesian Data Analysis*

Dates and times in R

Are dates and times special?

We made a bunch of plots without really thinking about data types

We can map "dates" to x regardless of whether they are `date` or `numeric` objects

```
class(gme_prices$date)
```

```
## [1] "Date"
```

```
class(solar_data$date)
```

```
## [1] "Date"
```

```
class(measles$Year)
```

```
## [1] "numeric"
```

When do we need to pay attention to the details?

1. Converting from strings to dates
2. Getting components of date-time data
3. Computing time spans

Dates and times in R

R has "native" classes for storing calendar dates and times

- For background, see `?Dates` and `?DateTimeClasses`

The `lubridate` package offers convenient tools for working with dates and times

- See `vignette("lubridate")` and Chapter 16 of R4DS

Converting from strings to dates

Use `lubridate::ymd` to parse dates with **y**ear, **m**onth, and **d**ay components

```
independence_declared <- ymd("1776-07-04")
independence_declared
```

```
## [1] "1776-07-04"
```

```
class(independence_declared)
```

```
## [1] "Date"
```

`lubridate` offers many other functions for parsing dates. For example:

```
emancipation_proclaimed <- mdy("January 1, 1863")
emancipation_proclaimed
```

```
## [1] "1863-01-01"
```

Getting components of date-time data

What year was the Emancipation Proclamation?

```
year(emancipation_proclaimed)
```

```
## [1] 1863
```

What day of the week was that?

```
wday(emancipation_proclaimed)
```

```
## [1] 5
```

```
wday(emancipation_proclaimed, label = TRUE)
```

```
## [1] Thu
## Levels: Sun < Mon < Tue < Wed < Thu < Fri < Sat
```

What month was that?

```
month(emancipation_proclaimed)
```

```
## [1] 1
```

```
month(emancipation_proclaimed, label = TRUE)
```

```
## [1] Jan
## 12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul
```

Computing time spans

How many days passed between independence and emancipation?

```
emancipation_proclaimed - independence_declared
```

```
## Time difference of 31591 days
```

How many days are left until classes end?

```
ymd(20220510) - today()
```

```
## Time difference of 49 days
```

These are **durations**

See Chapter 16 of R4DS for functions to compute **periods** and **intervals**