Computational Social Science

Tabular data and visualization

Dr. Thomas Davidson

Rutgers University

January 29, 2024

Plan

- Recap
- Part I
  - Agent-based modeling
- Part II
  - Working with tabular data in R

Recap

Programming fundamentals

- Boolean logic
- If-else statements
- Loops
- Functions
- Pipes

Part I

Agent-based modeling

What is agent-based modeling?

Agent-based modeling and quantitative social science

- Most quantitative social science is *variable-centered*. We study the associations and interactions between variables using statistical techniques.
- For example, what is the association between college attendance and income for adults?
  - *Dependent variable*: Income
  - *Independent variable*: College attendance
  - *Controls*: Test scores, parental education, socioeconomic status, gender, etc.

What is agent-based modeling?

Agent-based modeling and quantitative social science

- Agent-based modeling is the study of "social life as interactions among adaptive agents who influence one another in response to the influence they receive." (Macy and Willer 2002)
  - Rather than interactions between variables, we consider *interactions between interdependent individuals*

What is agent-based modeling?

Agent-based modeling and quantitative social science

- Often we are interested in the *emergent* properties of local interactions between agents and how they aggregate into system-level processes such as diffusion, polarization, and segregation
  - How do cultural tastes spread through social networks?
  - Why do people become polarized about politics?
  - What explains patterns of residential segregation?

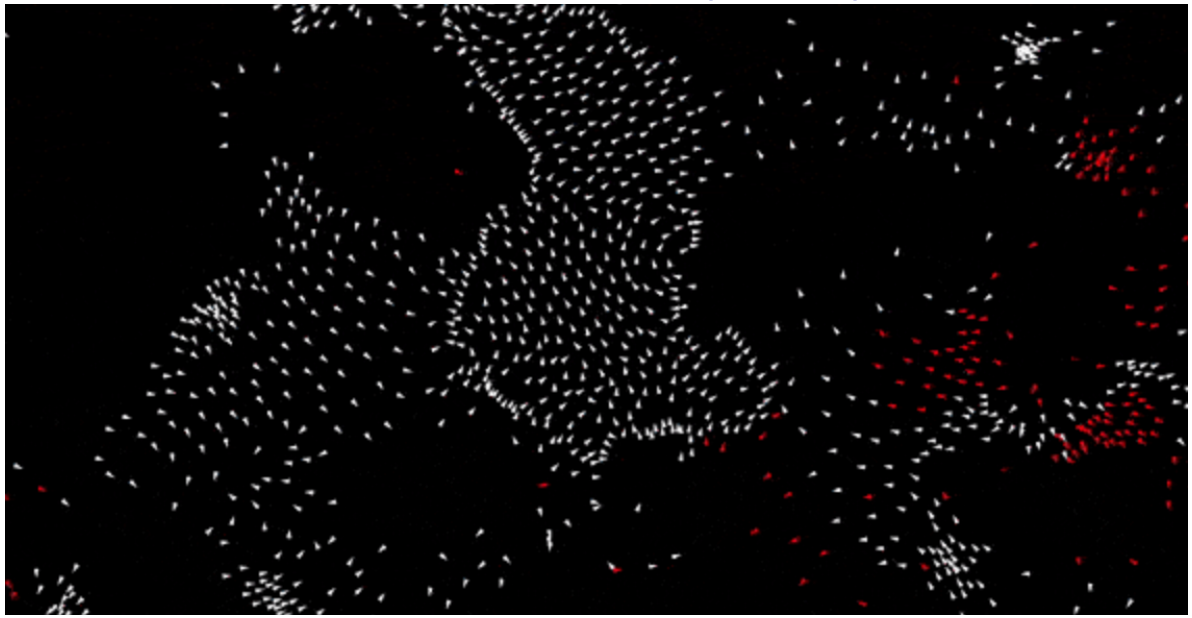
What is agent-based modeling?

Key assumptions

- Macy and Willer 2002 outline four key assumptions that underpin many sociological agent-based models
  - Agents are *autonomous*
    - There is no system-wide coordination
  - Agents are *interdependent*
    - Agents respond to each other and to their environment
  - Agents follow *simple rules*
    - Simple local rules can generate global complexity
  - Agents are *adaptive* and *backwards looking*
    - Agents can alter their behavior through processes such as imitation and learning

What is agent-based modeling?

Craig Reynolds *Flocking behavior* (1987)



Reynolds, Craig W. 1987. "Flocks, Herds and Schools: A Distributed Behavioral Model." In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, 25–34.

What is agent-based modeling?

Thomas Schelling *Homophily and segregation*

DYNAMIC MODELS OF SEGREGATION

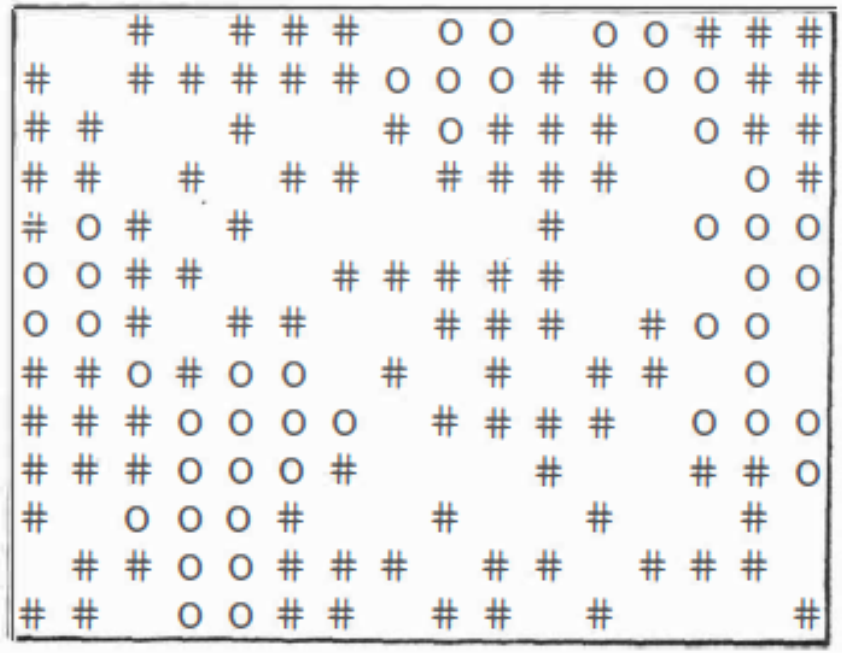
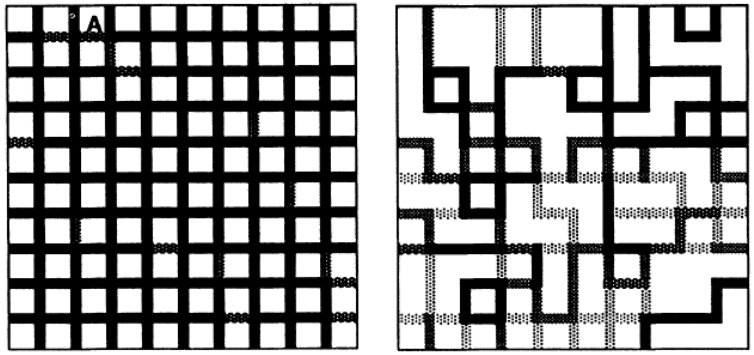


Fig. 13

Schelling, Thomas C. 1971. "Dynamic Models of Segregation." *Journal of Mathematical Sociology* 1: 143–86.

What is agent-based modeling?

Robert Axelrod *Local convergence and global polarization* (1987)



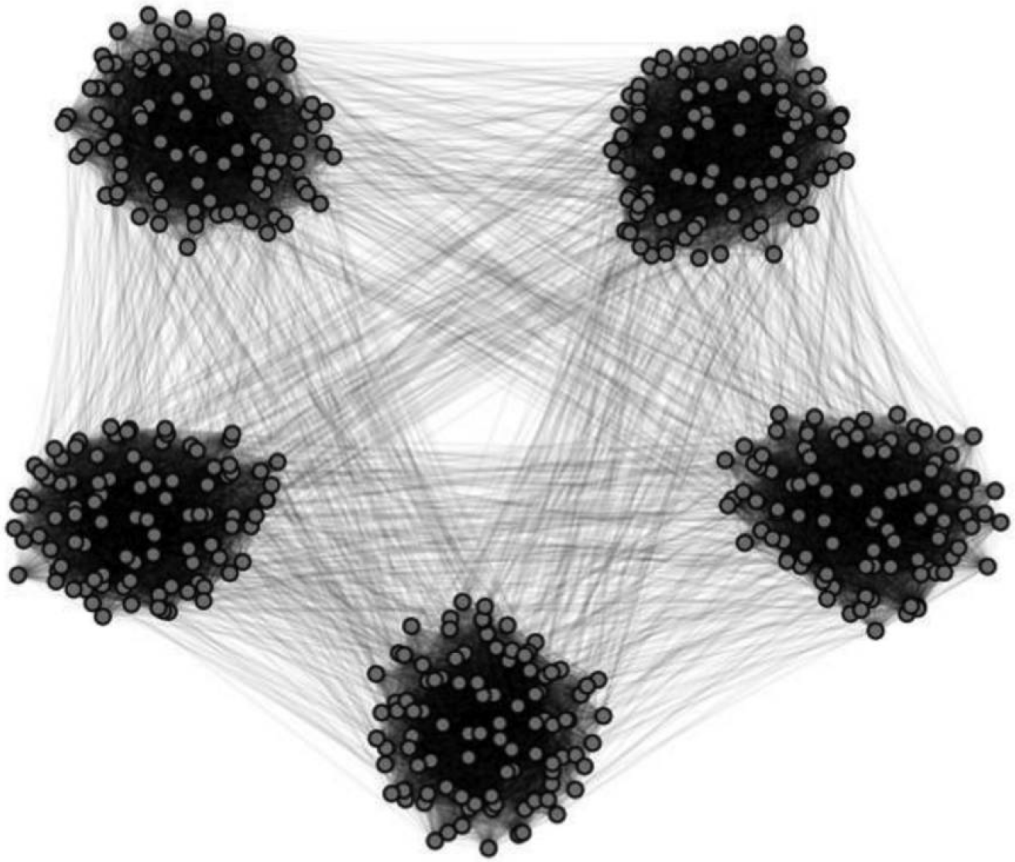
(a) At start (b) After 20,000 events

(c) After 40,000 events (d) After 80,000 events

Axelrod, Robert. 1997. "The Dissemination of Culture: A Model with Local Convergence and Global Polarization." *Journal of Conflict Resolution* 41 (2): 203–26.

What is agent-based modeling?

DellaPosta, Shi, and Macy *Why Do Liberals Drink Lattes?* (2015)



DellaPosta, Daniel, Yongren Shi, and Michael Macy. 2015. "Why Do Liberals Drink Lattes?" *American Journal of Sociology* 120(5): 1473–1511.

What is agent-based modeling?

Why use it?

- Benefits
  - Test theories that are hard to evaluate using data
  - Dynamic and flexible
- Drawbacks
  - Works best with simple models, intractable with more complex simulations
  - Difficult to evaluate or benchmark

Example: Schelling's segregation model

Copy the code at this link and try to run it as an R script in RStudio: http://tinyurl.com/schellingR

Part II

Tabular data and data visualization

Tabular data

- Often we will be using intermediate objects like lists, vectors, and matrices in conjunction with these different programming techniques to produce tabular data

Tabular data

Data frames

These data extend some of the functionality of basic types. A data frame is a tabular representation that has a similar structure to a list but allows us to organize the data more neatly.

```
data(iris) # The `data` function loads a built in dataset
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
iris$Sepal.Length[1] # explicitly call column name
```

```
## [1] 5.1
```

```
iris[[1]][1] # reference column using index
```

```
## [1] 5.1
```

Tabular data

The tidyverse

- The `tidyverse` is an R package that contains an entire suite of functions designed for such tasks including
  - `dplyr` contains functions for basic manipulation and merges and joins
  - `tidyr` implements functions for data cleaning and shape transformations
  - `purr` contains methods to easily map functions to lists and data frames.
- RStudio cheatsheets is an excellent resource.

Tabular data

The tidyverse

```
library(tidyverse)
tidyverse::tidyverse_packages()
```

```
##  [1] "broom"        "conflicted"    "cli"          "dbplyr"
##  [5] "dplyr"        "dtplyr"        "forcats"      "ggplot2"
##  [9] "googledrive"  "googlesheets4" "haven"        "hms"
## [13] "httr"         "jsonlite"      "lubridate"    "magrittr"
## [17] "modelr"       "pillar"        "purrr"        "ragg"
## [21] "readr"        "readxl"        "reprex"       "rlang"
## [25] "rstudioapi"   "rvest"         "stringr"      "tibble"
## [29] "tidyr"        "xml2"          "tidyverse"
```

Visit the tidyverse website for more information on the different packages website

Tabular data

Tibbles

A `tibble` is the `tidyverse` take on a data.frame. We can easily convert any `data.frame` into a `tibble`.

```
iris.t <- as_tibble(iris) # convert to tibble
class(iris.t)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

Tabular data

Tibbles

Tibbles only show the first ten rows when printing (both look the same in RMarkdown, so we have to use the console to compare.) Tibbles also provide information on the type of each variable.

```
print(iris.t)
```

```
## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>       <dbl>        <dbl>       <dbl> <fct>
##  1          5.1         3.5          1.4         0.2 setosa
##  2          4.9         3            1.4         0.2 setosa
##  3          4.7         3.2          1.3         0.2 setosa
##  4          4.6         3.1          1.5         0.2 setosa
##  5          5           3.6          1.4         0.2 setosa
##  6          5.4         3.9          1.7         0.4 setosa
##  7          4.6         3.4          1.4         0.3 setosa
##  8          5           3.4          1.5         0.2 setosa
##  9          4.4         2.9          1.4         0.2 setosa
## 10          4.9         3.1          1.5         0.1 setosa
```

Tabular data

Tibbles

Tibbles also tend to provide more warnings when potential issues arise, so they should be less prone to errors than data frames.

```
iris$year # data.frame shows null
```

```
## NULL
```

```
iris.t$year # tibble provides a warning
```

```
## Warning: Unknown or uninitialised column: `year`.
## NULL
```

Tabular data

Reading data

We can read data from files or directly from the web using `readr`. Here we're reading in data from the *New York Times* state-level COVID-19 tracker. The `glimpse` command shows us a preview of the table. We can use `View` to open up the data in a new window.

```
c19 <- read_csv("https://raw.githubusercontent.com/nytimes/covid-19-dat
dim(c19)
```

```
## [1] 61942     5
```

```
glimpse(c19)
```

```
## Rows: 61,942
## Columns: 5
## $ date   <date> 2020-01-21, 2020-01-22, 2020-01-23, 2020-01-24, 2020
## $ state  <chr> "Washington", "Washington", "Washington", "Illinois",
## $ fips   <chr> "53", "53", "53", "17", "53", "06", "17", "53", "04",
## $ cases  <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2,
## $ deaths <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
tail(c19)
```

Tabular data

Selecting columns

We can use the select command to select subsets of columns in the dataset.

```
c19 %>%
    select(date, state, cases)  # Select these columns
```

```
## # A tibble: 61,942 x 3
##    date       state      cases
##    <date>     <chr>      <dbl>
##  1 2020-01-21 Washington     1
##  2 2020-01-22 Washington     1
##  3 2020-01-23 Washington     1
##  4 2020-01-24 Illinois       1
##  5 2020-01-24 Washington     1
##  6 2020-01-25 California     1
##  7 2020-01-25 Illinois       1
##  8 2020-01-25 Washington     1
##  9 2020-01-26 Arizona        1
## 10 2020-01-26 California     2
```

Tabular data

Filtering

The `filter` command allows us to subset rows that meet one or more conditions.

```
c19 %>%
    filter(cases > 1000)  # conditional filtering
```

```
## # A tibble: 58,936 x 4
##    date       state      cases deaths
##    <date>     <chr>      <dbl>  <dbl>
##  1 2020-03-17 New York    1375     18
##  2 2020-03-18 New York    2387     32
##  3 2020-03-18 Washington  1026     68
##  4 2020-03-19 California  1067     19
##  5 2020-03-19 New York    4161     39
##  6 2020-03-19 Washington  1228     75
##  7 2020-03-20 California  1283     24
##  8 2020-03-20 New York    7113     68
##  9 2020-03-20 Washington  1404     83
## 10 2020-03-21 California  1544     28
```

Tabular data

Sampling

We can also filter our dataset by taking a sample. This can be very useful for testing purposes.

```
sample_n(c19, 10)  # Randomly pick n rows
```

```
## # A tibble: 10 x 4
##    date       state                cases deaths
##    <date>     <chr>                <dbl>  <dbl>
##  1 2020-12-03 Colorado            248779   3335
##  2 2022-02-03 Wyoming             149126   1650
##  3 2021-02-18 Wisconsin           610056   6820
##  4 2020-06-28 Tennessee            39945    577
##  5 2021-08-10 District of Columbia  51726   1149
##  6 2021-12-02 Nebraska            312517   3168
##  7 2022-12-02 Puerto Rico        1048678   5383
##  8 2022-09-12 Iowa                848811   9968
##  9 2021-02-18 Idaho               168783   1828
## 10 2021-01-26 Florida            1667755  25672
```

```
sample_frac(c19, 0.01)  # Randomly pick 1% of rows
```

Tabular data

Slicing

The `slice` commands can be used to select ordered subsets of rows.

```
slice_max(c19, order_by = cases, n = 10)  # Get the top n rows by a spe
```

```
## # A tibble: 10 x 4
##    date       state          cases deaths
##    <date>     <chr>          <dbl>  <dbl>
##  1 2023-03-23 California 12169158 104277
##  2 2023-03-22 California 12155467 104196
##  3 2023-03-21 California 12154941 104185
##  4 2023-03-20 California 12154293 104165
##  5 2023-03-19 California 12153083 104130
##  6 2023-03-17 California 12153079 104130
##  7 2023-03-18 California 12153079 104130
##  8 2023-03-16 California 12152129 104114
##  9 2023-03-15 California 12136243 104047
## 10 2023-03-14 California 12135588 104038
```

```
slice_min(c19, order_by = cases, n = 1)  # with_ties determines whether
```

Tabular data

Making new columns using mutate

The `mutate` function allows us to generate new columns.

```
c19 <- c19 %>%
    mutate(deaths_per_case = deaths/cases)
colnames(c19)
```

```
## [1] "date"            "state"           "cases"           "deaths"
## [5] "deaths_per_case"
```

Tabular data

Mutate

Although these data are cumulative, we can recover the new cases and deaths each day by using the `lag` operator.

```
c19 <- c19 %>%
    group_by(state) %>%
    mutate(new_cases = cases - lag(cases), new_deaths = deaths - lag(de
    ungroup()
tail(c19 %>%
    filter(state == "Oregon"))
```

```
## # A tibble: 6 x 7
##   date       state   cases deaths deaths_per_case new_cases new_deat
##   <date>     <chr>   <dbl>  <dbl>           <dbl>     <dbl>      <db
## 1 2023-03-18 Oregon 965286   9432         0.00977         0
## 2 2023-03-19 Oregon 965286   9432         0.00977         0
## 3 2023-03-20 Oregon 965286   9432         0.00977         0
## 4 2023-03-21 Oregon 965286   9432         0.00977         0
## 5 2023-03-22 Oregon 967156   9451         0.00977      1870
## 6 2023-03-23 Oregon 967156   9451         0.00977         0
```

Tabular data

Summarizing

We can use `summarize` to create statistical summaries of the data. Like `mutate`, we define a new variable within `summarize` to capture a defined summary.

```
# Summarize specific variables
c19 %>%
    summarise(mean_deaths = mean(deaths), median_deaths = median(deaths
```

```
## # A tibble: 1 x 3
##   mean_deaths median_deaths max_deaths
##         <dbl>         <dbl>      <dbl>
## 1      11779.          5035     104277
```

Tabular data

Summarizing

The `summarize_all` command takes a summary function (e.g. mean, min, max) and applies it to all columns. This can be useful if there are lots of variables. See documentation for other variants of summarize. Note that the mean is undefined for non-numeric columns AND columns with missing data.

```
c19 %>%
    summarize_all(mean)  # Map a summary function to all valid columns
```

```
## # A tibble: 1 x 7
##   date       state   cases deaths deaths_per_case new_cases new_deat
##   <date>     <dbl>   <dbl>  <dbl>           <dbl>     <dbl>      <db
## 1 2021-09-13    NA 889830. 11779.          0.0170        NA
```

Tabular data

Summarizing

We can *impute* missing data to get an estimate of the mean. In this case, values are missing for early rows where the lag operator was not defined. Missing `new_cases` or `new_deaths` will be set to zero using `replace_na`.

```
c19 <- c19 %>%
    replace_na(list(new_cases = 0, new_deaths = 0))
c19 %>%
    summarize_all(mean)  # Map a summary function to all valid columns
```

```
## # A tibble: 1 x 7
##   date       state   cases deaths deaths_per_case new_cases new_deat
##   <date>     <dbl>   <dbl>  <dbl>           <dbl>     <dbl>      <db
## 1 2021-09-13    NA 889830. 11779.          0.0170     1678.       18
```

Tabular data

Grouping

Often we want to group our data before summarizing. What does this example tell us?

```
c19 %>%
    group_by(state) %>%
    summarise(mean(deaths_per_case))
```

```
## # A tibble: 56 x 2
##    state                `mean(deaths_per_case)`
##    <chr>                                  <dbl>
##  1 Alabama                              0.0175
##  2 Alaska                               0.00598
##  3 American Samoa                       0.00267
##  4 Arizona                              0.0182
##  5 Arkansas                             0.0147
##  6 California                           0.0143
##  7 Colorado                             0.0165
##  8 Connecticut                          0.0316
##  9 Delaware                             0.0172
```

Tabular data

Grouping

Sometimes we might want to create a group-level variable then revert back to the original dataset. We can do this using the ungroup command. What does this new column represent?

```
c19 %>%
    group_by(date) %>%
    mutate(daily_mean = mean(new_cases)) %>%
    ungroup() %>%
    tail()
```

```
## # A tibble: 6 x 8
##   date       state  cases deaths deaths_per_case new_cases new_death
##   <date>     <chr>  <dbl>  <dbl>           <dbl>     <dbl>      <dbl
## 1 2023-03-23 Virg~ 2.48e4    130         0.00523         0
## 2 2023-03-23 Virg~ 2.30e6  23782         0.0103        454
## 3 2023-03-23 Wash~ 1.94e6  15905         0.00820         0
## 4 2023-03-23 West~ 6.46e5   8132         0.0126          0
## 5 2023-03-23 Wisc~ 2.01e6  16485         0.00818      1039
## 6 2023-03-23 Wyom~ 1.86e5   2014         0.0108          0
```

Tabular data

Joins

We often want to join together different datasets. Venn diagrams are a useful way for thinking about this.

Tabular data

Joins

The `left_join` is the most commonly used type of join. We keep all rows in our left dataset and the rows on the right dataset with valid matches. Here we're download a dataset about state governors and joining it on state. The `by` argument defines the columns we should join on.

```
gov <- read_csv("https://raw.githubusercontent.com/OpenGovDataMirror/F_
gov <- gov %>%
    select(state_name, party)  # just select two columns

c19 <- c19 %>%
    left_join(gov, by = c(state = "state_name"))  # We can pipe c19 int
```

Tabular data

Joining

Let's consider another example to get state-level population data. In this case, we're reading an Excel file from the Census bureau so we have to do a little more processing to load the file.

```
library(readxl)
census <- "https://www2.census.gov/programs-surveys/popest/tables/2010-
# read_excel function from readxl does not currently handle files from
# so we need to get it manually
tmp <- tempfile(fileext = ".xlsx")
httr::GET(url = census, httr::write_disk(tmp))
```

```
## Response [https://www2.census.gov/programs-surveys/popest/tables/201
##   Date: 2024-01-29 00:48
##   Status: 200
##   Content-Type: application/vnd.openxmlformats-officedocument.spread
##   Size: 18.1 kB
## <ON DISK>  /var/folders/by/t5qdf0996h12f6ngxhxrqpf40000gs/T//Rtmpoxb
```

```
pop <- read_excel(tmp)
```

Tabular data

Joining

These data are a little messier. We need to do a bit of cleaning up.

```
pop.states <- pop[9:61, c(1, 13)]
colnames(pop.states) <- c("state", "pop")
pop.states <- pop.states %>%
    mutate(state = str_replace(state, ".", "")) %>%
    drop_na()
```

Tabular data

Joining

Now we can join our new column to the dataset. Finally, we drop rows that do not have a governor (party column is missing).

```
c19 <- c19 %>%
    left_join(pop.states, by = "state")
c19 <- c19 %>%
    drop_na(party)  # Dropping any row not considered a state
length(unique(c19$state))  # Verifying the correct number of states
```

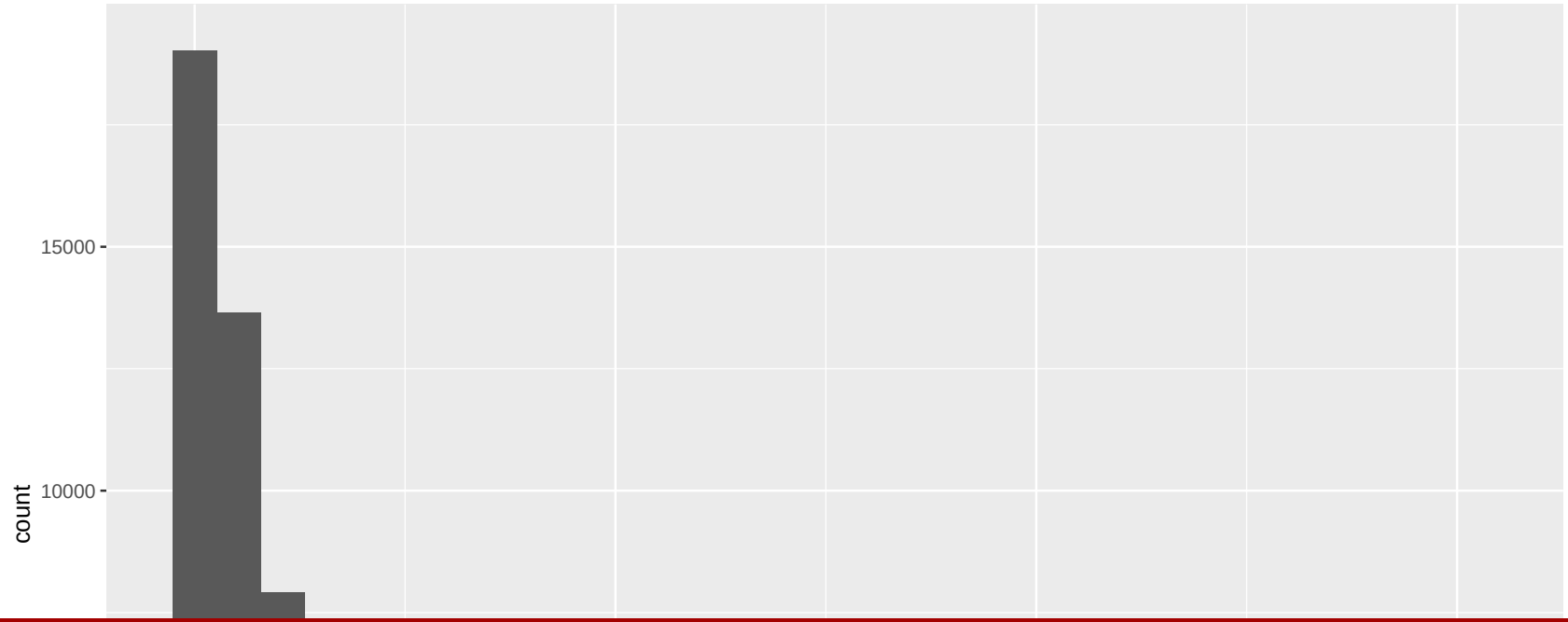
```
## [1] 50
```

Data visualization

ggplot2

The ggplot2 library is loaded as part of the tidyverse. It can produce may different styles of plots with a simple, tidy syntax. Let's consider a basic example.

```
ggplot(c19, # data
       aes(x = cases)) + # aesthetic mapping
    geom_histogram() # plot type
```

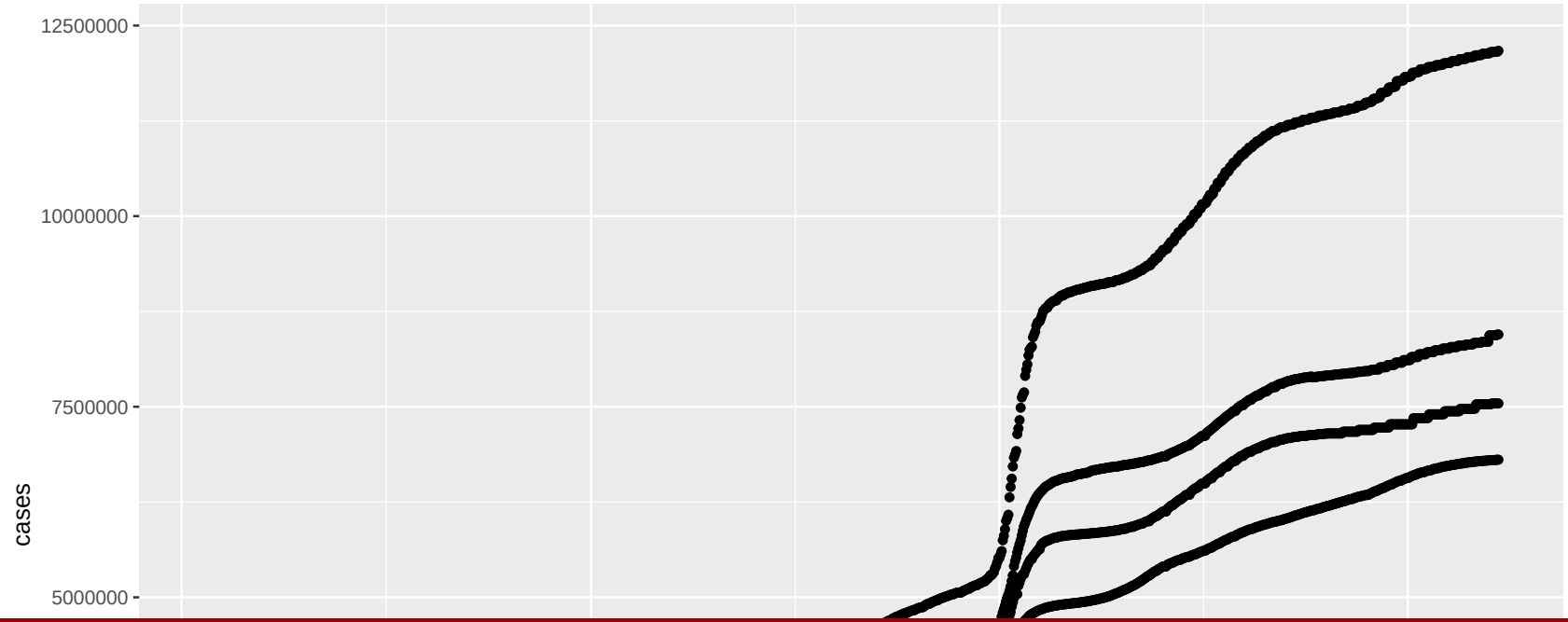


Data visualization

ggplot2

The previous histogram wasn't very informative because it doesn't show the trends over time. A better option would be to plot the cases over time.

```
ggplot(c19, # data
       aes(x = date, y= cases)) + # aesthetic mapping
    geom_point() # plot type
```



Data visualization

ggplot2

We can see that the points above are lines, since we have daily measures for each state. Let's examine the linear trend by plotting the line of best fit to the data points.

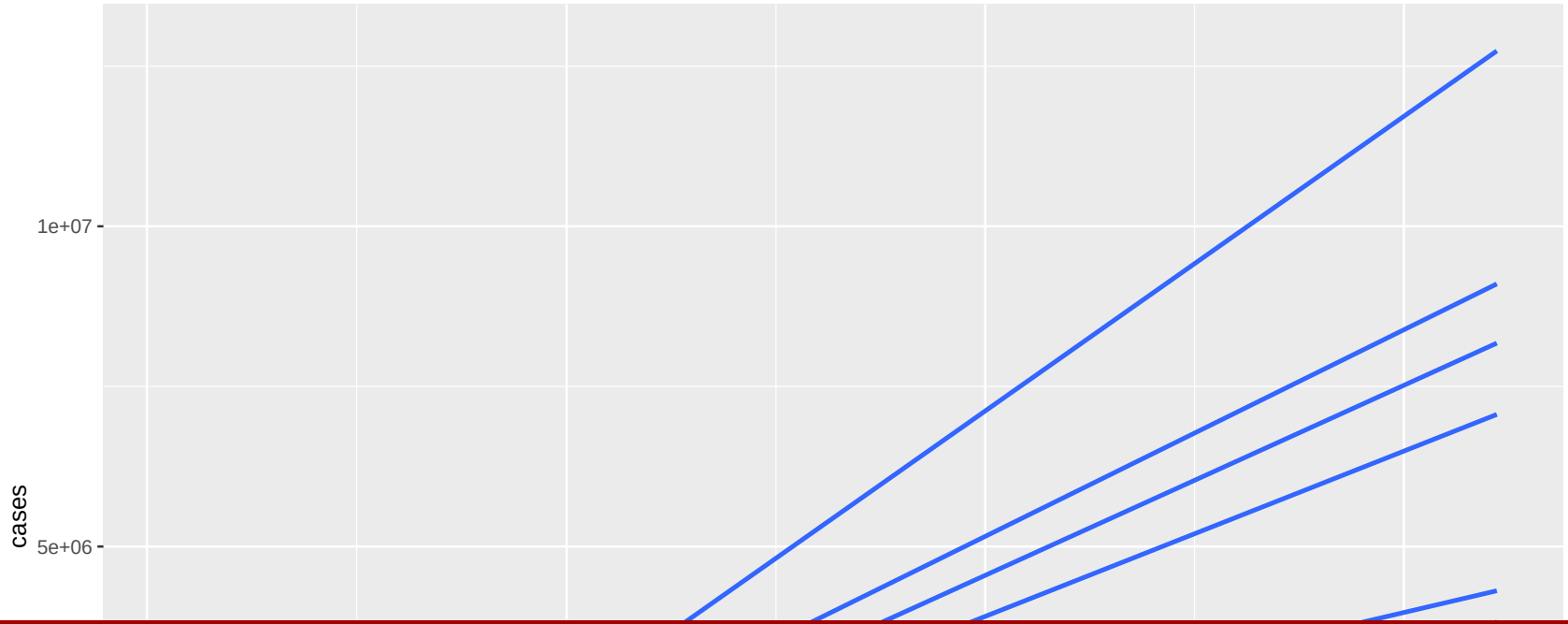
```
ggplot(c19, # data
       aes(x = date, y= cases)) + # aesthetic mapping
    geom_point() +
    geom_smooth(method='lm', se = F) # plot type
```

Data visualization

ggplot2

The previous line is not too informative due to variation among states. We can easily break it out by state by adding a group parameter. Now each state has a separate line fitted.

```
ggplot(c19, # data
       aes(x = date, y= cases, group=state)) + # aesthetic mapping
    geom_smooth(method='lm', se = F) # plot type
```

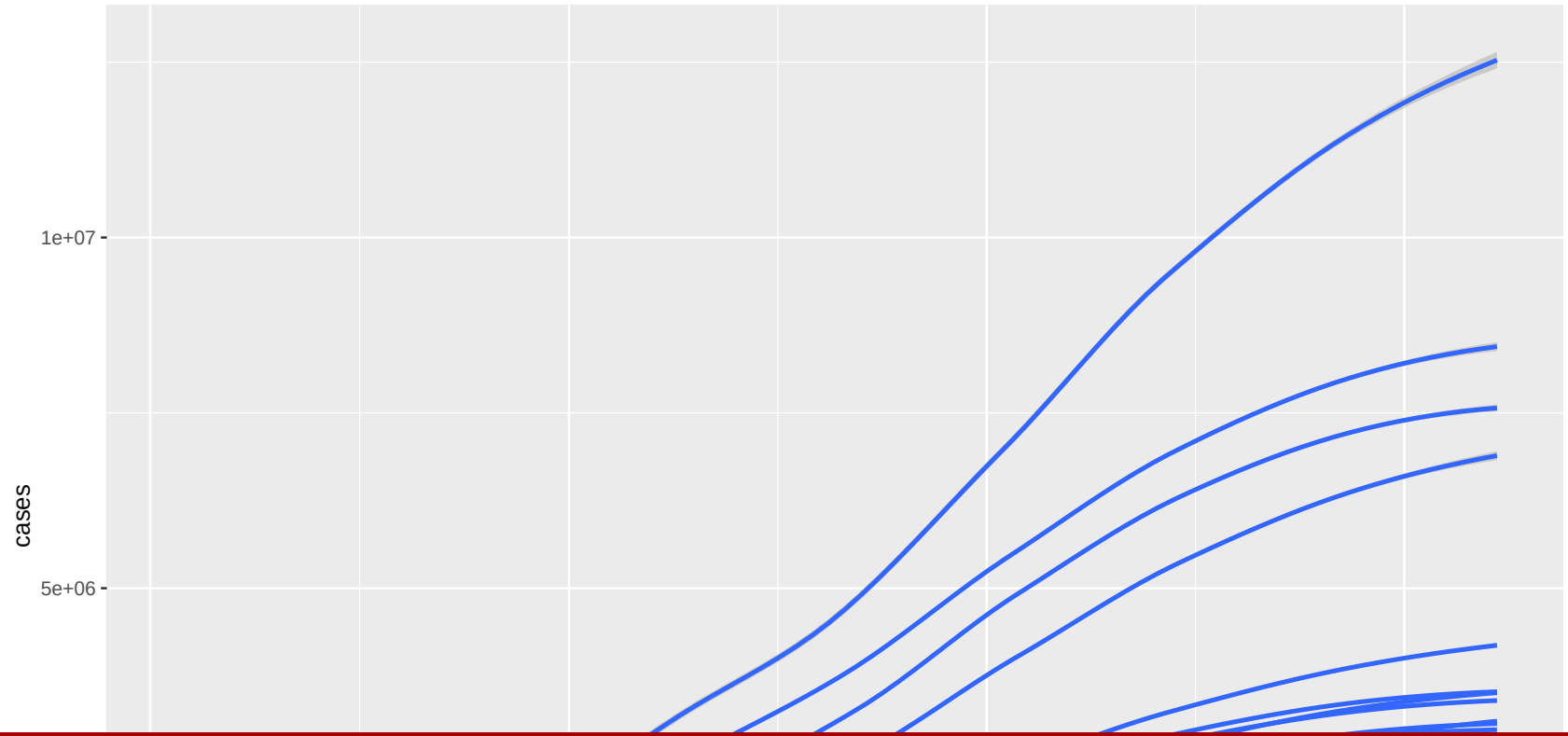


Data visualization

ggplot2

We can also fit a smoothed line to better capture the trends.

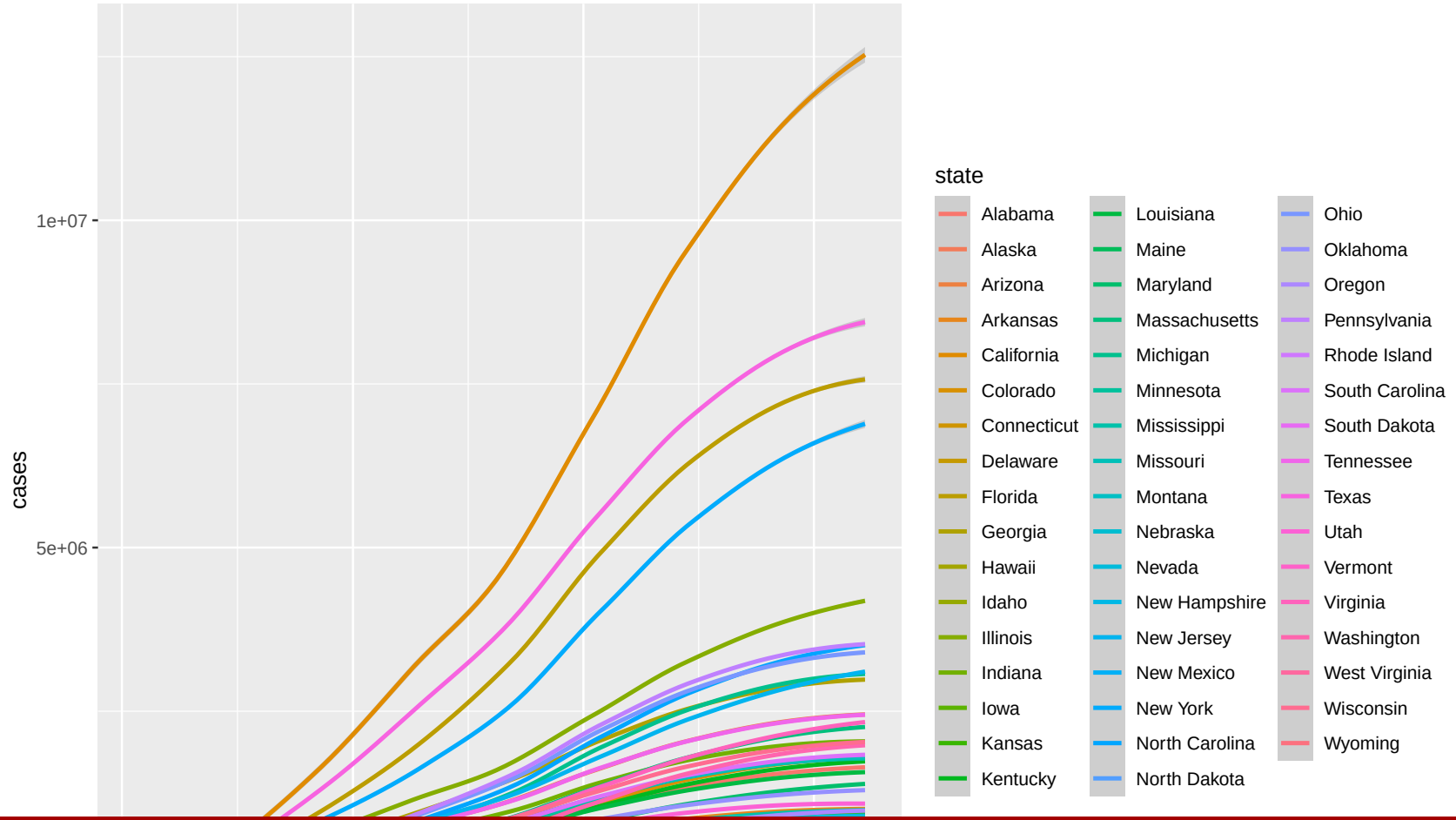
```
ggplot(c19, # data
       aes(x = date, y= cases, group=state)) + # aesthetic mapping
    geom_smooth(method='loess') # plot type
```



Data visualization

ggplot2

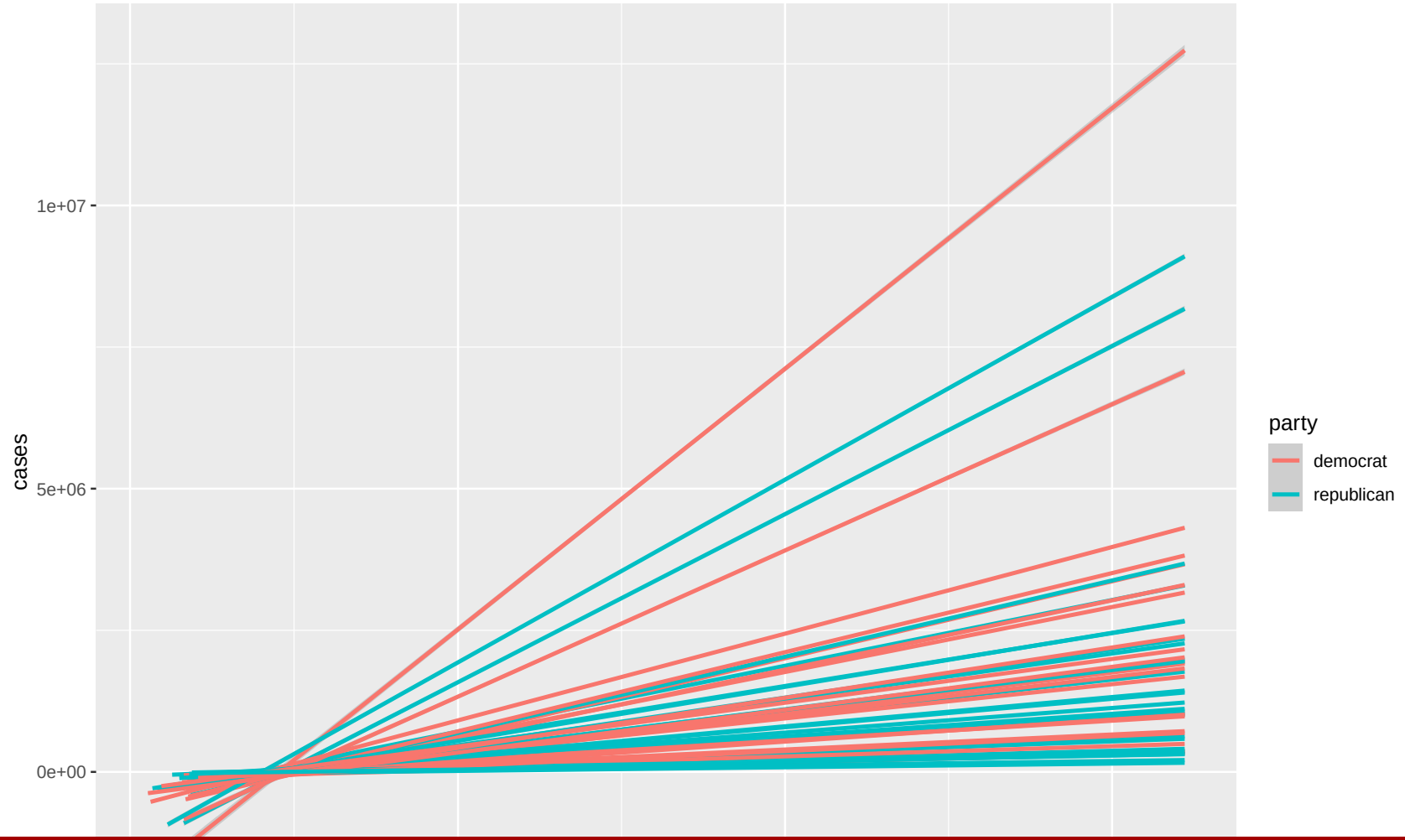
The color parameter allows us to assign a different color to each line. Note how things get a little difficult to read now.



Data visualization

ggplot2

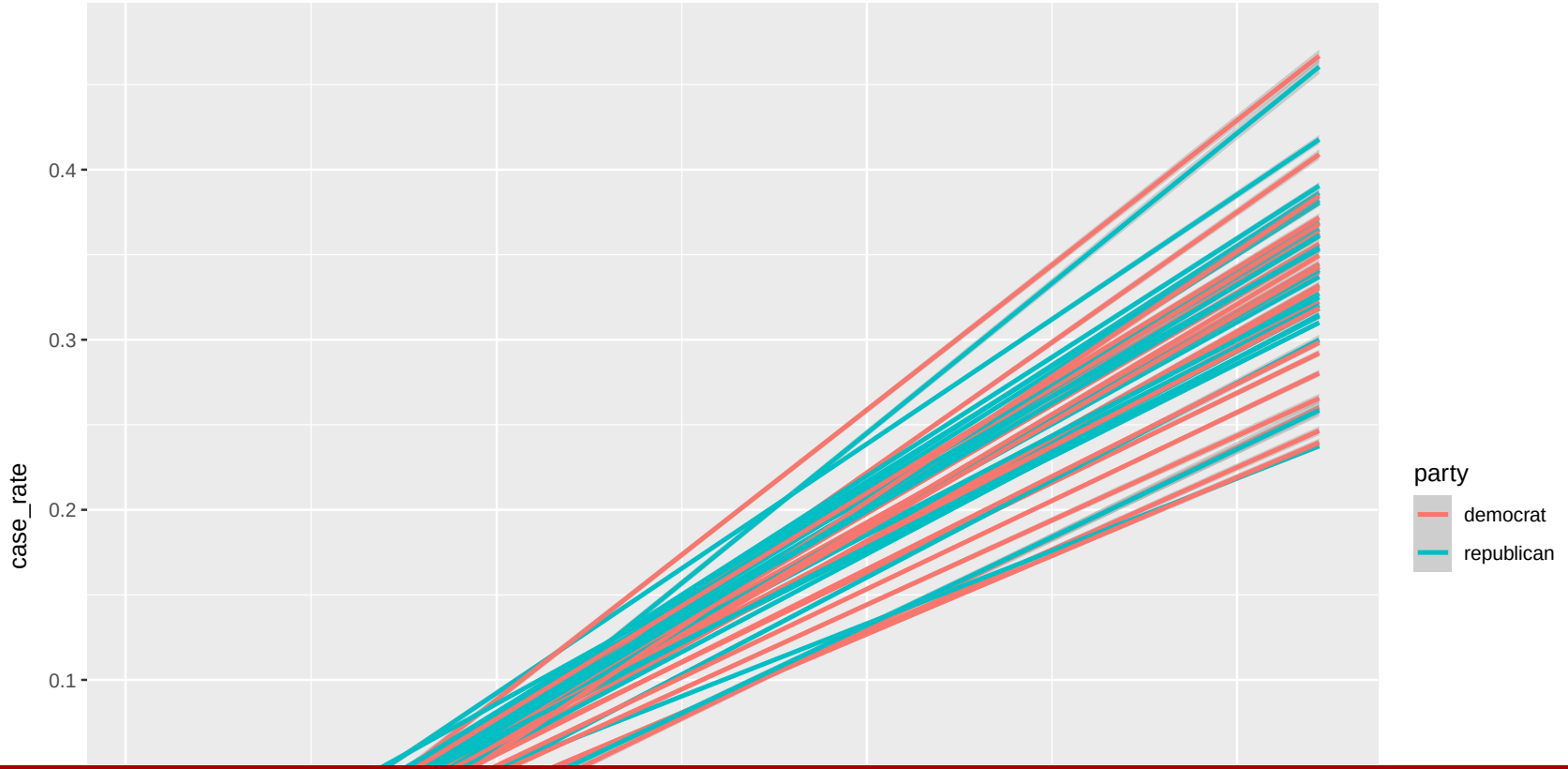
We can easily group by other variables.



Data visualization

ggplot2

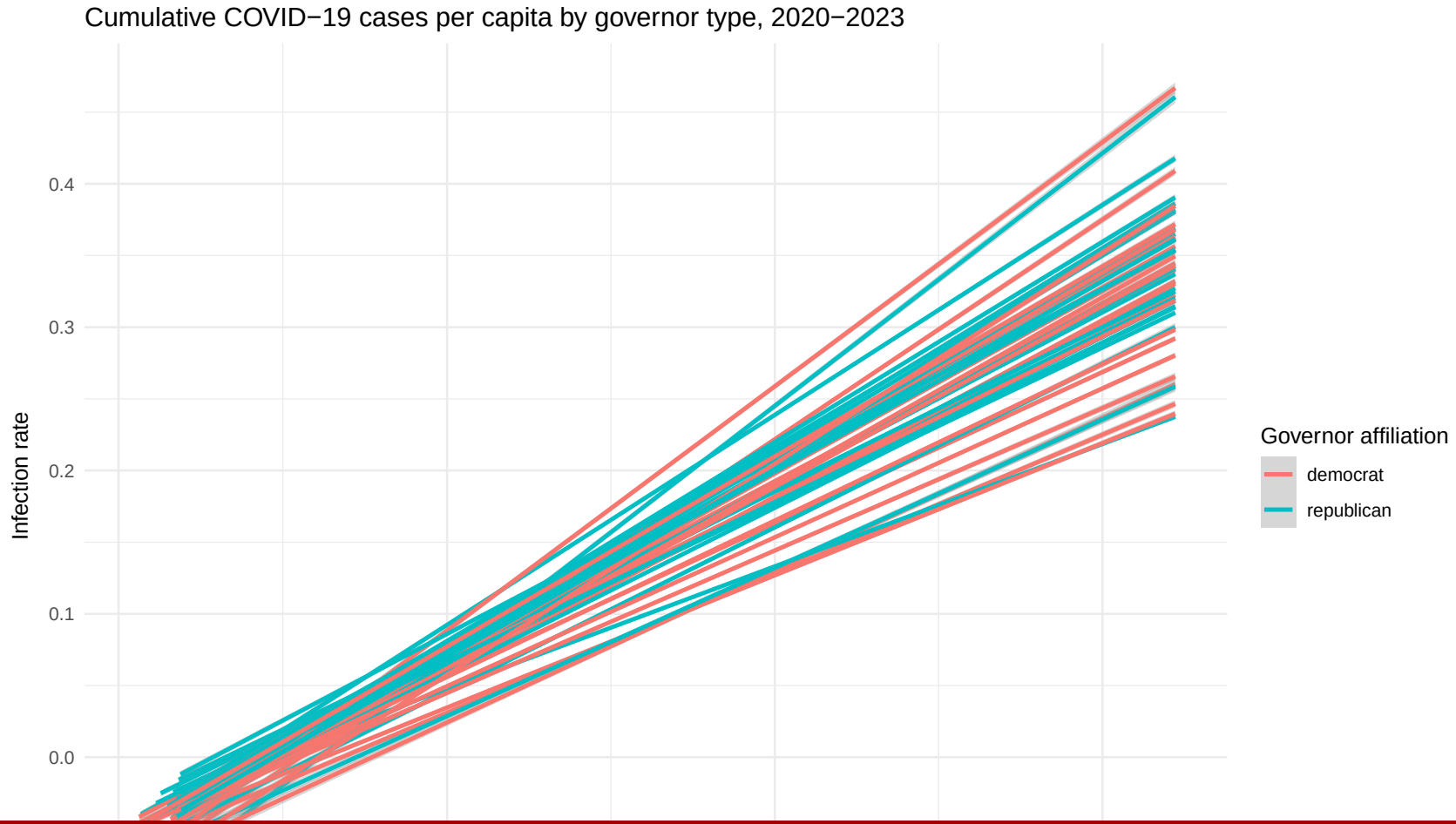
Why might the previous plot be misleading? Is there a better way to look at how cases vary by partisanship of the governor? Note: The plot is now stored as an object `p` before plotting. This allows us to modify it later on.



Data visualization

ggplot2

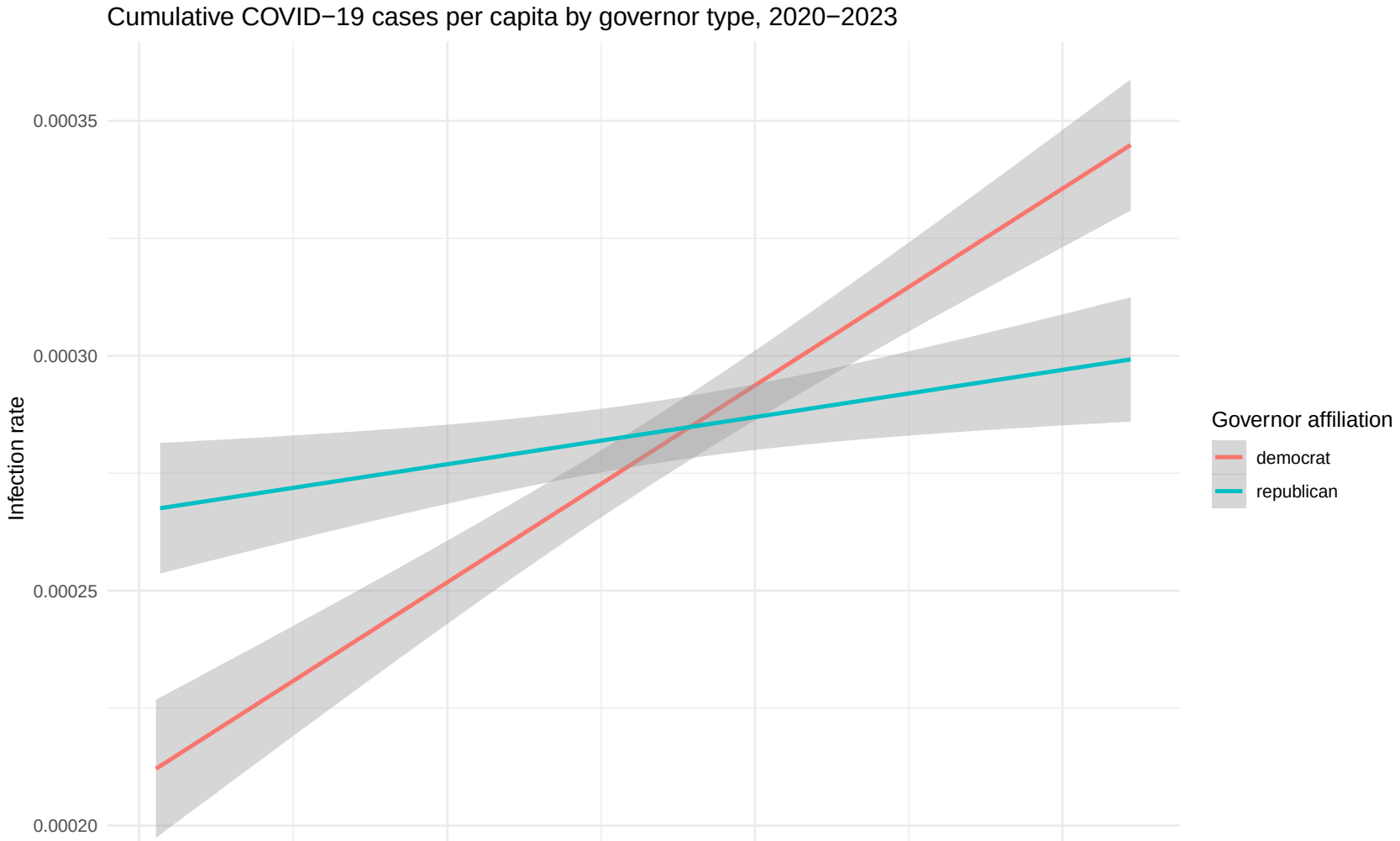
Now we have a plot, let's
make it look a bit nicer. We can easily add labels and modify the axes.



Data visualization

ggplot2

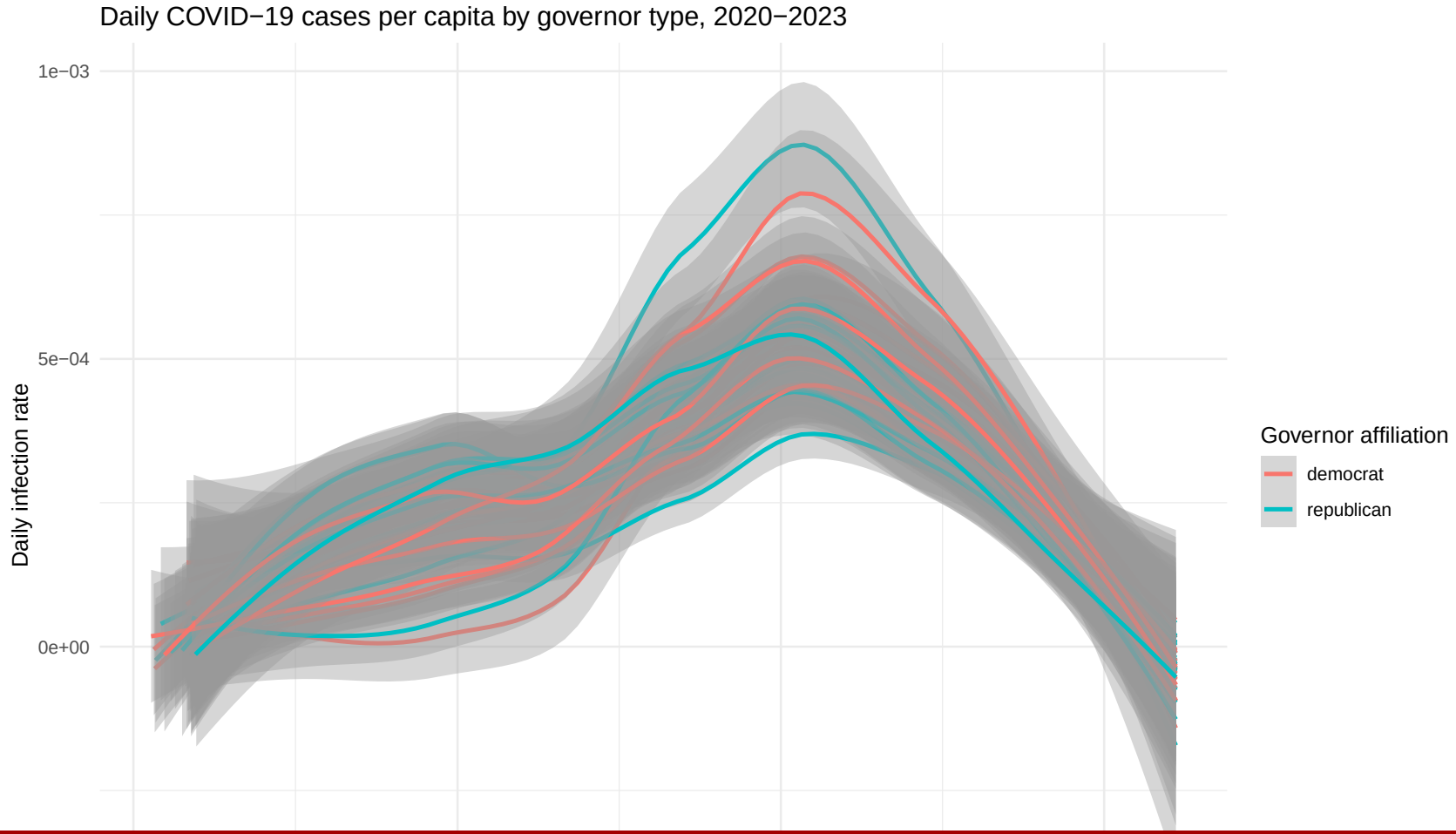
We can easily modify this code to look at the data in a different way.



Data visualization

ggplot2

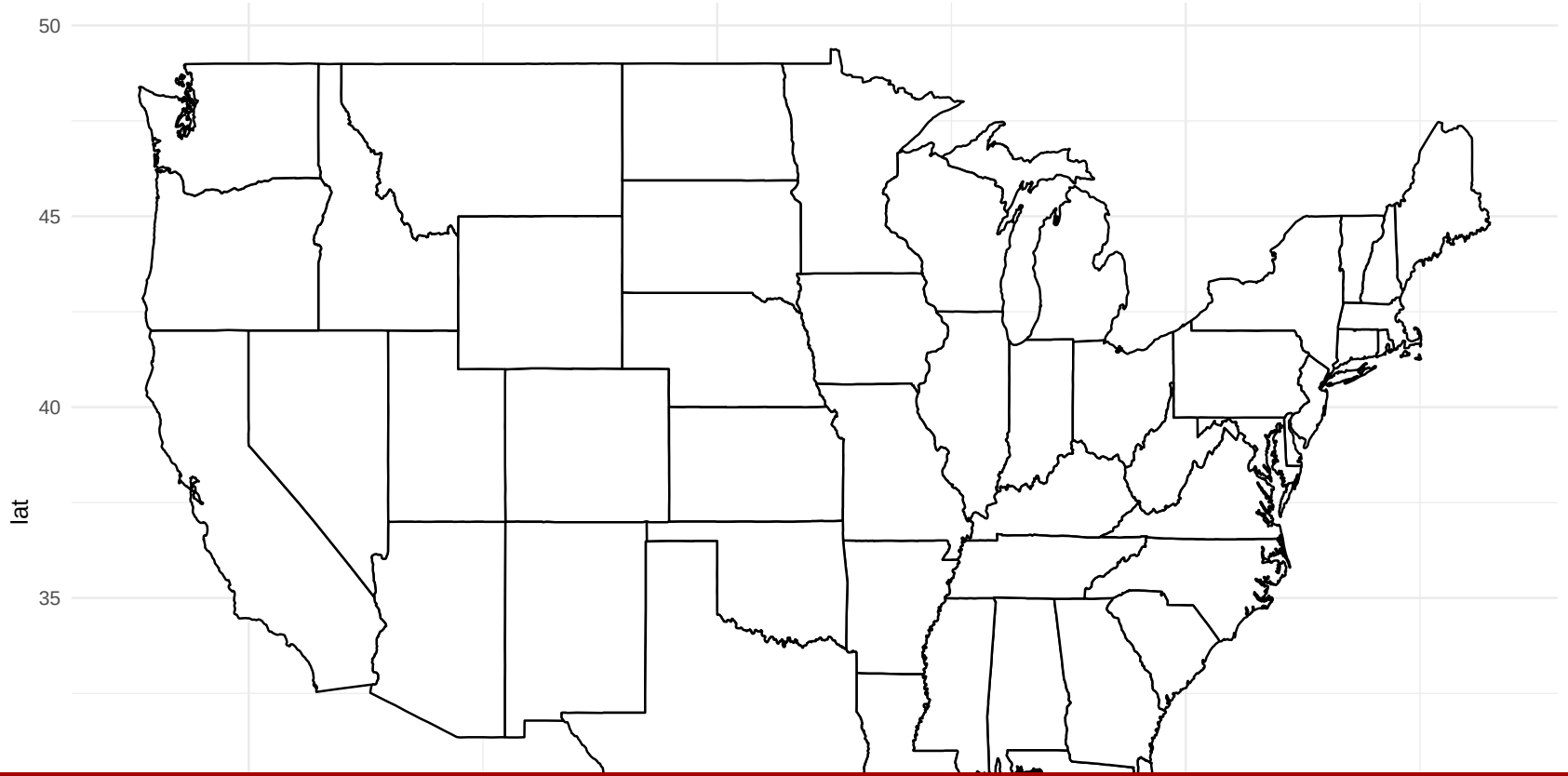
Based on these results, we might want to go back to a non-linear fit and allow each state to have its own line to better see the trends.



Data visualization

ggplot2 maps

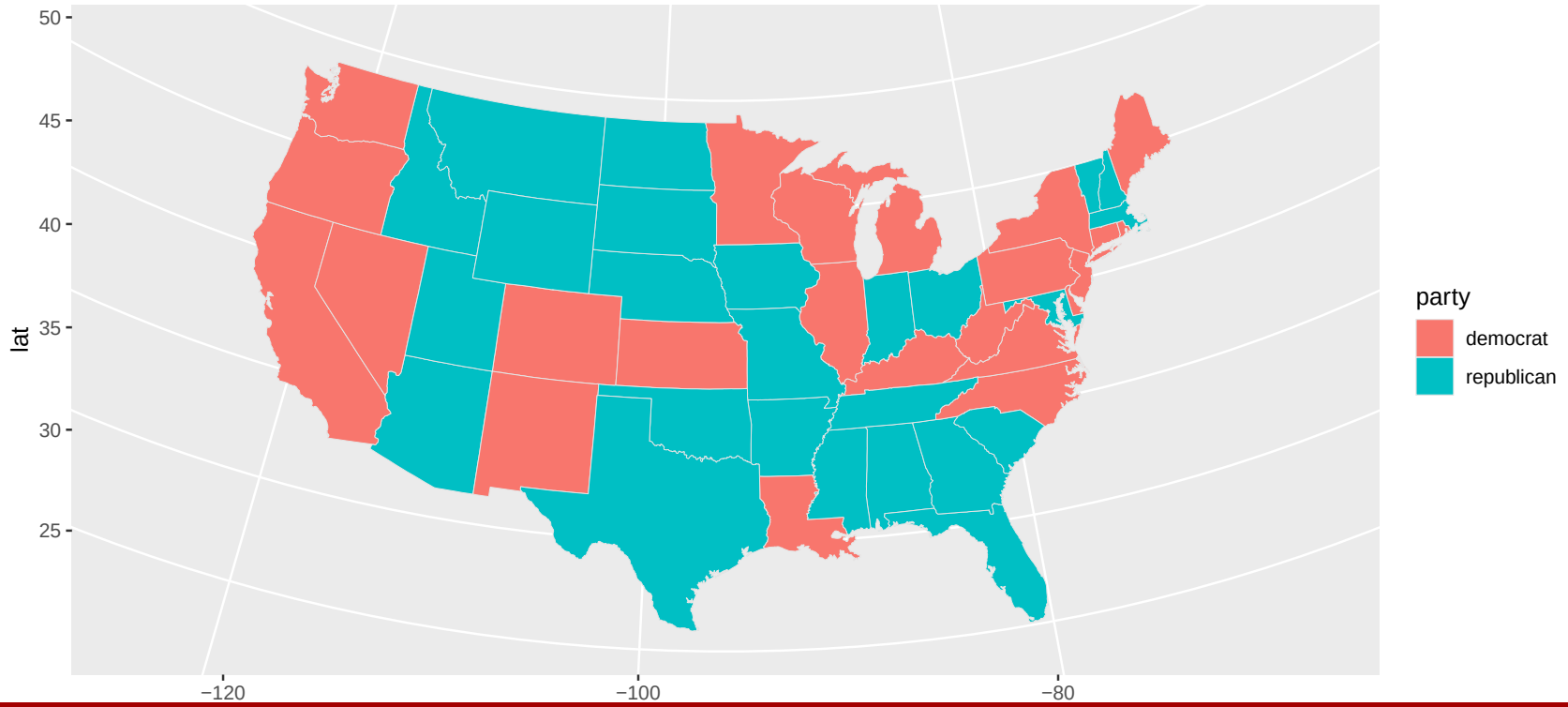
The `ggplot` package can be used to produce many different types of visualizations. For example, we can use it to produce maps. Here we load the package `maps` to get the shapefile for each state. The example



Data visualization

ggplot2

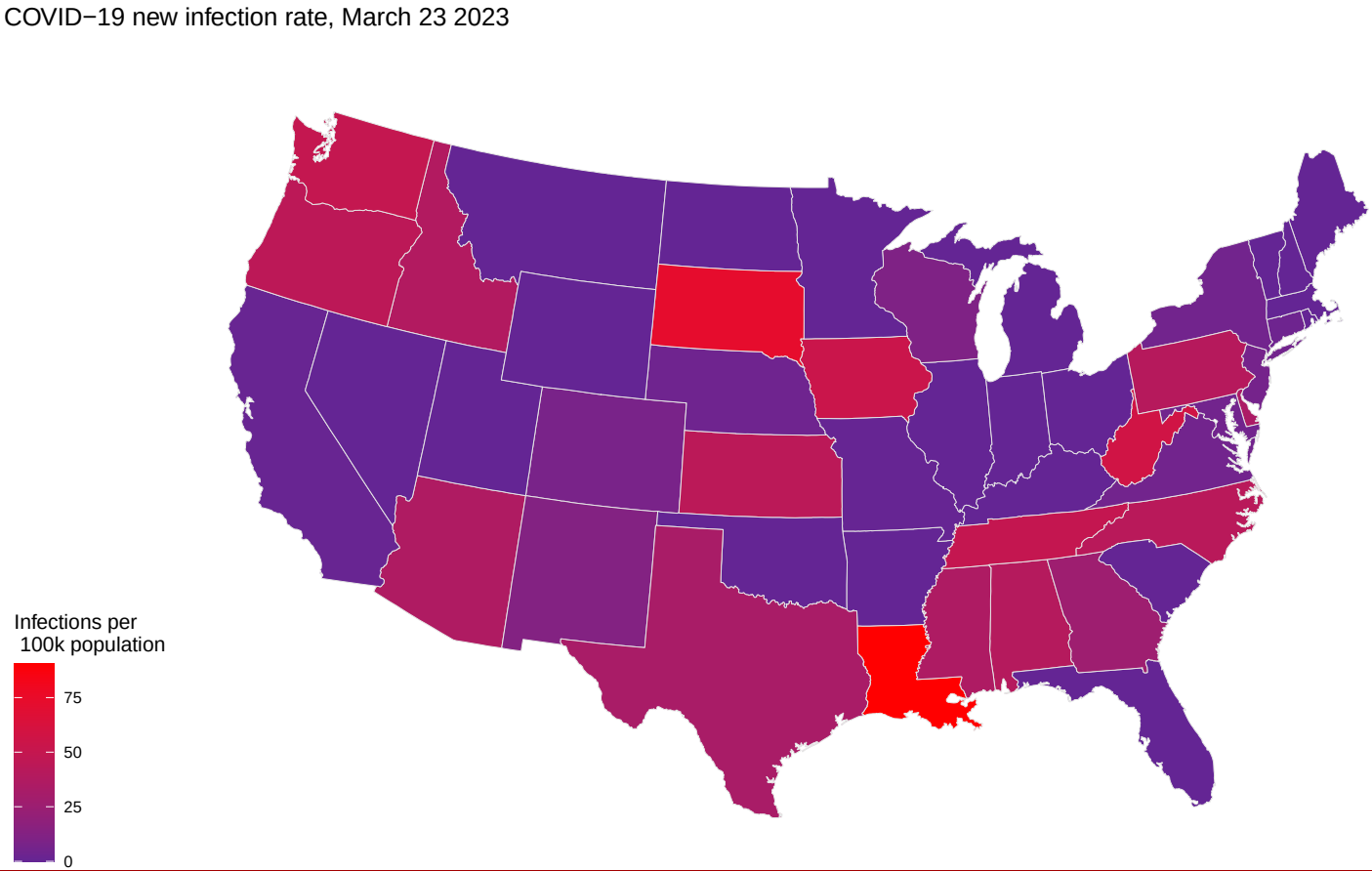
We have to merge our data with the shapefile in order to plot it on the map.



Data visualization

ggplot2

Let's try to do something more interesting.



Next lecture

- File management
- Github