Computational Social Science

Tabular data and visualization

Dr. Thomas Davidson

Rutgers University

January 31, 2022

Plan

- Recap
- Tabular data and the tidyverse
- Data visualization with ggplot2
- A primer on Github

Recap

Programming fundamentals

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

Tabular data

The tidyverse

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

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

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

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] 38590     5
```

```
glimpse(c19)
```

```
## Rows: 38,590
## 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,
```

```
head(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: 38,590 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 > 10000)  # conditional filtering
```

```
## # A tibble: 31,644 x 4
##    date       state      cases deaths
##    <date>     <chr>      <dbl>  <dbl>
##  1 2020-03-21 New York   10371     95
##  2 2020-03-22 New York   15188    142
##  3 2020-03-23 New York   20899    183
##  4 2020-03-24 New York   25704    264
##  5 2020-03-25 New York   33117    381
##  6 2020-03-26 New York   39058    502
##  7 2020-03-27 New York   44746    645
##  8 2020-03-28 New Jersey 11124    140
##  9 2020-03-28 New York   53517    935
## 10 2020-03-29 New Jersey 13386    161
```

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 2021-03-09 Connecticut    288145   7739
##  2 2020-12-31 Virgin Islands   2031     23
##  3 2021-06-22 Idaho          194444   2142
##  4 2020-07-19 Georgia        130794   3110
##  5 2021-02-08 Rhode Island   119104   2236
##  6 2021-09-09 Guam            12788    155
##  7 2021-01-24 Virginia       472447   6078
##  8 2020-04-22 Illinois        35108   1577
##  9 2021-02-23 Guam             8696    131
## 10 2020-04-24 Idaho            1870     54
```

```
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 2022-01-30 California 8287535  80038
##  2 2022-01-29 California 8270373  80004
##  3 2022-01-28 California 8248681  79934
##  4 2022-01-27 California 8172855  79643
##  5 2022-01-26 California 8055410  79353
##  6 2022-01-25 California 7984924  79118
##  7 2022-01-24 California 7904252  79001
##  8 2022-01-23 California 7688422  78839
##  9 2022-01-22 California 7660930  78775
## 10 2022-01-21 California 7621774  78700
```

```
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 date

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 2022-01-25 Oregon  597172   5994         0.0100       6902
## 2 2022-01-26 Oregon  605363   6048         0.00999      8191
## 3 2022-01-27 Oregon  613221   6067         0.00989      7858
## 4 2022-01-28 Oregon  613221   6067         0.00989         0
## 5 2022-01-29 Oregon  613221   6067         0.00989         0
## 6 2022-01-30 Oregon  620653   6086         0.00981      7432
```

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       7669.          2721      80038
```

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 2021-02-15     NA 438431.  7669.          0.0209        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 2021-02-15     NA 438431.  7669.          0.0209     1926.        22
```

Tabular data

Grouping

Often we want to group our data before summarizing. What do these two examples 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                                   0.0197
##  2 Alaska                                    0.00678
##  3 American Samoa                            0
##  4 Arizona                                   0.0206
##  5 Arkansas                                  0.0157
##  6 California                                0.0174
##  7 Colorado                                  0.0214
##  8 Connecticut                               0.0426
##  9 Delaware                                  0.0213
```

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(cases)) %>%
    ungroup()
```

```
## # A tibble: 38,590 x 8
##    date       state cases deaths deaths_per_case new_cases new_death
##    <date>     <chr> <dbl>  <dbl>           <dbl>     <dbl>     <dbl
##  1 2020-01-21 Wash~     1      0               0         0
##  2 2020-01-22 Wash~     1      0               0         0
##  3 2020-01-23 Wash~     1      0               0         0
##  4 2020-01-24 Illi~     1      0               0         0
##  5 2020-01-24 Wash~     1      0               0         0
##  6 2020-01-25 Cali~     1      0               0         0
##  7 2020-01-25 Illi~     1      0               0         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/CivilServiceUSA/us-g
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: 2022-01-31 15:28
##   Status: 200
##   Content-Type: application/vnd.openxmlformats-officedocument.spread
##   Size: 18.1 kB
## <ON DISK>  /var/folders/by/t5qdf0996h12f6ngxhxrqpf40000gs/T//RtmpaCW
```

```
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()
```

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
```

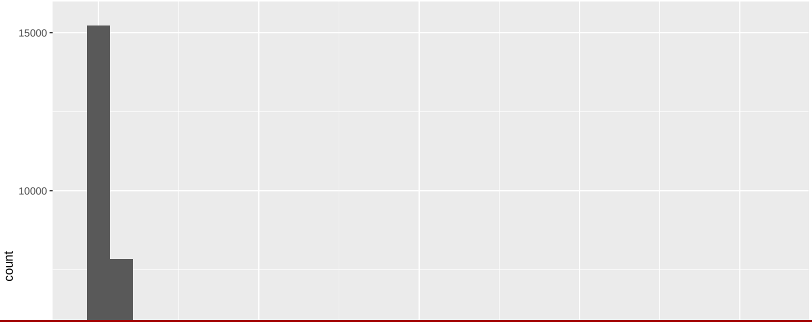
```
## [1] 50
```

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
```

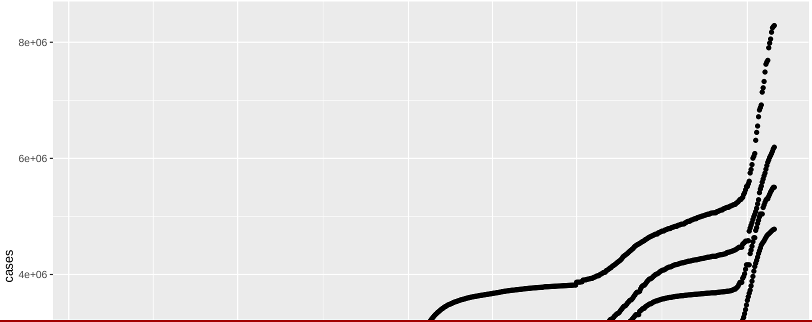


Data visualization

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
```

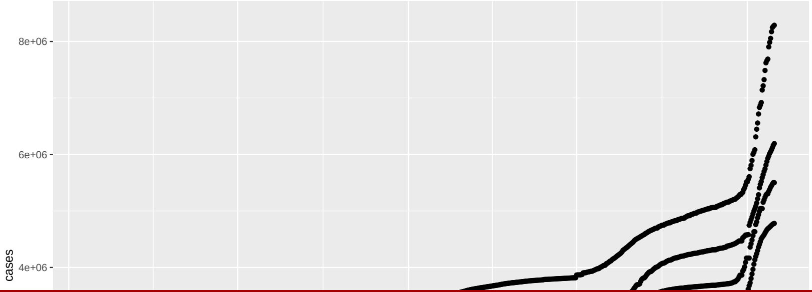


Data visualization

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
```

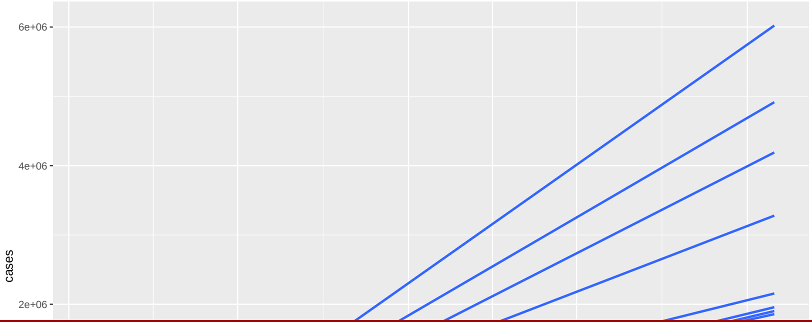


Data visualization

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
```

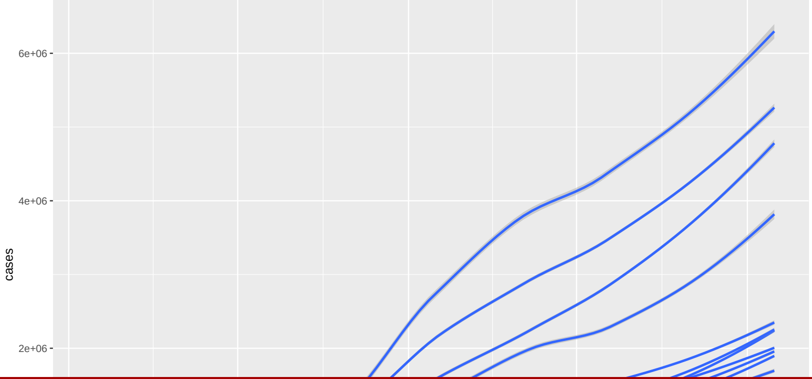


Data visualization

ggplot2

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

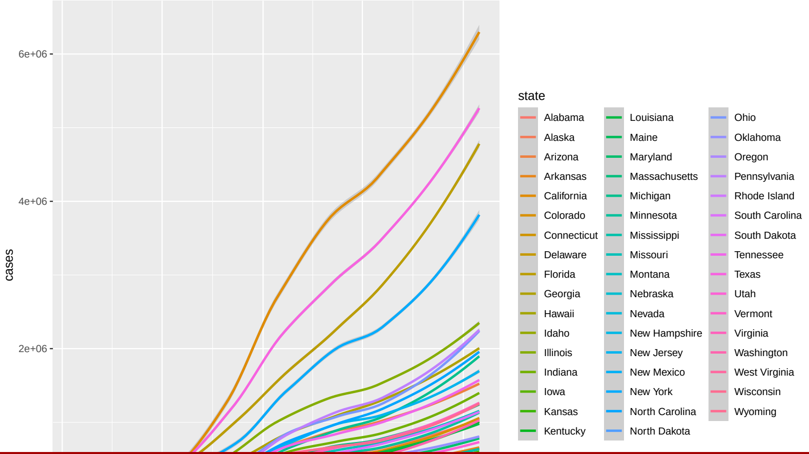
```
ggplot(c19, # data
       aes(x = date, y= cases, group=state)) + # aesthetic mapping
    geom_smooth(method='loess') # plot type
```



Data visualization

ggplot2

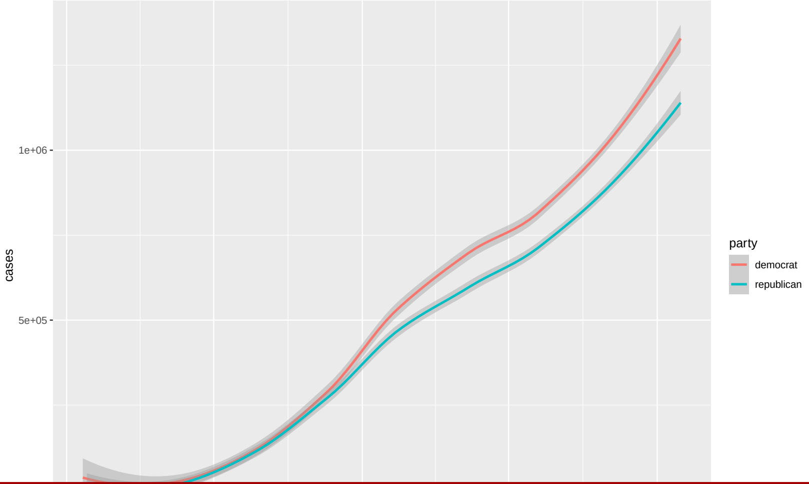
The color parameter allows us to assign a different color to each line. Note how things get a little difficult to read now.



Data visualization

ggplot2

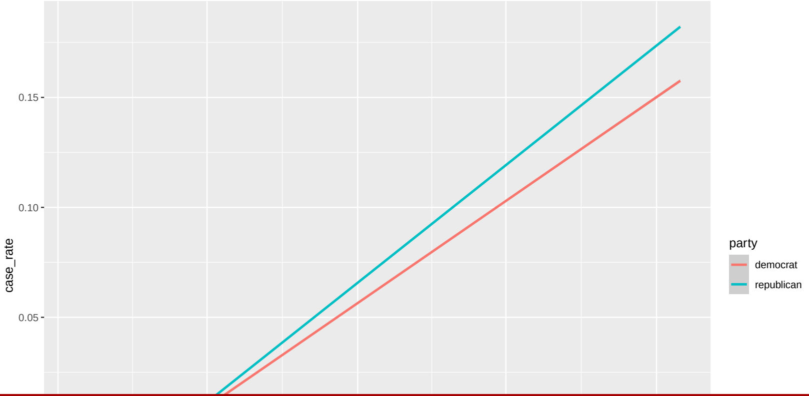
We can easily group by other variables.



Data visualization

ggplot2

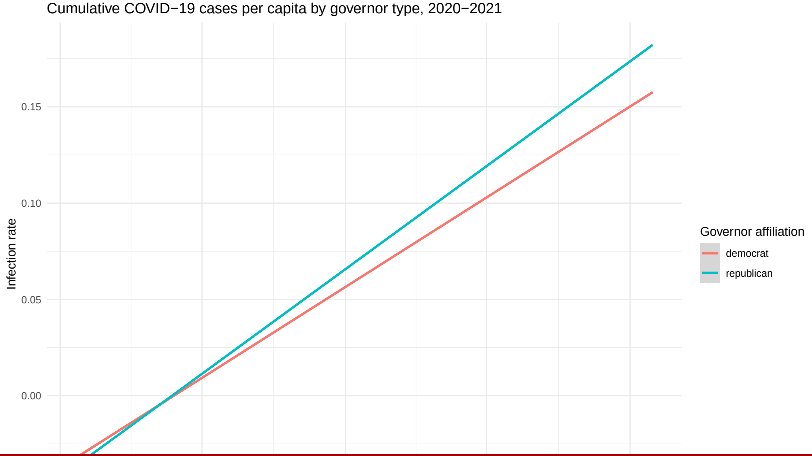
Why might the previous plot be misleading? Is there a better way to look at how cases vary by partisanship of the governor? Note: The plot is now rendered as an object p before plotting. This allows us to modify it later on.



Data visualization

ggplot2

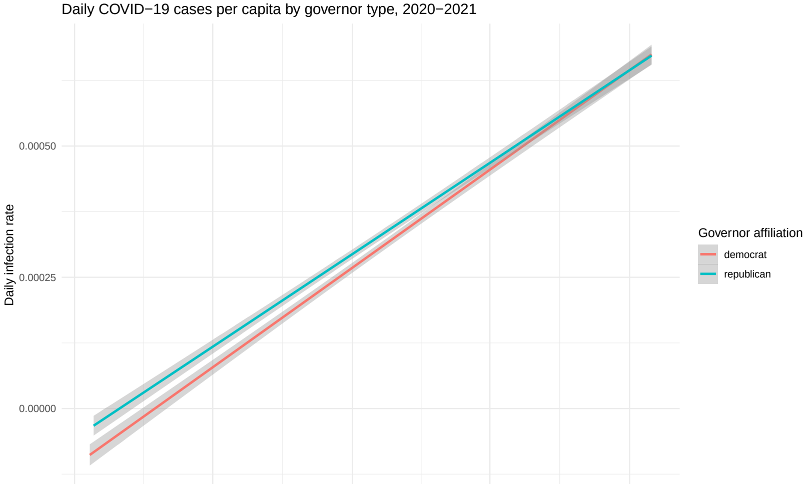
Now we have a plot, let's
make it look a bit nicer. We can easily add labels and modify the axes.



Data visualization

ggplot2

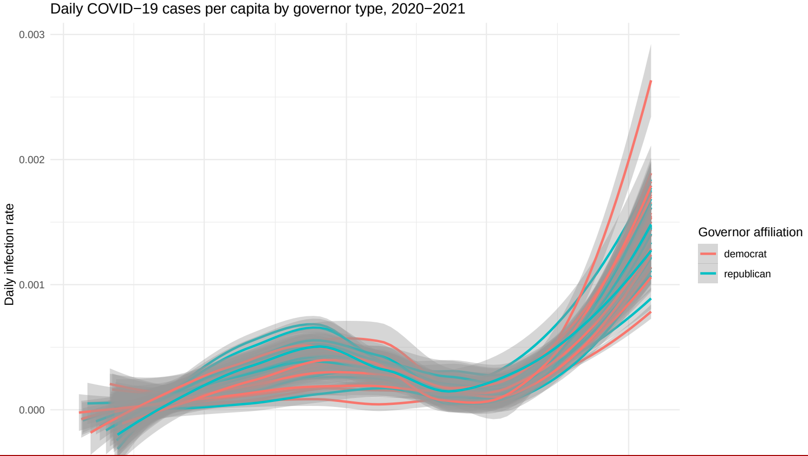
We can easily modify this code to look at the data in a different way.



Data visualization

ggplot2

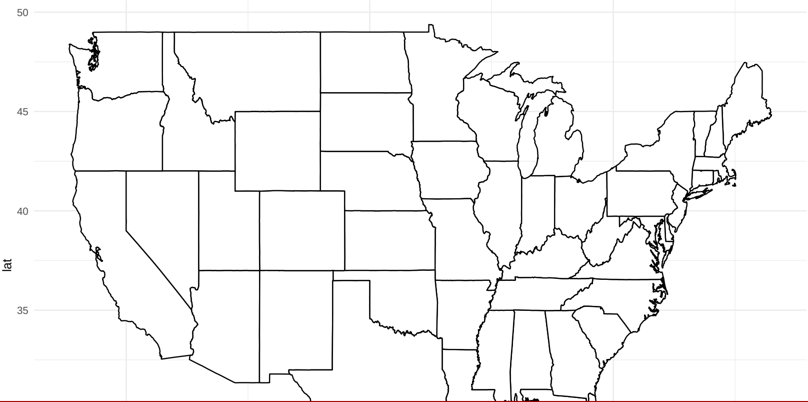
What could we change to include separate lines for each state while maintaining the color.



Data visualization

ggplot2 maps

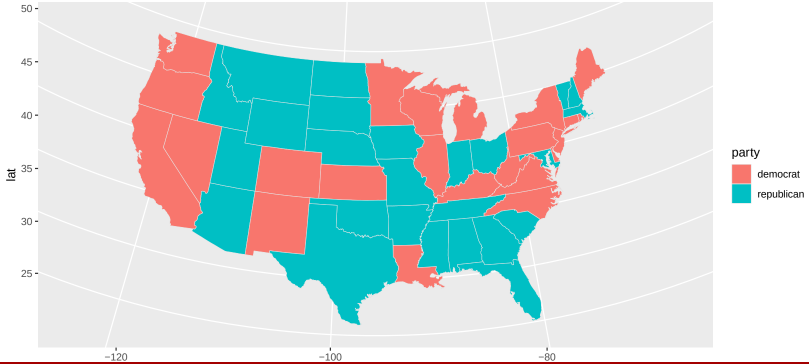
The `ggplot` package can be used to produce many different types of visualizations. For example, we can use it to produce maps. Here we load the package `maps` to get the shapefile for each state. The example



Data visualization

ggplot2

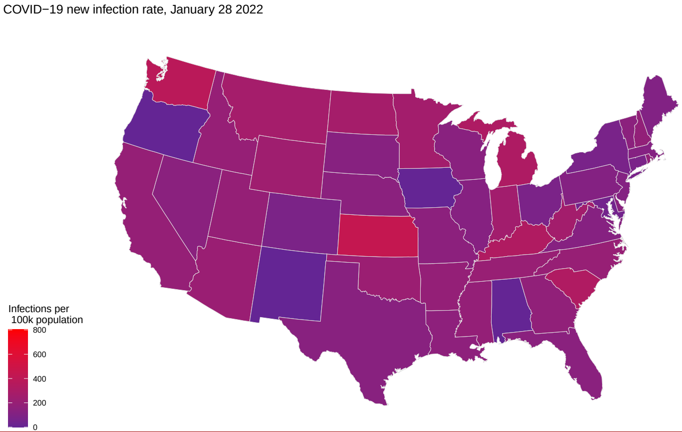
We have to merge our data with the shapefile in order to plot it on the map.



Data visualization

ggplot2

Let's try to do something more interesting.



Some very preliminary data science

What predicts the state-level daily infection rate?

We can use linear regression to predict the number of new cases given information about the state.

```
summary(lm(new_cases ~ new_cases.lag + pop + party + pop * party + as.n
    data = c19))
```

```
##
## Call:
## lm(formula = new_cases ~ new_cases.lag + pop + party + pop *
##     party + as.numeric(date), data = c19)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -47668  -1442   -338    829 212871
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)         -9.973e+04  2.779e+03 -35.884   <2e-16 ***
## new_cases.lag        1.869e-01  4.861e-03  38.456   <2e-16 ***
```

Some very preliminary data science

What predicts the state-level daily infection rate?

We can use linear regression to predict the number of new cases given information about the state.

```
summary(lm(new_cases ~ new_cases.lag + pop + party + pop * party + as.n
    data = c19 %>%
        filter(date <= as.Date("2020-2-29"))))
```

```
##
## Call:
## lm(formula = new_cases ~ new_cases.lag + pop + party + pop *
##     party + as.numeric(date), data = c19 %>% filter(date <= as.Date(
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.1631 -0.2763 -0.1020  0.0842 14.0120
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)         -3.191e+02  1.329e+02  -2.402  0.01711 *
```

Some very preliminary data science

What predicts the state-level daily infection rate?

We can use linear regression to predict the number of new cases given information about the state.

```
summary(lm(new_cases ~ new_cases.lag + pop + party + pop * party + as.n
    data = c19 %>%
        filter(date <= as.Date("2020-12-31"))))
```

```
##
## Call:
## lm(formula = new_cases ~ new_cases.lag + pop + party + pop *
##     party + as.numeric(date), data = c19 %>% filter(date <= as.Date(
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  -9491   -813   -148    548  56385
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)         -2.026e+05  3.785e+03 -53.535  < 2e-16 ***
```

Some very preliminary data science

What predicts the state-level daily infection rate?

We can use linear regression to predict the number of new cases given information about the state.

```
summary(lm(new_cases ~ new_cases.lag + pop + party + pop * party + as.n
    data = c19 %>%
        filter(date > as.Date("2020-12-31"))))
```

```
##
## Call:
## lm(formula = new_cases ~ new_cases.lag + pop + party + pop *
##     party + as.numeric(date), data = c19 %>% filter(date > as.Date("
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -48637  -1876   -409    863 208723
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)         -2.050e+05  8.347e+03 -24.559   <2e-16 ***
```

Next lecture

- File management
- Github