Social Data Science

Tabular data and visualization

Dr. Thomas Davidson

Rutgers University

September 15, 2021

Plan

- Recap on programming fundamentals
- 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-10 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
glimpse(c19)
```

```
## Rows: 30,869
## 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,
```

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: 30,869 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: 24,302 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 2021-08-08 Arkansas        404277   6301
##  2 2021-03-03 Texas          2673115  44627
##  3 2021-09-14 North Carolina 1310185  15322
##  4 2021-04-02 Pennsylvania   1038349  25206
##  5 2020-04-22 Kentucky          3373    185
##  6 2021-03-03 Arizona         820561  16089
##  7 2020-06-24 Pennsylvania     87775   6568
##  8 2020-07-30 Nevada           47034    805
##  9 2020-08-26 Virgin Islands    1030     14
## 10 2020-01-31 Washington           1      0
```

```
sample_frac(c19, 0.01)  # Randomly pick a fraction 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 2021-09-14 California 4608094  67422
##  2 2021-09-13 California 4599348  67312
##  3 2021-09-12 California 4583292  67199
##  4 2021-09-11 California 4575900  67149
##  5 2021-09-10 California 4566746  67062
##  6 2021-09-09 California 4556775  66897
##  7 2021-09-08 California 4545832  66722
##  8 2021-09-07 California 4536771  66569
##  9 2021-09-06 California 4525630  66496
## 10 2021-09-05 California 4518576  66481
```

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

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

How can we recover the new cases and deaths from the cumulative data using mutate? Could this allow us to see the daily case rate?

```
c19 <- c19 %>%
    group_by(state) %>%
    mutate(new_cases = cases - lag(cases), new_deaths = deaths - lag(de
glimpse(c19)
```

```
## Rows: 30,869
## Columns: 7
## Groups: state [55]
## $ date            <date> 2020-01-21, 2020-01-22, 2020-01-23, 2020-01
## $ state           <chr> "Washington", "Washington", "Washington", "I
## $ cases           <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1,
## $ deaths          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## $ deaths_per_case <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## $ new_cases       <dbl> NA, 0, 0, NA, 0, NA, 0, 0, NA, 1, 0, 0, 0, 0
## $ new_deaths      <dbl> NA, 0, 0, NA, 0, NA, 0, 0, NA, 0, 0, 0, 0, 0
```

Tabular data

Summarizing

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

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

```
## # A tibble: 55 x 4
##    state                mean_deaths median_deaths max_deaths
##    <chr>                      <dbl>         <dbl>      <dbl>
##  1 Alabama                    5785.          4102      12718
##  2 Alaska                      176.           169        449
##  3 Arizona                    8882.          6406.      19304
##  4 Arkansas                   3162.          2911        7334
##  5 California                29640.         18555       67422
##  6 Colorado                   3950.          3671        7498
##  7 Connecticut                5658.          5345        8446
##  8 Delaware                    986.           815        1902
```

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.

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

```
## # A tibble: 55 x 7
##    state      date       cases deaths deaths_per_case new
##    <chr>      <date>     <dbl>  <dbl>           <dbl>
##  1 Alabama    2021-09-14 7.54e5  12718          0.0426
##  2 Alaska     2021-09-14 9.66e4    449          0.0247
##  3 Arizona    2021-09-14 1.05e6  19304          0.0501
##  4 Arkansas   2021-09-14 4.77e5   7334          0.0241
##  5 California 2021-09-14 4.61e6  67422          0.0414
##  6 Colorado   2021-09-14 6.46e5   7498          0.0571
##  7 Connecticut 2021-09-14 3.81e5   8446          0.0935
##  8 Delaware   2021-09-14 1.26e5   1902          0.0465
```

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))  # mean deaths per case by state
```

```
## # A tibble: 55 x 2
##    state                `mean(deaths_per_case)`
##    <chr>                                  <dbl>
##  1 Alabama                              0.0201
##  2 Alaska                               0.00713
##  3 Arizona                              0.0213
##  4 Arkansas                             0.0157
##  5 California                           0.0182
##  6 Colorado                             0.0239
##  7 Connecticut                          0.0483
##  8 Delaware                             0.0233
##  9 District of Columbia                 0.0319
```

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.

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

```
## # A tibble: 30,869 x 8
##    date       state      cases deaths deaths_per_case new_cases new_
##    <date>     <chr>      <dbl>  <dbl>           <dbl>     <dbl>
##  1 2020-01-21 Washington     1      0               0        NA
##  2 2020-01-22 Washington     1      0               0         0
##  3 2020-01-23 Washington     1      0               0         0
##  4 2020-01-24 Illinois       1      0               0        NA
##  5 2020-01-24 Washington     1      0               0         0
##  6 2020-01-25 California     1      0               0        NA
##  7 2020-01-25 Illinois       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 governors and joining it on state.

```
gov <- read_csv("https://raw.githubusercontent.com/CivilServiceUSA/us-g
gov <- gov %>%
    select(state_name, party)

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.

```
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: 2021-09-15 18:28
##   Status: 200
##   Content-Type: application/vnd.openxmlformats-officedocument.spread
##   Size: 18.1 kB
## <ON DISK>  /var/folders/by/t5qdf0996h12f6ngxhxrqpf40000gs/T//Rtmpk94
```

```
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.

```
c19 <- c19 %>%
    left_join(pop.states, by = "state")
```

Data visualization

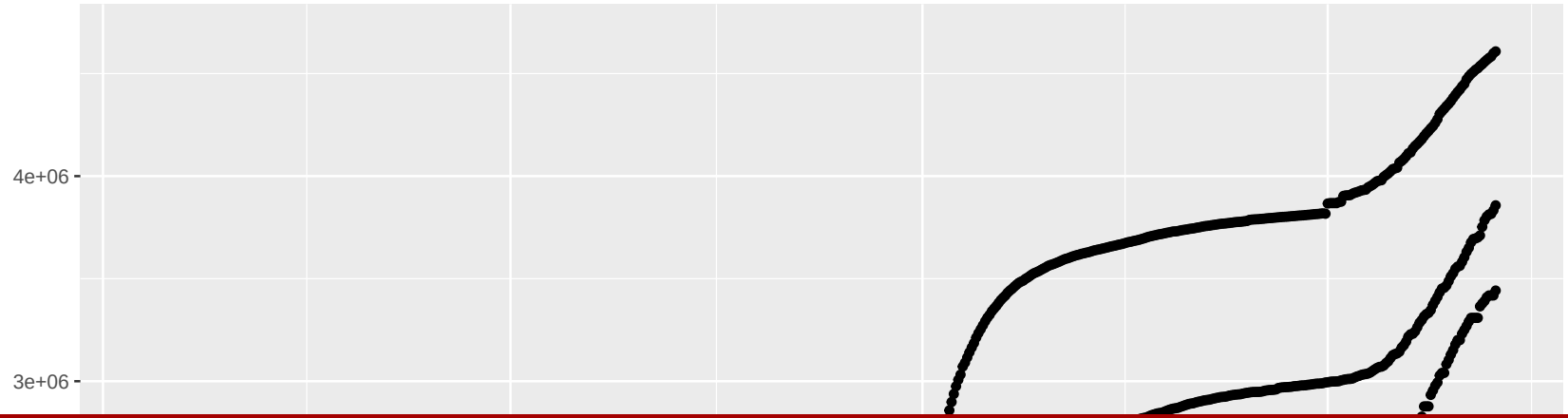
ggplot2

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.

```
c19 <- c19 %>% drop_na(party) # Dropping any row not considered a state
length(unique(c19$state)) # Verifying the correct number of states
```

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

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

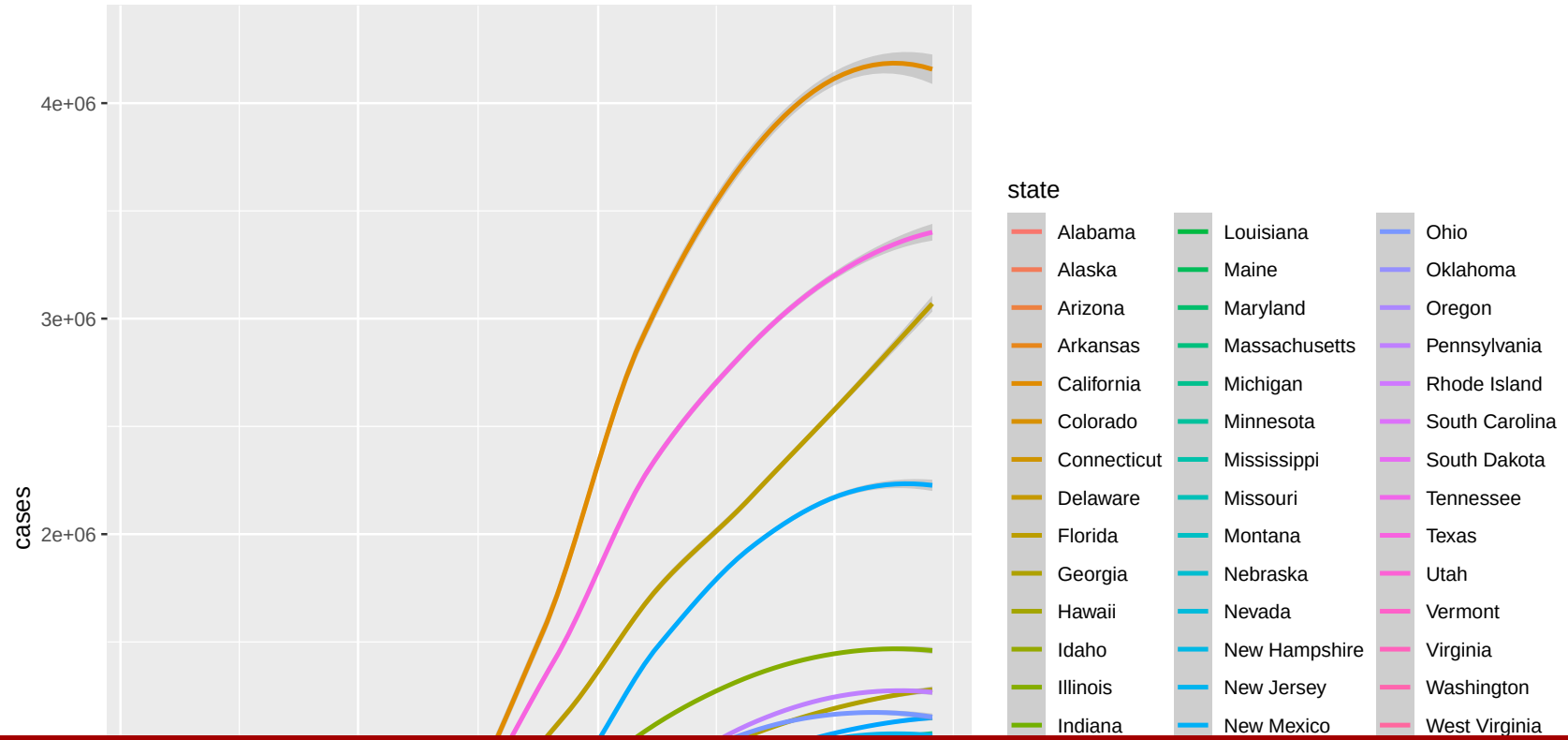


Data visualization

ggplot2

The color parameter allows us to assign a different color to each line.

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

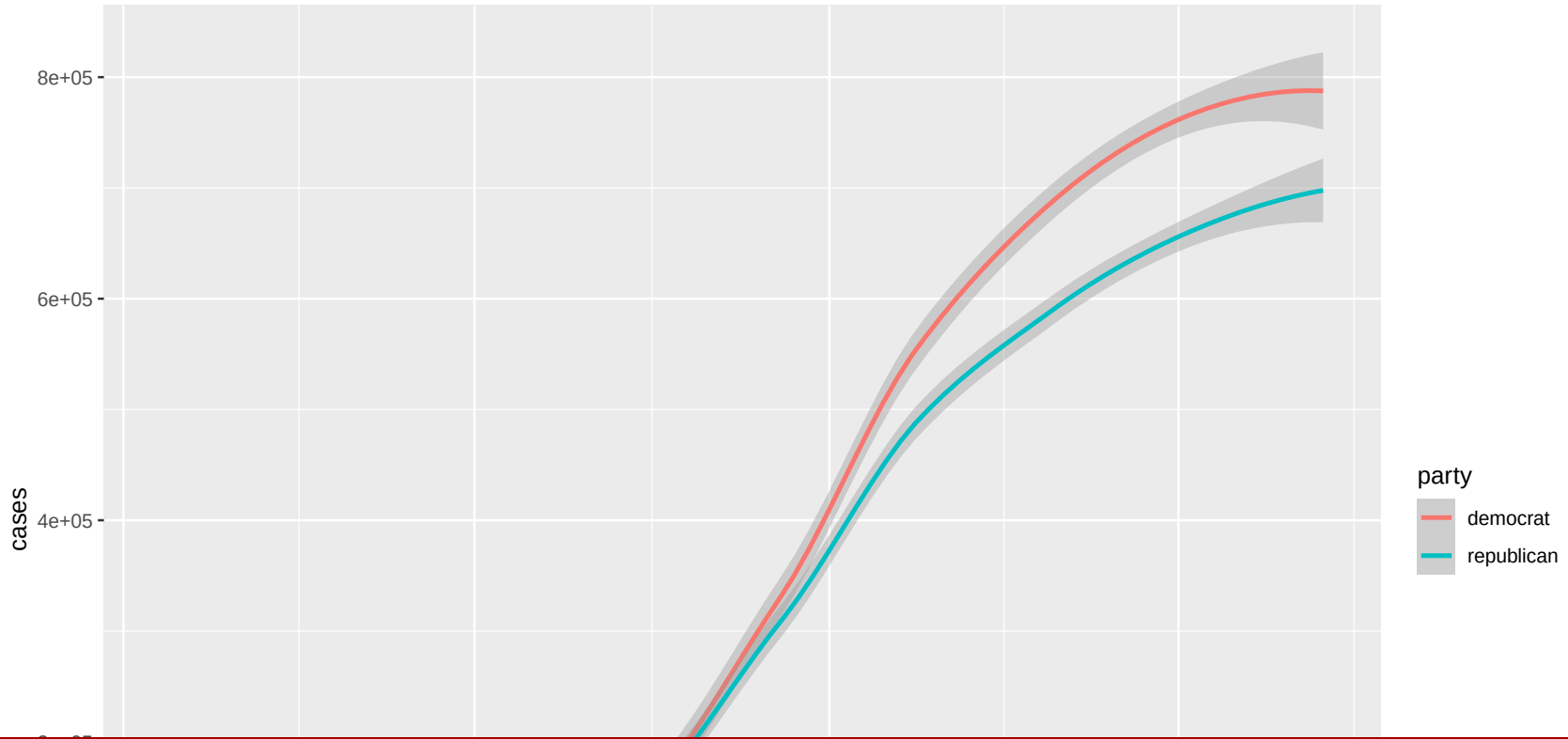


Data visualization

ggplot2

We can easily group by other variables.

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

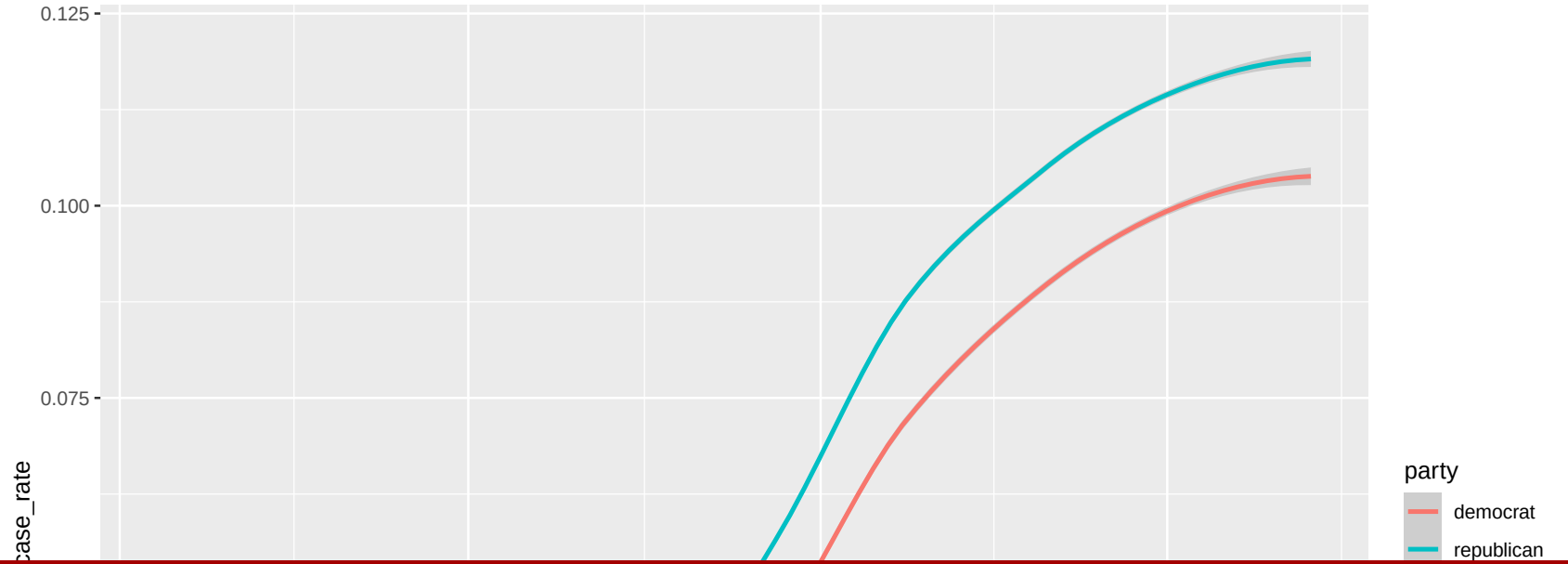


Data visualization

ggplot2

Why might the previous plot be misleading? Is there a better way to look at how cases vary by partisanship of the governor?

```
c19 <- c19 %>% mutate(case_rate = cases / pop)
p <- ggplot(c19, # data
       aes(x = date, y= case_rate, group=party, color=party)) + # aesth
    geom_smooth(method='loess') # plot type
p
```

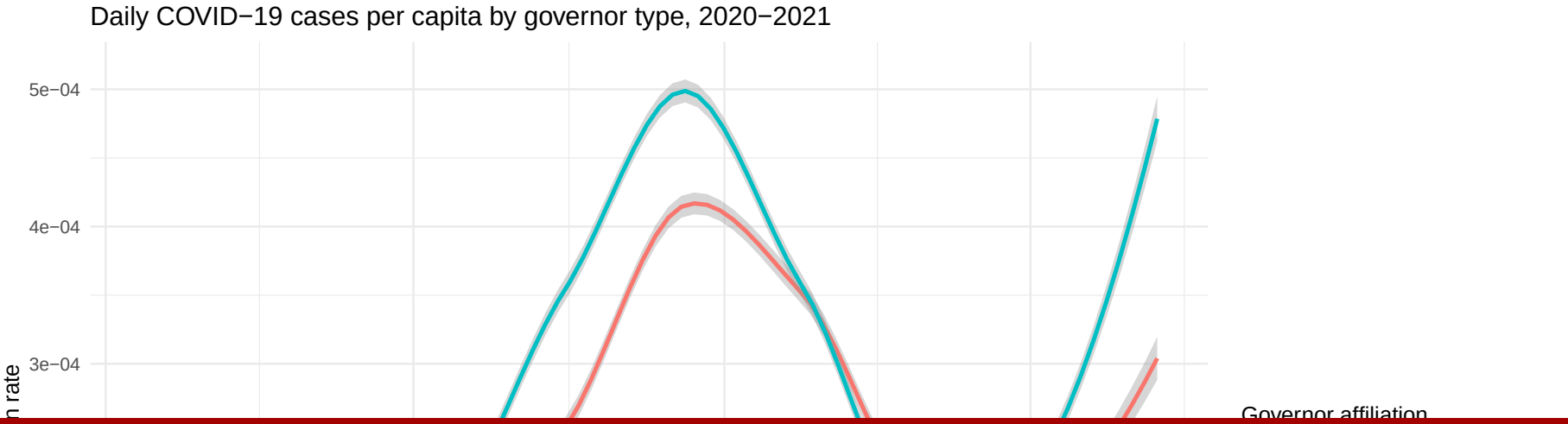


Data visualization

ggplot2

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

```
c19 <- c19 %>% mutate(daily_case_rate = new_cases / pop)
ggplot(c19, # data
       aes(x = date, y= daily_case_rate, group=party, color=party)) + #
    geom_smooth(method='loess') + # plot type
theme_minimal() + # themes change the overall look of a plot
    labs(x = "", y = "Daily infection rate", title = "Daily COVID-19 ca
         color = "Governor affiliation", caption = "COVID-19 data from
    theme(axis.text.x = element_text(angle = 90))
```



Data visualization

ggplot2

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

```
# install.packages('maps')
library(maps)
us_states <- map_data("state")

# We can plot an empty map
ggplot(data = us_states, mapping = aes(x = long, y = lat, group = group
    color = "black") + theme_minimal()
```

The code for this example is based on Chapter 7 of Kieran Healy's *Data Visualization*

Data visualization

ggplot2

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

```
c19.map <- c19 %>%
    mutate(state_lower = tolower(state)) %>%
    left_join(us_states, by = c(state_lower = "region"))
glimpse(c19.map)

p <- ggplot(data = c19.map, aes(x = long, y = lat, group = group, fill

p + geom_polygon(color = "gray90", size = 0.1) + coord_map(projection =
    lat0 = 39, lat1 = 45)
```

Data visualization

ggplot2

Let's try to do something more interesting.

```
#install.packages("ggthemes")
library(ggthemes)

c19.map <- c19.map %>% mutate(cases_per_100k = new_cases / (pop/100000))

p <- ggplot(data = c19.map %>% filter(date == as.Date("2021-09-14")),
            aes(x = long, y = lat,
                group = group, fill = cases_per_100k))

p + geom_polygon(color = "gray90", size = 0.1) +
    coord_map(projection = "albers", lat0 = 39, lat1 = 45) +
  scale_fill_gradient2(low = "blue", # Determines the color scale
                                mid = scales::muted("purple"),
                                high = "red") +
  theme_map() + # A theme for making maps
  labs(title = "COVID-19 new infection rate, September 14th 2021",
```

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. What do the results reveal? Which model is more trustworthy?

```
c19$new_cases.lag <- lag(c19$new_cases)
summary(lm(new_cases ~ new_cases.lag + pop + party + as.numeric(date),
```

```
##
## Call:
## lm(formula = new_cases ~ new_cases.lag + pop + party + as.numeric(da
##     data = c19)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -45570   -795   -206    347  56478
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)      -4.109e+04  1.837e+03 -22.369   <2e-16 ***
```

Github

Overview

- Github is a version-control system
  - This allows you to easily control and manage changes to your code (similar to Track Changes in Word)
  - It can facilitate collaboration
  - Version-control helps to ensure reproducibility
  - It makes it easy to share code
- Github is *not* designed as a place to store large datasets (100Mb file size limit)

Github

Terminology

- A Github *repository* (or *repo* for short) contains all files and associated history
  - A repository can be public or private
  - Files should be organized into folders
  - Github can render Markdown files (suffix `.md` in Markdown), useful for documentation
- Github repositories exist online and you can *clone* them to your local computer

Github

Using Github

- You can interact with Github in several different ways
  - Github Desktop (recommended)
  - Through your browser (not recommended)
  - Using the command line
  - RStudio integration
    - See https://happygitwithr.com/index.html for a guide

Github

Basic commands

- Let's say you want to make changes to a repository, in this case adding a single file called `myfile.txt`:

1. Make changes to `myfile.txt` and save the file.
2. `git status` will show information about the status of your repo.
3. `git add myfile.txt` will stage the file to be added to the online repo.
   - Avoid using `git add *`
4. `git commit -m "Adding a new file"` commits the file to the repo, along with an informative message.

Github

Basic commands

5. `git push origin main` then tells Github to push the local changes to the main branch of the online repository
   - Conversely, `git pull origin main` will pull the latest updates from your main branch to your local machine
6. Now visit the web page for your repository and you should see the changes.

Github

Viewing commit histories

- You can view the history of a given file by looking at the commits
  - e.g. Let's look at the syllabus for this course https://github.com/t-davidson/social-data-science-fall-2021/commits/main/syllabus.Rmd

Github

Branches and merging

- A *branch* consists of a particular version of the repo
  - All repos start with a single branch called *main* (formerly *master*)
  - You can create separate branches for particular tasks
    - This is particularly useful for collaboration
  - You can then *merge* the branch back into main
    - But be careful of *merge conflicts*
- A *pull request* is a mechanism for merging content into a repository
  - This can enable the code to be reviewed before it is integrated
- The *issue* function can be used to note any issues with the code and to bring them to the repo owner's attention (e.g. https://github.com/tidyverse/ggplot2)

Github

Forks

- A *fork* is a copy of another repository (usually from another user)
  - This allows you to easily copy the repository and modify it without changing the original content

Github

Classroom

- We will be using a tool called *Github Classroom* for the homework assignments
  - You will receive a special template repository containing the homework
  - The submission will occur when you push the final commits to Github
    - Further instructions will be included

Github

Student Developer Pack

- If you haven't already, log in and apply for the Github Student Developer pack
  - https://education.github.com/pack
- This allows you to make unlimited private repositories and gives access to many other tools

Questions?