15: Analysis of COVID-19 data

Graphical Analyis of Biological Data

This is the *final* assignment. It serves to challenge your ability to apply skills you learned throughout the course. You may need to review the notes, assignments, textbook, and your previously written code to complete the tasks.

You will use four data sets to reproduce five graphs and one table related to the incidence of COVID-19. The graphs and table are shown below for you to serve as a guide. You are to come as close as possible. Pay attention to details such as axis labels, etc.

I will provide minimal guidance, both here and in response to questions. I want you to apply what you have learned and think independently, not do what I tell you to do. *You have to make decisions.* I do provide some specific code or information for you to incorporate for certain graphs to help you match the example plot.

Preparation

- Right-click and download hw15_data.zip. Unzip the file and move all of the data files into your data folder. The COVID-19 data are from USA Facts.
  - `covid_confirmed_usafacts.csv`: This file lists the accumulated daily total number of confirmed covid cases for each county of each state in the U.S. since the start of the pandemic.
  - `covid_deaths_usafacts.csv`: This file lists the accumulated daily total number of deaths attributed to covid for each county of each state in the U.S. since the start of the pandemic.
  - `covid_county_population_usafacts.csv`: This file lists the population size of each county of each state in the U.S.
  - `semo_county_enrollment.csv`: This file contains data from the SEMO Fact Book with the number of Missouri students enrolled at Southeast, broken down by Missouri county. Unfortunately, a county breakdown for other states is not available. *You only need these data for Plot 2.*
- Create an `hw15` folder inside the same folder as your project file.
- Create a new R *script* and save it as `lastname_hw15.R` in your `hw15` folder. Do not create a notebook (.Rmd) file. All of your code will be in the script.
- Install the `data.table` package.

Your script

Use section dividers (`Code > Insert Section...`, or cmd/ctrl + shift + r) as appropriate to divde your R script into the following sections.

1. Code to load your libraries. The libraries you will need are

- `tidyverse`
- `here`
- `lubridate`

- `sf`
- `patchwork`
- `gghighlight`
- `ggthemes` (optional)

  Do not load the `data.table` package. I will give you the code to use when needed.

2. Define constants; Constants are values you set and are not changed by your R code. Constants you should use, but are not necessarily limited to, include

- First US case: 19 Jan 2020,
- First MO case: 08 Mar 2020,
- The lower 48 from the mapping exercise,
- CDC regions (see below), and
- anything else you feel is appropriate.

  Note: The first COVID-19 case in Missouri was reported 07 March 2020 but does not appear in these data until 08 March. A "constant variable" allows you to use the same value repeatedly in your code. If you have to later change the constant, you change it only once. *I might change your constants when I run your code to ensure you used them as expected.*

3. Functions; write the code for your functions here. Do not include your functions in the `my_functions.R` script you wrote in an earlier exercise. You will be required below to write one function although there are opportunities for others.

4. importing, wrangling, graphs. You have flexibility here. Use a programming structure that seems logical to you. However, most data you can import once, then wrangle the data as needed for each plot. As an example, my code has a section for the initial import, wrangling, and merging, then a section for each plot with additional wrangling and graphing.

Import and wrangling guidelines

You can import the data *once* and merge the sets together via `left_join()` to create a master data set. You can then wrangle the master file as necessary for the five plots.

I found it easiest to start with a fresh import of data for plot 3. Otherwise, import just once. You may discover that you don't have to reimport for that plot.

You should be able to figure out what `tidyverse` functions are necessary and when.

After you import `covid_confirmed_usafacts` and `covid_deaths_usafacts.csv`, remove rows with FIPS code equal to zero. They are not necessary and will cause problems down the road. You may have to remove other rows as needed for individual plots. You'll have to wrangle as necessary.

Use `lubridate` to convert the date column (that was a hint) to ISO 8601 format.

Remove all dates before the first reported U.S. COVID-19 case.

If you find yourself writing the same code repeatedly, restructure you code so that you don't have to do it as often, preferably just once.

There is no single correct solution to these exercises. If your graphs match closely with those shown below, you are probably correct. Use the end result to guide your process. Study each figure, think logially about what you will have to do, then do it. You can use pseudocode to get yourself started.

Plots

Plot 1

This plot is a summary of the cumulative number of confirmed cases and deaths for the four CDC regions.

You will need to group states according to the four CDC regions (follow the link to get the regions). You can add these to your "master" tibble or add it to the tibble that you use for this plot. You only need these regions for the first plot. You do not need to encode the divisions.

- Northeast
- South
- Midwest
- West (inc. Alaska and Hawaii)

Requirements *Match the plot as closely as possible.* Pay attention to details.

Reduce your data set to dates greater than or equal to the first Missouri COVID case. That will make each plot a little narrower to fit side-by-side a little better without significantly affecting the display of the data.

Place the graphs side by side with confirmed cases on left, deaths on right.

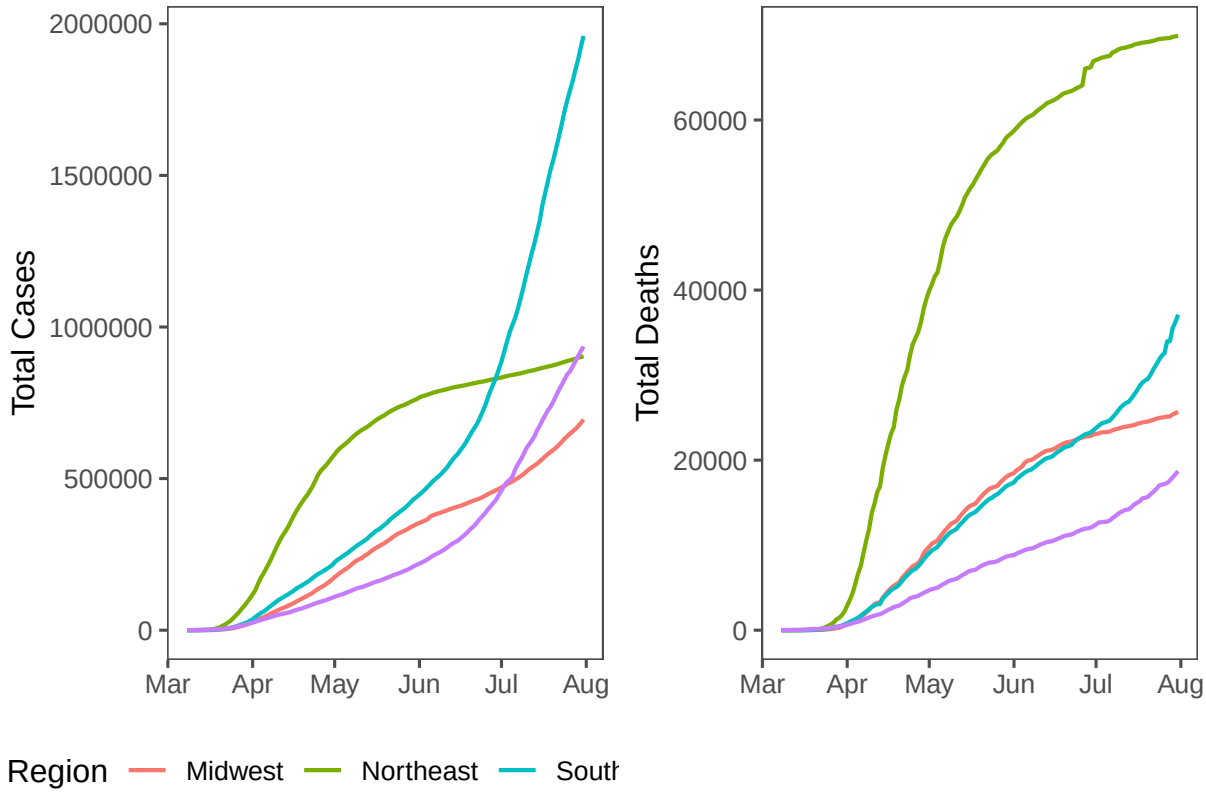
Use a line size between 0.5 and 1, something that you feel is easy to see without being too heavy.

If neccesary, add a `scale_x_date()` layer to your plot to set date format to three letter abbreviations e.g., Jun, Jul). R's help has specific examples on how to use this function.

Do not include a label for x-axis. Saying "Date" is somewhat redundant when dates are obvious.

Use a theme that makes for a clean plot. Try themes that come with `ggplot2` or from `ggthemes`.

It's okay if your legend appears cropped like the example although I don't think it will. What's most important is that you have only one legend.



Plot 1

Plot 2: Highlight Missouri Counties with 200+ students at SEMO

Show the cumulative total cases of COVID-19 for Missouri counties. Highlight the counties that have 200 or more students attending SEMO in 2019. 200 students is approximately 2.5% of the total undergraduate student body.

Requirements Reduce your total confirmed cases data to Missouri and for all date greater than or equal to the first Missouri COVID-19 case.

Use string functions to make the following changes to the county names in the confirmed cases data:

- Remove " County" from all county names; e.g., “Perry County” becomes “Perry”.
- Change “Jackson County (including other portions of Kansas City)” to “Jackson”. Be careful here. The original string has parentheses that must escaped to be recognized.

Import the data from `semo_counties.csv` and reduce it to the needed columns.

Use string functions to make the following changes to the SEMO counties data.

- “De Kalb” to “DeKalb”
- “Sainte” to “Ste.” (for Sainte Genevieve County)
- “Saint” to “St.”
- “Saint Louis City” to “City of St. Louis”

  Think carefully about the order you make the changes to the county names or some changes may not work as intended.

Use `left_join` to merge the two data files together by county name. Recall that the columns in the two data sets must have the exact same column name to work properly.

Use `scale_x_date` to set date format to `dd mmm` or `mmm dd` (e.g., 01 Jun or Jun 01). Use three letter abbreviations for the month.

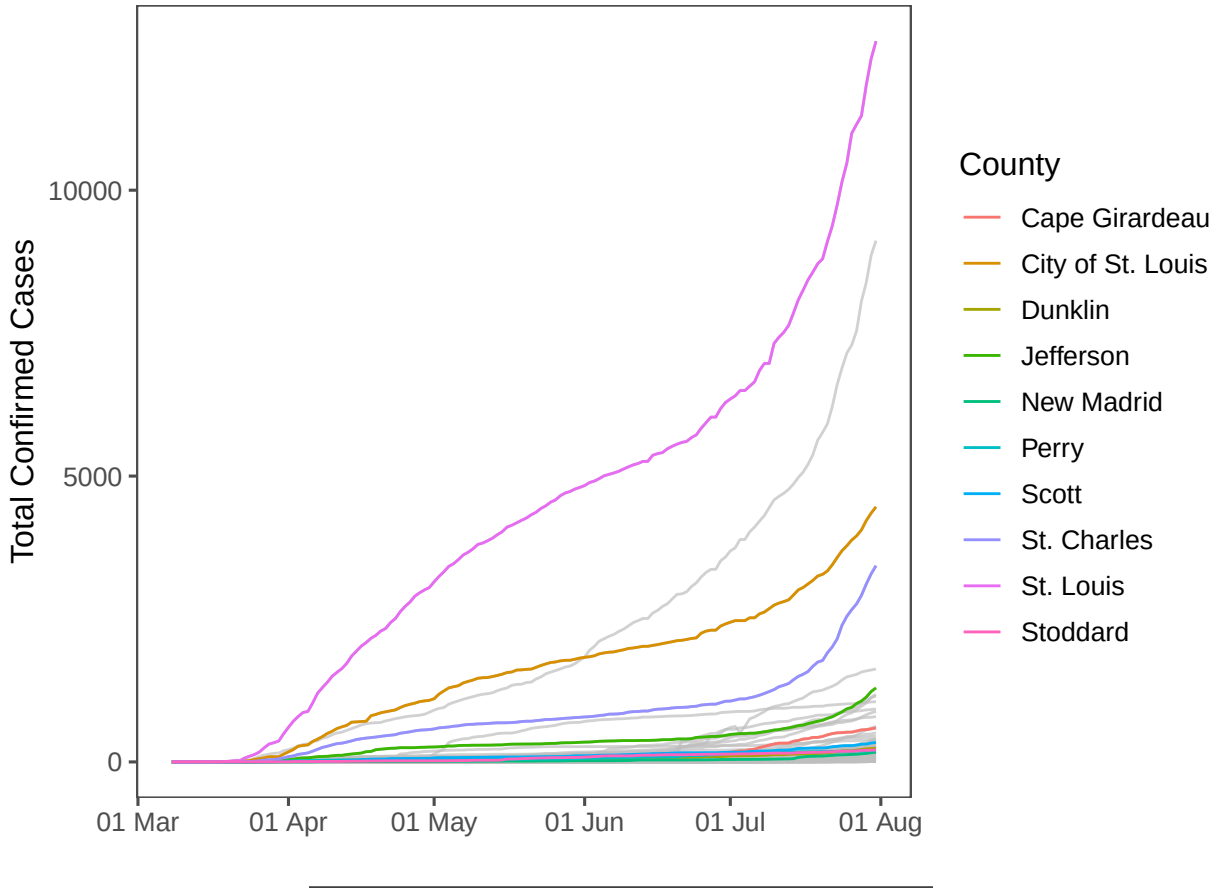
Use `gghighlight()` to highlight counties with 200 or more students attending SEMO and turn off direct labelling so the legend appears. You must color by county name in your `geom_line()` mapping for this to work.

It's okay if you get these warning messages shown above the figure.

Plot 2

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Warning: Tried to calculate with group_by(), but the calculation failed.
## Falling back to ungrouped filter operation...
```



Plot 3: Cleveland plot comparing number of cases in April and July

This plot shows the difference in the number of cases for April (squares) and July (circles) 2020.

I found it easiest to do a fresh import of the data to obtain the values needed for this plot. Whether you do depends on how much initial wrangling you did for your initial import.

Requirements Calculate the total number of new cases for April 1 to April 30 and the total number of new cases for July 1 to July 30. Do not include 31 July so that you compare the same number of days from both months.

The per capita case rate is cases divided by population size for each county, multiplied by 100,000 to standardize the rate to "per 100,000" people.

Sort the states by the number of July cases.

Choose a suitable point size, not too large but not too small. You have to decide what you think is visually appropriate.

Change the default `gplot2` colors. I installed the `RColorBrewer` package and created a color palette with the first three colors of the `Dark2` palette. I used the first two colors from the vector. You can do that or you can use a different, suitable palette.

You may use whatever shapes you want (including filled shapes). Your axis label must reflect the shapes you use.

I used `theme_minimal()` because of the guiding light gray grid lines and lack of a bounding box but you can try other themes.

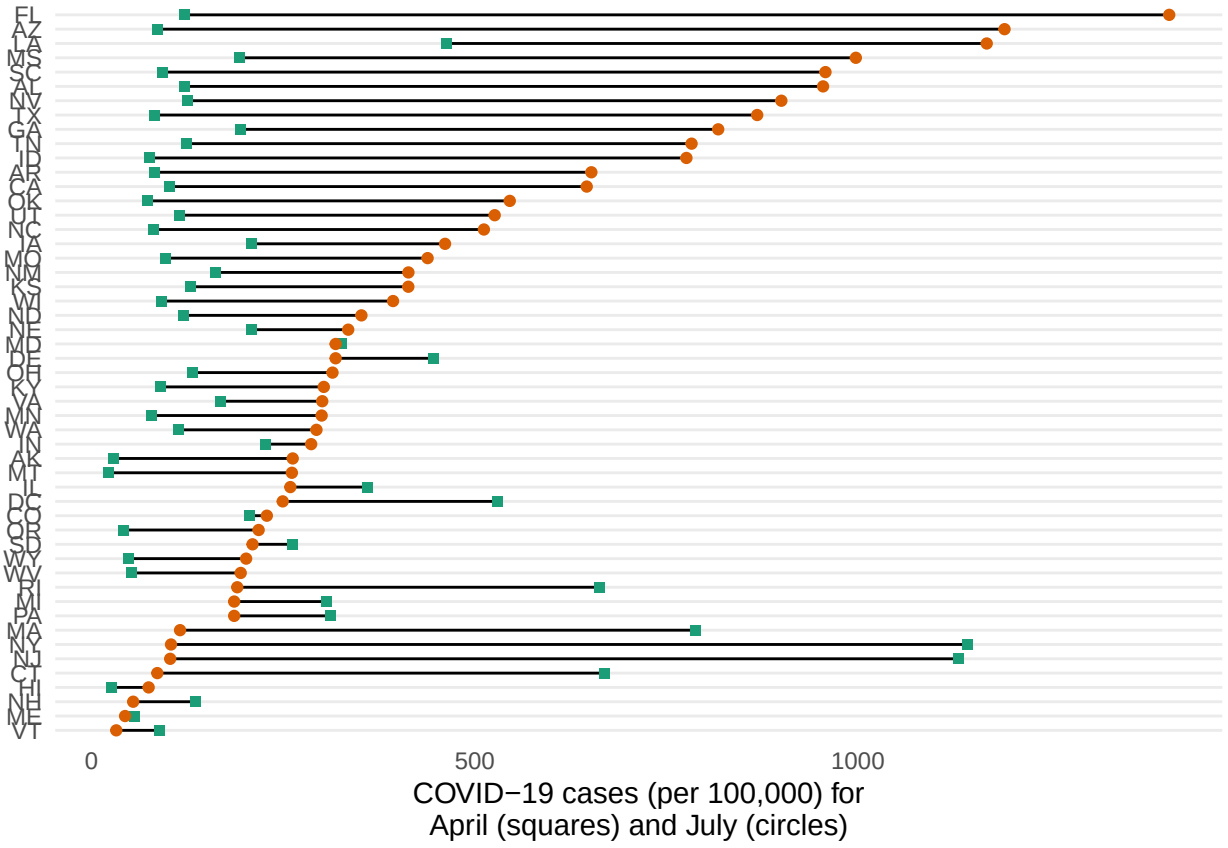
Add a `theme()` layer and use `element_blank()` to remove the major and minor panel grids for the x axis. This leaves horizontal light gray lines to guide the eyes to state codes on the vertical axis. Do this *after* you change your overall theme.

Tidy vs non-tidy data

You can make this graph from tidy or non-tidy data. The tidy form follows closely from HW 13. You'll have to think carefully about the code needed to arrange the states by number of July cases.

For non-tidy data, you may need add a `geom_segment` layer first (see `?geom_segment`) to add horizontal lines that connect the points. They need an `x` and `y` coordinate as a starting point and an `xend` and `yend` coordinate to end. `x` and `xend` will be the same and correspond to `State`. You should be able to think through this to determine what you need for `y` and `yend`.

Regardless of tidy vs non-tidy data, change the color of the connecting line to a lighter shade of grade. The segment visually connect the two points but is not the focus. You want it to stand out from the panel grid but you don't want it to be pure black. You can also play around with the size (thickness) of the line.



Plot 3

Plot 4: Missouri COVID cases, highlighting when state opened.

Plot the number of daily new cases for Missouri. The columns in the plot show the number of new cases for each day. The highlighted column is the official day Missouri reopened after the initial shut down. The line is the rolling mean that shows the average number of new cases for the previous seven days.

Requirements Exclude all states except Missouri and exclude dates before the first case was recorded for Missouri (the date for the first case should be included).

Write an R *function* that calculates the number of daily new cases from a vector of daily total cases. Take advantage of the R's vector-oriented approach in your function to calculate daily new cases. Conceptually:

Date	05/21	05/22	05/23	05/24	05/25
Vector	47	52	67	73	91
Date	05/20	05/21	05/22	05/23	05/24
Vector	39	47	52	67	73
Result	8	5	15	6	18

- Hint: `cases[12:16] - cases[11:15]`. **Do not hard code the vector length.**
- You will need to pad the *left* side of the resulting vector with a single `0` (zero) to keep the length of the output the same as the length of the input.

Use your function to create a `daily` column of new cases in your tibble, calculated from the cases column.

The following code calculates the rolling mean using the `frollmean()` function from `data.table`, and adds it to your tibble. Replace `tibble` with the actual name of your tibble. Replace `daily` with whatever column name you used *if you didn't use* `daily`. The average is calculated on the 7th day for the previous week. Thus, the first six days will be `NA`, so `replace_na(0)` replaces those NAs with zeros. *This will not work unless you installed the* ***`data.table`*** *package as instructed above.*

```
# frollmean is calculates a 7-day rolling average of daily new cases.
tibble$roll_mean <-
  data.table::frollmean(tibble$daily, 7, align = "right") %>%
  replace_na(0)
```

Missouri Governor Mike Parsons officially declared Missouri reopened for business on 16 June 2020. Highlight that column in the plot.

Use `annotate` to add `label = "Missouri reopened\n16 June 2020"`. We haven't used `annotate` so you'll have to look up how to use it with `?annotate`. The `\n` in the label adds the line break so the text appears on two lines. Position the label in the approximate position shown in the example below.

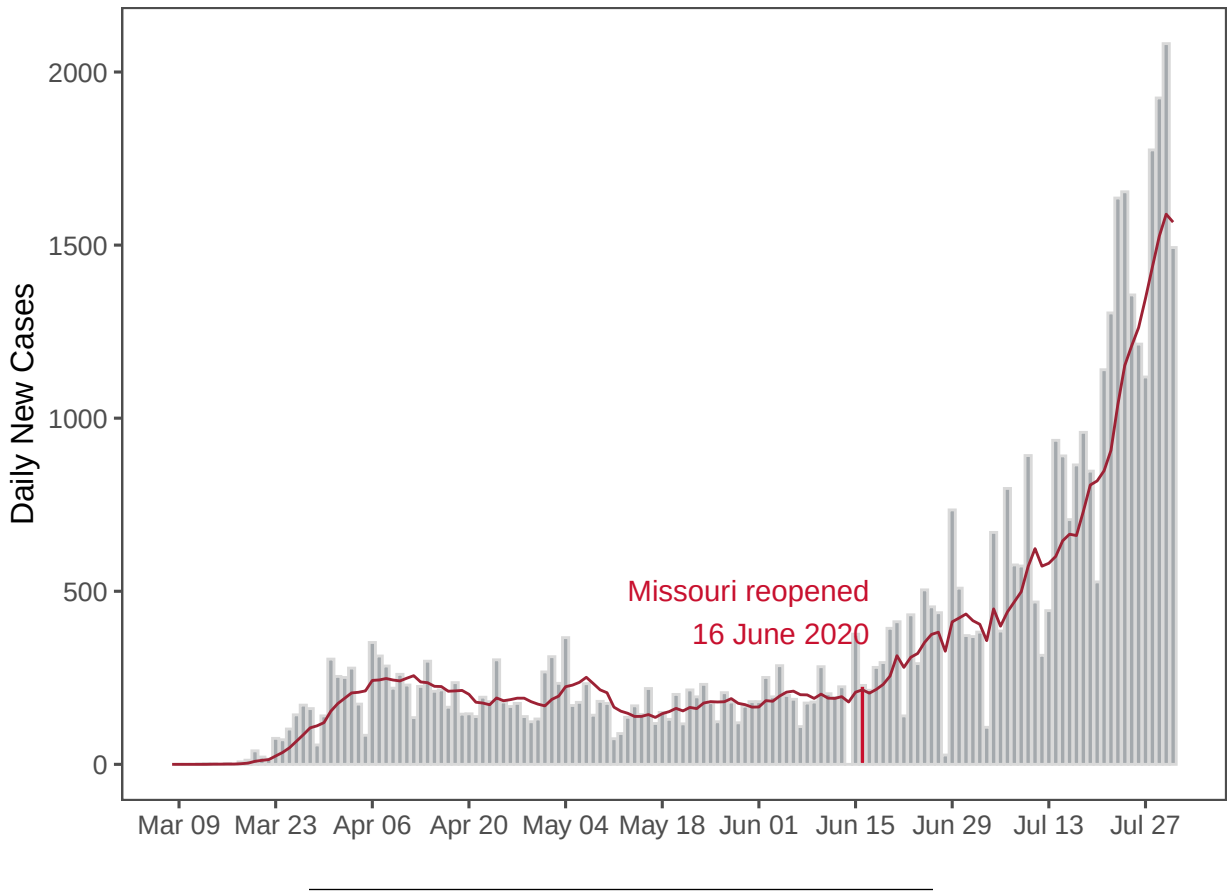
Read the help for `scale_x_date` to set the date labels on the x-axis to `dd mmm` and the date breaks to every two weeks.

Use SEMO's Cardiac Red (#9D2235) for the rolling mean line and Southeast Red (#C8102E) for the annotation and fill color for the highlighted column. I used `gray85` for the other columns but you are free to experiment with similar shades of gray or light colors.

It's okay if you get this warning message shown above the figure.

Plot 4

```
## Warning: Removed 1 rows containing missing values (geom_col).
```



Plot 5: Table and choropleth map

SKIP