Week 5 R tutorial (supplement)

Statistics and statistical programming
Northwestern University
MTS 525

Aaron Shaw

October 13, 3030

Contents

Getting started (more better plots)

This is a supplement to the Week 5 R tutorial focused on elaborating some examples of time series plots and more polished plots using `ggplot2`. I'll work some data on state-level COVID-19 in the United States published by *The New York Times* (*NYT*). You can access the data an details about the sources, measurement, and different datasets available via the *NYT* github repository.

To start, I'll load up the `tidyverse` library and also attach the `lubridate` package to help handle dates and times. Then I'll import the "raw csv" from the web, and take a look at the dataset:

```r
library(tidyverse)
library(lubridate)

data_url <- url("https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-states.csv")

d <- read_csv(data_url)

d
```

```
## # A tibble: 12,004 x 5
##    date       state      fips  cases deaths
##    <date>     <chr>      <chr> <dbl>  <dbl>
##  1 2020-01-21 Washington 53        1      0
##  2 2020-01-22 Washington 53        1      0
##  3 2020-01-23 Washington 53        1      0
##  4 2020-01-24 Illinois   17        1      0
##  5 2020-01-24 Washington 53        1      0
##  6 2020-01-25 California 06        1      0
##  7 2020-01-25 Illinois   17        1      0
##  8 2020-01-25 Washington 53        1      0
##  9 2020-01-26 Arizona    04        1      0
## 10 2020-01-26 California 06        2      0
## # ... with 11,994 more rows
```

For the sake of my examples, I'm planning to work with the `date`, `state`, `cases`, and `deaths` variables. Notice that by using the `read_csv()` function to import the data, R already recognizes the `date` column as dates. It looks like I need to convert the state variable to a factor, however. After I do that I can get a quick sense of how much data I have for each state with a univariate table that just counts the number of observations (rows) for each value of `state`.

```r
d$state <- factor(d$state)
table(d$state)
```

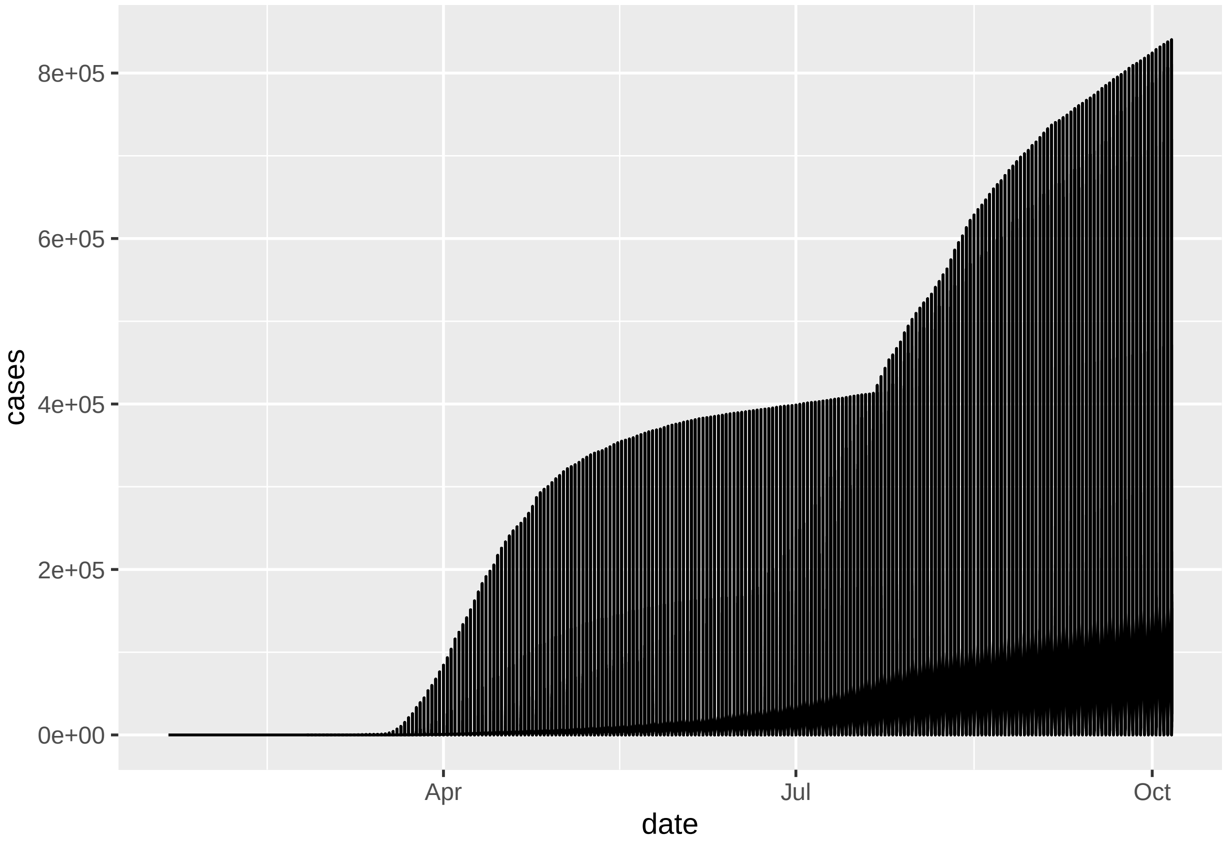
```
## 
##                  Alabama                   Alaska                  Arizona 
##                      208                      209                      255 
##                 Arkansas               California                 Colorado 
##                      210                      256                      216 
##              Connecticut                 Delaware     District of Columbia 
##                      213                      210                      214 
##                  Florida                  Georgia                     Guam 
##                      220                      219                      206 
##                   Hawaii                    Idaho                 Illinois 
##                      215                      208                      257 
##                  Indiana                     Iowa                   Kansas 
##                      215                      213                      214 
##                 Kentucky                Louisiana                    Maine 
##                      215                      212                      209 
##                 Maryland            Massachusetts                 Michigan 
##                      216                      249                      211 
##                Minnesota              Mississippi                 Missouri 
##                      215                      210                      214 
##                  Montana                 Nebraska                   Nevada 
##                      208                      233                      216 
##            New Hampshire               New Jersey               New Mexico 
##                      219                      217                      210 
##                 New York           North Carolina             North Dakota 
##                      220                      218                      210 
## Northern Mariana Islands                     Ohio                 Oklahoma 
##                      193                      212                      215 
##                   Oregon             Pennsylvania              Puerto Rico 
##                      222                      215                      208 
##             Rhode Island           South Carolina             South Dakota 
##                      220                      215                      211 
##                Tennessee                    Texas                     Utah 
##                      216                      238                      225 
##                  Vermont           Virgin Islands                 Virginia 
##                      214                      207                      214 
##               Washington            West Virginia                Wisconsin 
##                      260                      204                      245 
##                  Wyoming 
##                      210
```

Plotting a univariate time series

I recommend using `geom_path()` to create univariate time series plots. Specifically, I'll call `geom_line()`, which is a specialized version of `geom_path()` that connects observations in order according to the values of variable that is mapped to the x-axis. By convention, a univariate time series maps dates to the x-axis, so this will just plot a line connecting the dots over time.

For my first example, I want to build up a plot of weekly case counts in Illinois. I can start off by just plotting the cumulative cases for all of the states and work my way towards the specific plot I want from there:

```
ggplot(data = d, aes(date, cases)) +
  geom_line()
```



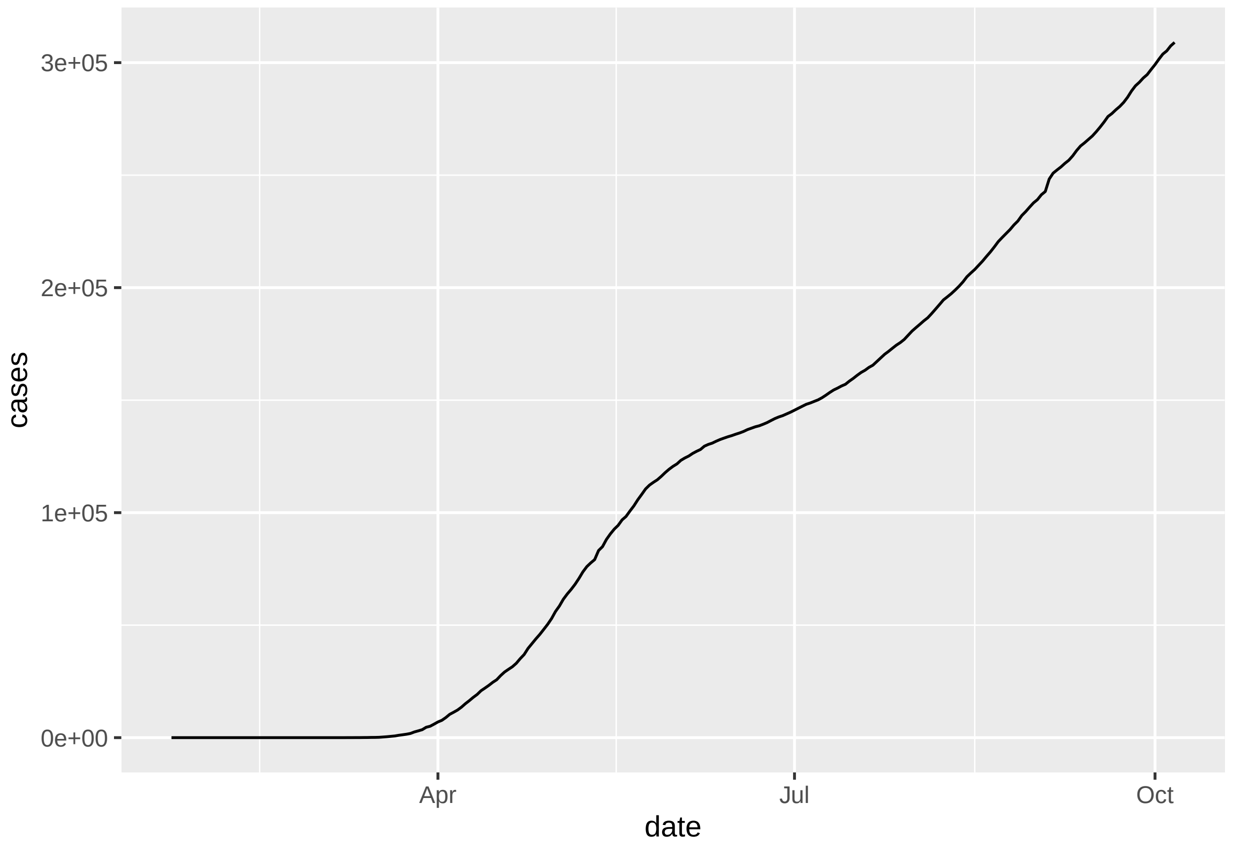
Notice that ggplot handles the `date` variable quite well by default! It recognizes the units of time and generates axis labels in terms of months. Also notice that ggplot handles the axis labels for the `cases` variable. . . less well. I don't know about you, but my brain doesn't parse scientific notation quickly/easily.

Tidying timeseries data for better plots

Okay, let's get to work cleaning all this up. At this point, my next steps are to (1) restrict the data to the Illinois cases; (2) reorganize the *cumulative* daily case counts into weekly counts; and (3) plot it again with better axis labels and a nice title.

I can restrict the data to Illinois in a few ways. Since I'm using ggplot, I'll work with Tidyverse "pipes" (`%>%`) and "verbs" (in this case, `filter`):

```
d %>%
  filter(state == "Illinois") %>%
  ggplot(aes(date, cases)) +
  geom_line()
```



That's already much less cluttered. Inserting a call to the Tidyverse `mutate`, `group_by`, and `summarize` verbs can help me generate the weekly counts I'm looking for. Here's the code to produce a new object. I'll walk through it below:

```
il_weekly_cases <- d %>%
  filter(state == "Illinois") %>%
  mutate(
    diff_cases = c(cases[1], diff(cases, lag = 1)),
    weekdate = cut(date, "week")
  ) %>%
  group_by(weekdate) %>%
  summarize(new_cases = sum(diff_cases, na.rm = T), )

il_weekly_cases
```

```
## # A tibble: 38 x 2
##    weekdate   new_cases
##    <fct>          <dbl>
##  1 2020-01-20         1
##  2 2020-01-27         1
##  3 2020-02-03         0
##  4 2020-02-10         0
##  5 2020-02-17         0
##  6 2020-02-24         1
##  7 2020-03-02         4
##  8 2020-03-09        87
##  9 2020-03-16       953
## 10 2020-03-23      3568
## # ... with 28 more rows
```

There's quite a lot happening there. I'll go through it verb-by-verb.

First, I use `mutate` to create a `diff_cases` variable that disaggregates the cumulative values of `cases` (read the documentation for `diff` to learn more about this one). Differenced values alone wouldn't produce the same number of items (try running `length(1:10)` and compare that with `length(diff(1:10, 1))` to see what I mean), so I stores the first value of my `cases` variable and then append the differenced values after that. Within the same call to mutate I also create a new variable `weekdate` that collapses the dates into weeks (see the documentation for `cut.Date`) and stores the resulting strings as factors (e.g., a factor where the levels correspond to a series of Mondays: "2020-01-20", "2020-01-27"...). Hopefully, so far so good?

Next, I use `group_by` to aggregate everything by my `weekdate` factor values.

Finally I use `summarize` to reshape my data and collapse everything into weekly counts of new cases (notice that I use `sum` inside the `summarize` call to add up the case counts within the grouping variable). Okay, let's see about plotting this now:

Hmm. looks like I have a problem with my dates. Let's troubleshoot this:

```
class(il_weekly_cases$weekdate)
```

```
## [1] "factor"
```

Whoops. It looks like I need to convert that `weekdate` variable into an object of class "date" so that it will work with ggplot. There are a number of ways I could do this, but I'll just make a new variable by first converting `weekdate` to a character vector and then converting that into a date using `as.Date` (and remember that it is sometimes easier to read these "nested" commands from the inside-out).

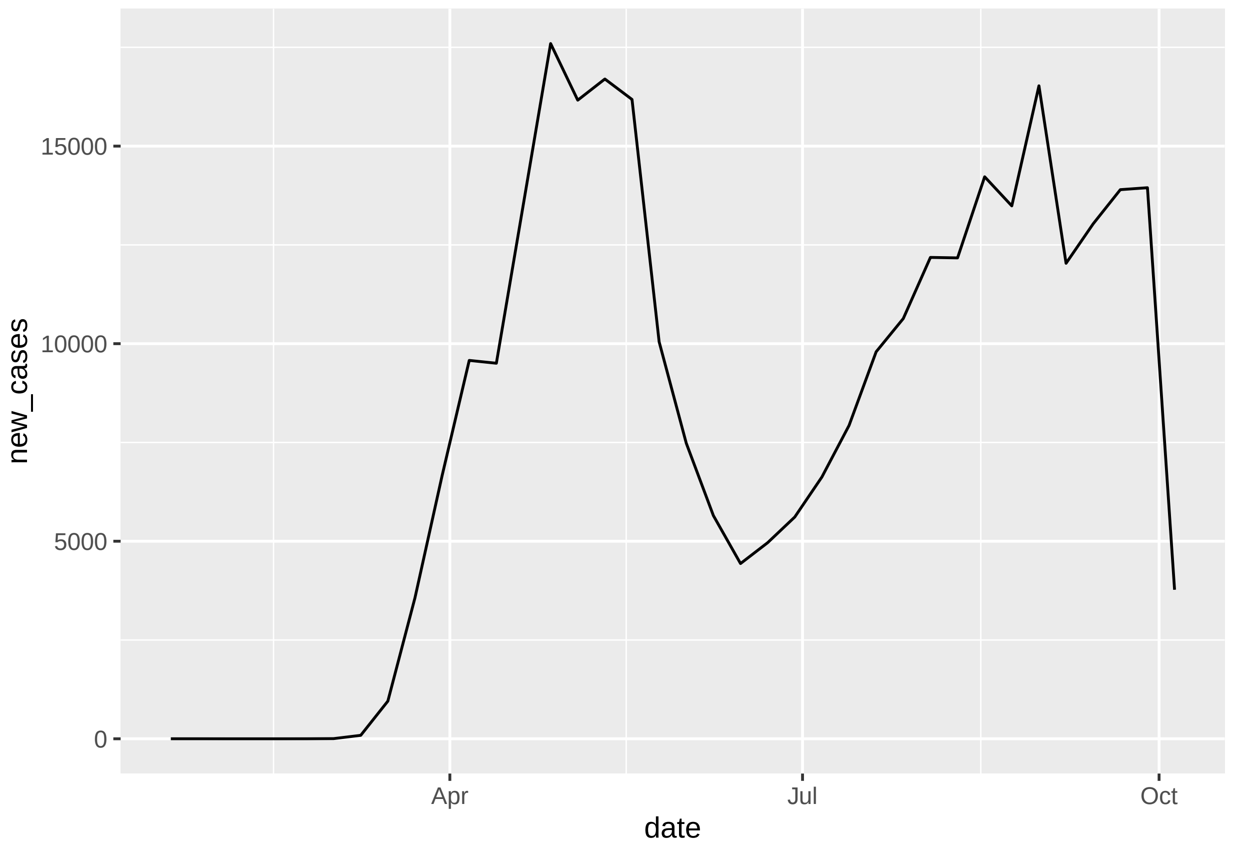
```
il_weekly_cases$date <- as.Date(as.character((il_weekly_cases$weekdate)))
il_weekly_cases
```

```
## # A tibble: 38 x 3
##    weekdate   new_cases date      
##    <fct>          <dbl> <date>    
##  1 2020-01-20         1 2020-01-20
##  2 2020-01-27         1 2020-01-27
##  3 2020-02-03         0 2020-02-03
##  4 2020-02-10         0 2020-02-10
##  5 2020-02-17         0 2020-02-17
##  6 2020-02-24         1 2020-02-24
##  7 2020-03-02         4 2020-03-02
##  8 2020-03-09        87 2020-03-09
##  9 2020-03-16       953 2020-03-16
## 10 2020-03-23      3568 2020-03-23
## # ... with 28 more rows
```

That ought to work now:

```
plot1 <- il_weekly_cases %>%
  ggplot(aes(date, new_cases)) +
  geom_line()

plot1
```



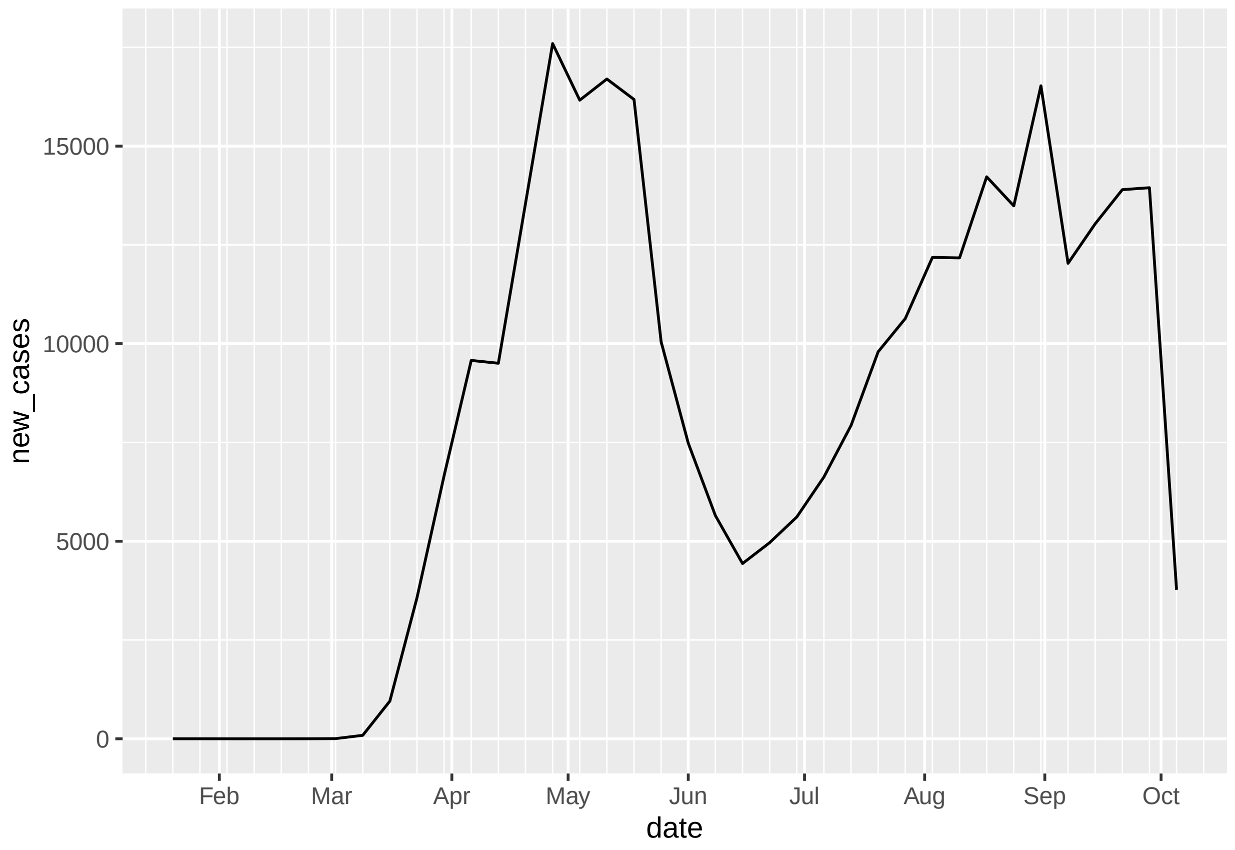
Much better! Notice that the final week of the data appears to fall off a cliff. That's just an artifact of the way that the *NYT* has published the data for part of the most recent week. Once it updates, the case count probably won't drop like that (yikes). Anyhow, onwards to cleaning things up and adding a title.

Working on ggplot axis labels, titles, and scales

As I mentioned briefly in class `ggplot2` treats labels, titles, and scales as "layers" within it's "grammar of graphics" (and yes, I'm rolling my eyes as I type those scare-quotes). For the purposes of our example here I'm going to use `scale_date` to work with the x-axis, `scale_continuous` to work with the y-axis, and `labs` to clean up the title and axis labels.

For starters, let's see whether there might be any way I want to improve the axis labels. The ggplot defaults for my `date` variable are pretty good already, but maybe I want to incorporate a label/break for each month as well as a more granular grid in the background that shows the weeks? Here's what all of that looks like:

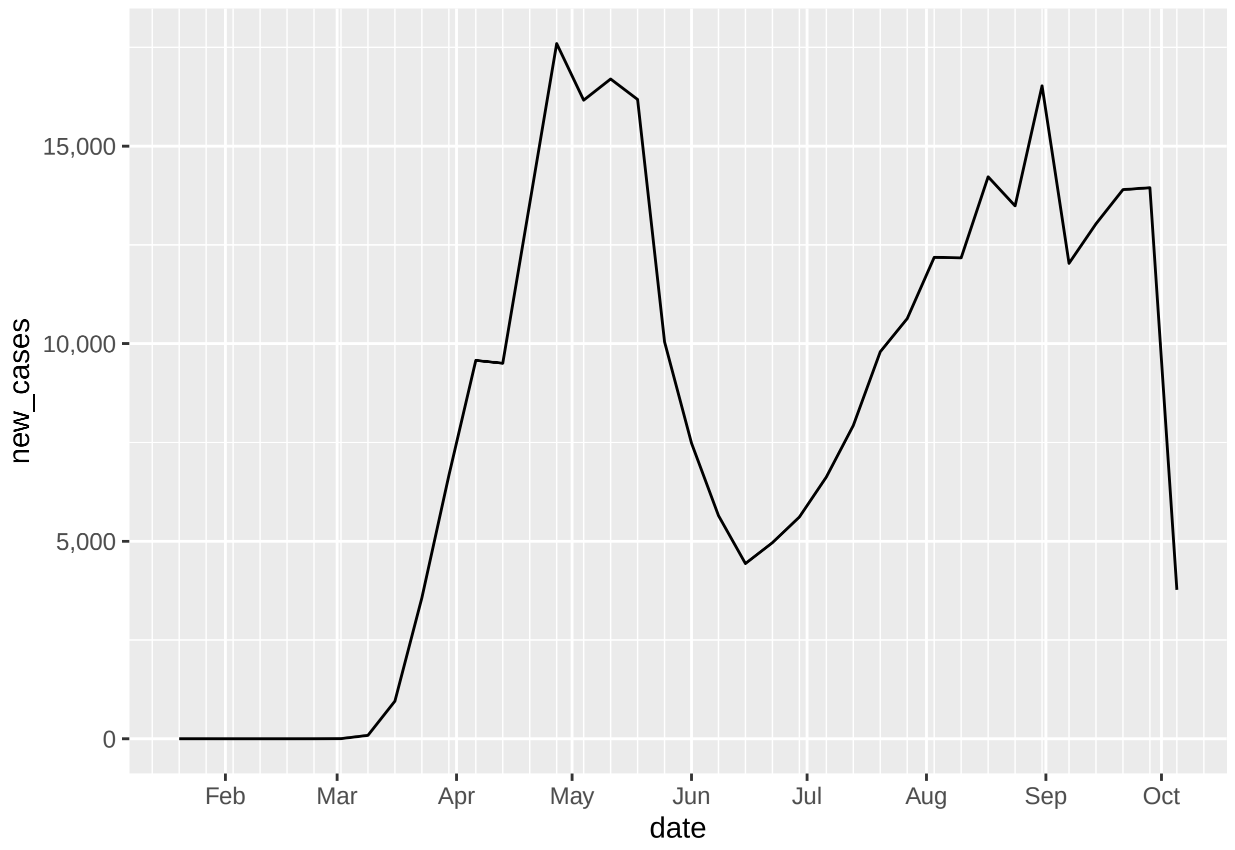
```
plot2 <- plot1 + scale_x_date(date_labels = "%b", date_breaks = "1 month", date_minor_breaks = "1 week"
plot2
```



The ggplot documentation for `scale_date` can give you some other examples and ideas. Also, notice how I appended the `scale_date` layer to my existing plot and stored it as a new object? This can make it easier to work iteratively without losing any of my earlier layers along the way.

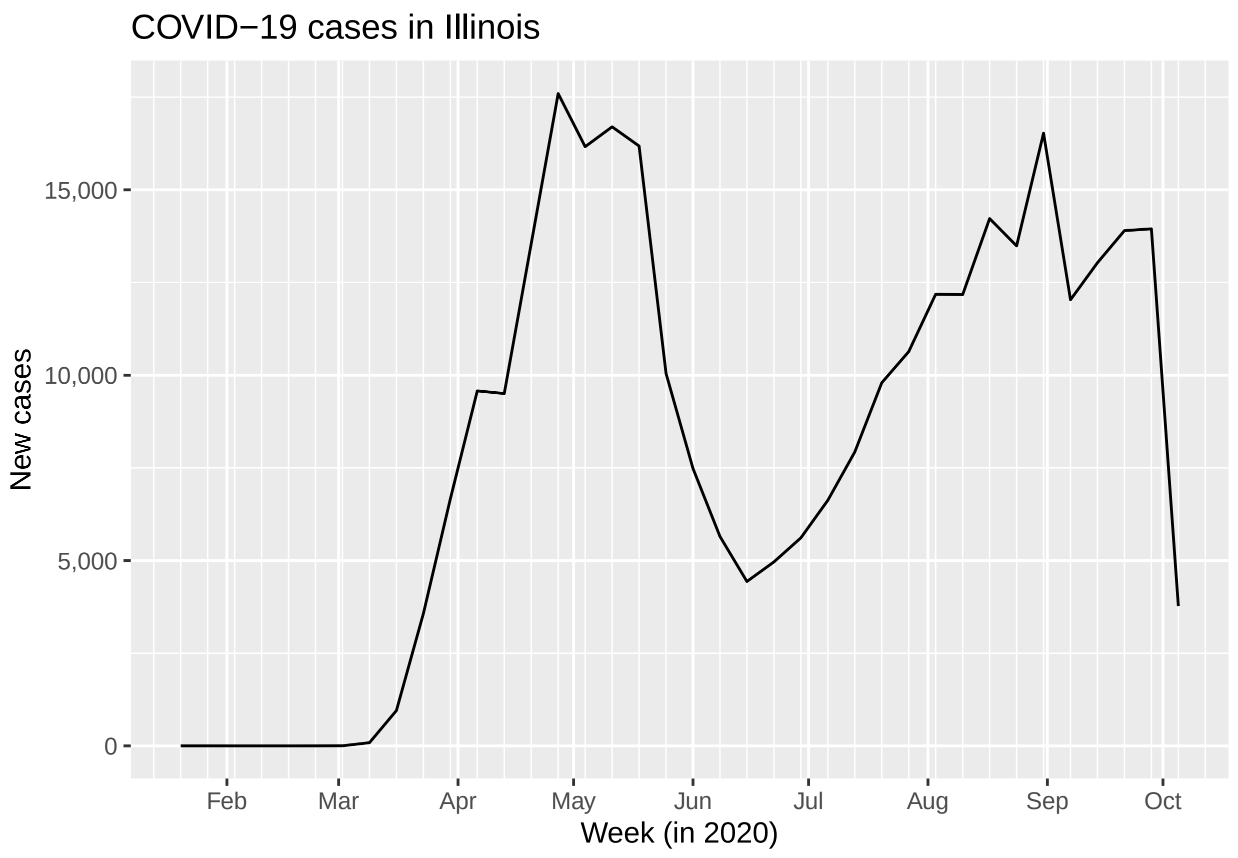
Now I can fix up the y-axis labels a bit using a call to the `labels` argument after I load the `scales` package.

```r
library(scales)
plot3 <- plot2 + scale_y_continuous(label = comma)
plot3
```



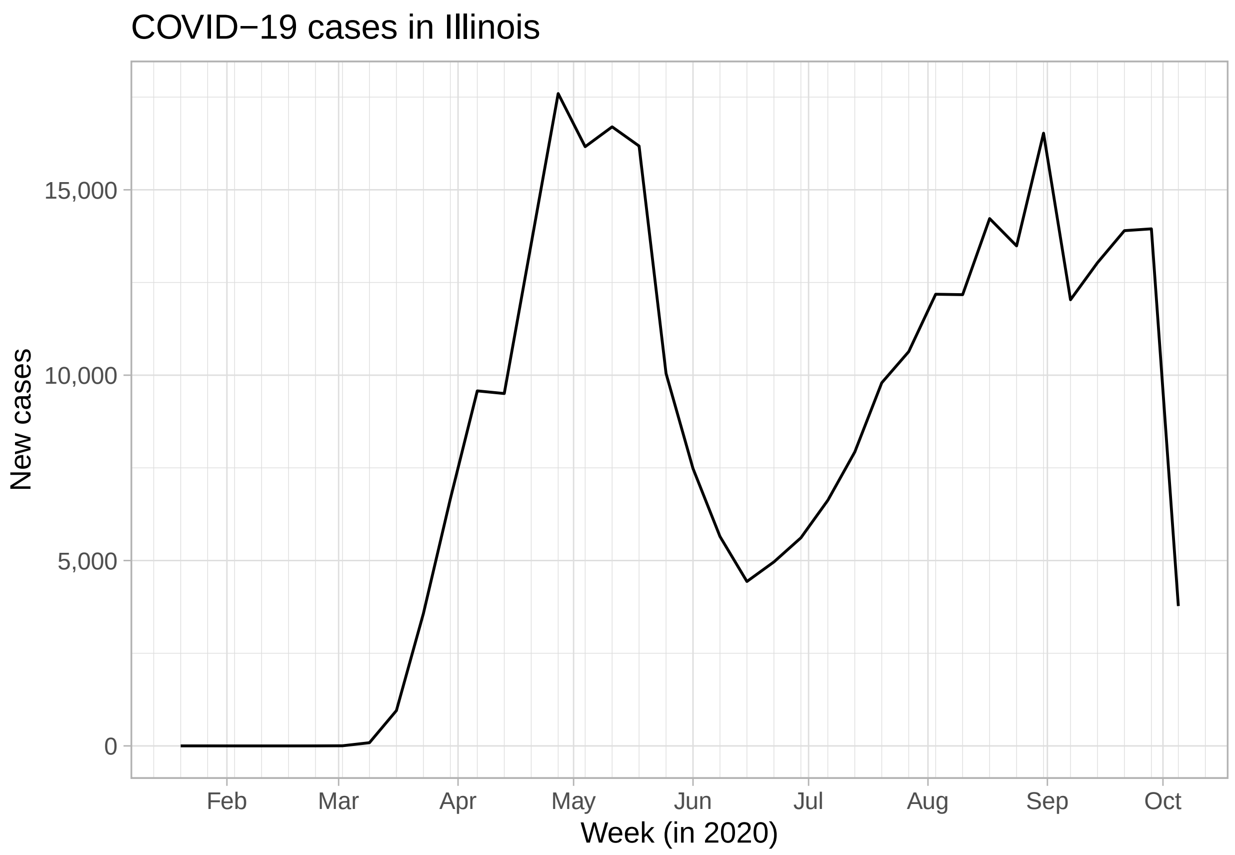
Nearly done. All that’s left is a title and better axis names. I’ll do that with yet another layer.

```
plot4 <- plot3 + labs(x = "Week (in 2020)", y = "New cases", title = "COVID-19 cases in Illinois")
plot4
```



Last, but not least, I mentioned in our class session that ggplot also has "themes" that can be useful for styling plots. One I have used for publications is the "light" theme. Here's how to apply that:

```r
plot4 + theme_light()
```



That's looking much better than when we started! If you wanted to export it as a standalone file (e.g., .png, .pdf, or whatever), I recommend looking at the documentation for the `ggsave()` function, which is available via ggplot2. Base R also has a `save()` function that you can work with, although it can be a bit more complicated to get comfortable with.

Long versus wide data (and why long data is often helpful)

So what if you wanted to plot a multivariate time series (e.g., the same plot for more than one state and/or for more than one measure)? As always, you have a number of options, but the most effective way to achieve this with ggplot involves learning to work with "long" format data.

Thus far, we have worked mostly with "wide" format data where (nearly) every row corresponds to a single unit/observation and every column corresponds to a variable (for which we usually have no more than one value attributed to any unit/observation). Wide format data is great for many things, but it turns out that learning to work with long format data can be super helpful for a number of purposes. Producing richer, multidimensional ggplot visualizations is one of them.

Consider the format of my tidied dataframe that I used for plotting:

```
il_weekly_cases
```

```
## # A tibble: 38 x 3
##    weekdate   new_cases date
##    <fct>          <dbl> <date>
##  1 2020-01-20         1 2020-01-20
##  2 2020-01-27         1 2020-01-27
##  3 2020-02-03         0 2020-02-03
##  4 2020-02-10         0 2020-02-10
##  5 2020-02-17         0 2020-02-17
##  6 2020-02-24         1 2020-02-24
##  7 2020-03-02         4 2020-03-02
##  8 2020-03-09        87 2020-03-09
##  9 2020-03-16       953 2020-03-16
## 10 2020-03-23      3568 2020-03-23
## # ... with 28 more rows
```

This dataframe is in a "wide" format. Each row is a week and each column is a variable unique to that week.

Our original dataframe was a bit "longer":

```
d
```

```
## # A tibble: 12,004 x 5
##    date       state      fips  cases deaths
##    <date>     <fct>      <chr> <dbl>  <dbl>
##  1 2020-01-21 Washington 53        1      0
##  2 2020-01-22 Washington 53        1      0
##  3 2020-01-23 Washington 53        1      0
##  4 2020-01-24 Illinois   17        1      0
##  5 2020-01-24 Washington 53        1      0
##  6 2020-01-25 California 06        1      0
##  7 2020-01-25 Illinois   17        1      0
##  8 2020-01-25 Washington 53        1      0
##  9 2020-01-26 Arizona    04        1      0
## 10 2020-01-26 California 06        2      0
## # ... with 11,994 more rows
```

We see multiple observations per state (I think I would say the units or rows correspond to "state-dates" or something like that). It's not completely "long" however, because we also have multiple columns corresponding to the two variables of interest: `cases` and `deaths`. The point I want to make is that there are a number of ways we can make this data "longer." For the purposes of producing a multi-state plot like the one above, the most important of these is going to involve dropping the step where I filtered by `state=="Illinois"` and replacing by a `group_by` step before I create my `weekdate` variable. I'm also going to go ahead and drop the `date` and `fips` variables because they're just getting in my way at this point. I'll start there

```
weekly <- d %>%
  group_by(state) %>%
  mutate(
    weekdate = cut(date, "week"),
  ) %>%
  select(state, cases, deaths, weekdate)
weekly
```

```
## # A tibble: 12,004 x 4
## # Groups:   state [55]
##    state      cases deaths weekdate
##    <fct>      <dbl>  <dbl> <fct>
##  1 Washington     1      0 2020-01-20
##  2 Washington     1      0 2020-01-20
##  3 Washington     1      0 2020-01-20
##  4 Illinois       1      0 2020-01-20
##  5 Washington     1      0 2020-01-20
##  6 California     1      0 2020-01-20
##  7 Illinois       1      0 2020-01-20
##  8 Washington     1      0 2020-01-20
##  9 Arizona        1      0 2020-01-20
## 10 California     2      0 2020-01-20
## # ... with 11,994 more rows
```

I'm getting somewhere with this, I promise. One of the principles of "tidy" data is to make it so that every variable has a column, every observation has a row, and every value has a cell. Right now, I've got multiple observations for each state-week spread across multiple rows. Remember that my `cases` and `deaths` variables are actually cumulative counts, so I really only need to store the maximum value for each state-week in order to calculate the new cases per state-week. Let's see what to do about that:

```
tidy_weekly <- weekly %>%
  group_by(state, weekdate) %>%
  summarize(
    cum_cases = max(cases, na.rm = T),
    cum_deaths = max(deaths, na.rm = T)
  )
```

```
tidy_weekly$weekdate <- as.Date(as.character(tidy_weekly$weekdate))

tidy_weekly <- tidy_weekly %>%
  group_by(state) %>%
  arrange(-desc(weekdate)) %>%
  mutate(
    new_cases = c(cum_cases[1], diff(cum_cases, lag = 1)),
    new_deaths = c(cum_deaths[1], diff(cum_deaths, lag = 1)),
  )

tidy_weekly
```

```
## # A tibble: 1,780 x 6
## # Groups:   state [55]
##    state         weekdate   cum_cases cum_deaths new_cases new_deaths
##    <fct>         <date>         <dbl>      <dbl>     <dbl>      <dbl>
##  1 Arizona       2020-01-20         1          0         1          0
##  2 California    2020-01-20         2          0         2          0
##  3 Illinois      2020-01-20         1          0         1          0
##  4 Washington    2020-01-20         1          0         1          0
##  5 Arizona       2020-01-27         1          0         0          0
##  6 California    2020-01-27         6          0         4          0
##  7 Illinois      2020-01-27         2          0         1          0
##  8 Massachusetts 2020-01-27         1          0         1          0
##  9 Washington    2020-01-27         1          0         0          0
## 10 Arizona       2020-02-03         1          0         0          0
## # ... with 1,770 more rows
```

This is headed in the right direction. For some purposes, though, it's still not quite "long" enough For starters, I can drop the cumulative cases and deaths columns. The other thing I can do is "pivot" the data to organize the `new_cases` and `new_deaths` measures a little differently. To manage this, I'll use the `pivot_longer()` function (part of the `tidyr` package from the tidyverse). I will also go ahead and coerce my `weekdate` into a Date object again:

```
long_weekly <- tidy_weekly %>%
  select(state, weekdate, new_cases, new_deaths) %>%
  pivot_longer(
    cols = starts_with("new"),
    names_to = "variable",
    values_to = "value"
  )

long_weekly
```

```
## # A tibble: 3,560 x 4
## # Groups:   state [55]
##    state      weekdate   variable   value
##    <fct>      <date>     <chr>      <dbl>
##  1 Arizona    2020-01-20 new_cases      1
##  2 Arizona    2020-01-20 new_deaths     0
##  3 California 2020-01-20 new_cases      2
##  4 California 2020-01-20 new_deaths     0
##  5 Illinois   2020-01-20 new_cases      1
##  6 Illinois   2020-01-20 new_deaths     0
##  7 Washington 2020-01-20 new_cases      1
##  8 Washington 2020-01-20 new_deaths     0
##  9 Arizona    2020-01-27 new_cases      0
## 10 Arizona    2020-01-27 new_deaths     0
## # ... with 3,550 more rows
```

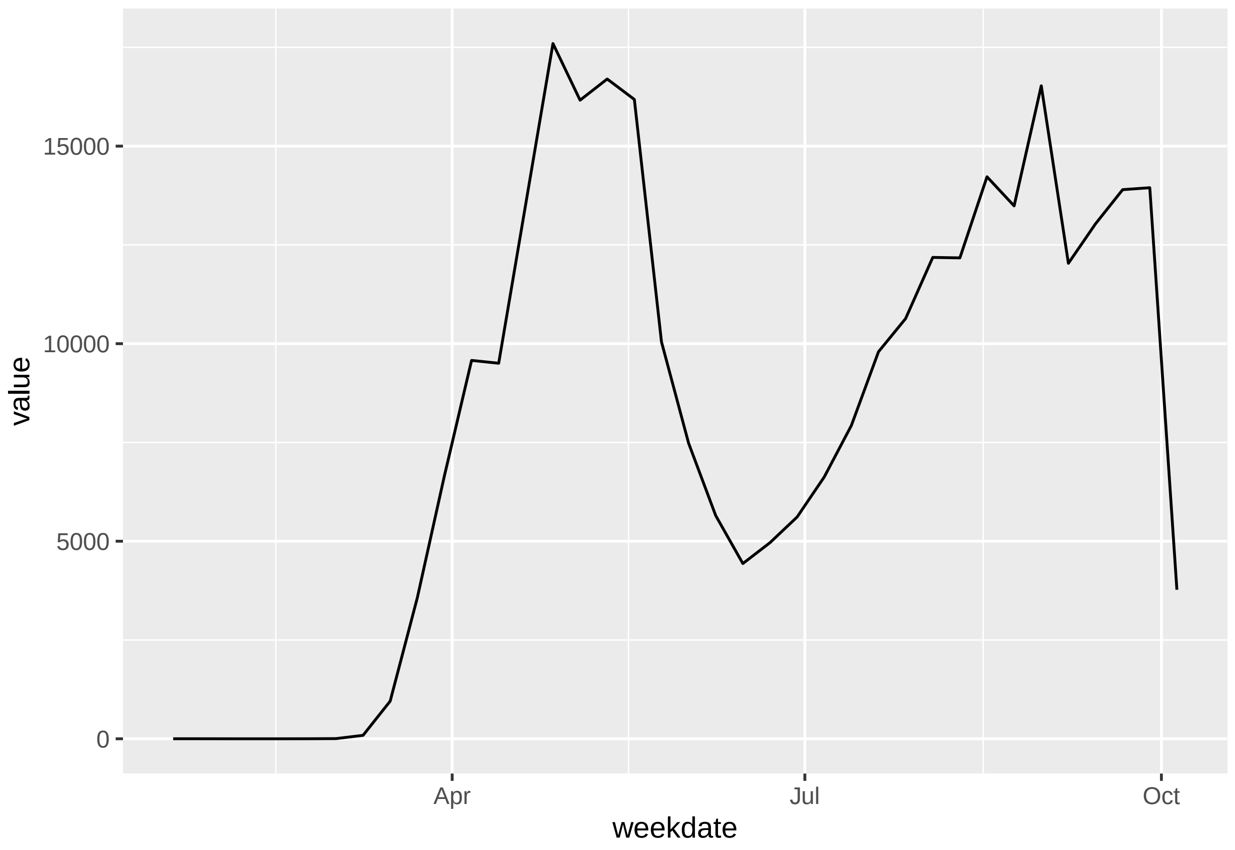
Can you see what that did? I now have two rows of data for every state-week. One that contains a value for `new_cases` and one that contains a value for `new_deaths`. Both of those variables have been "pivoted" into a single `variable` column.

Before we move forward I'm going to clean up the values of `variable`.

```
long_weekly <- long_weekly %>%
  mutate(
    variable = recode(variable, new_cases = "new cases", new_deaths = "new deaths")
  )
```

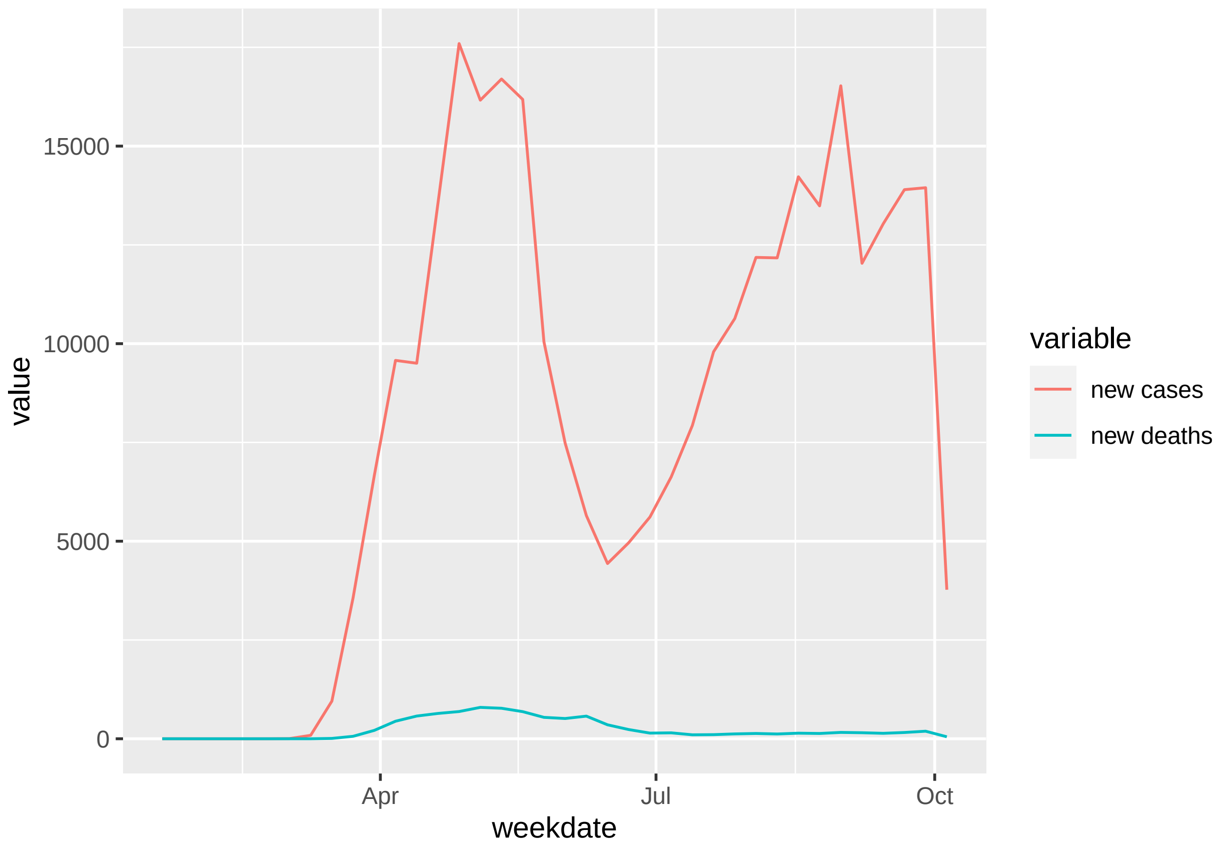
Okay, prepared with my `tidy_weekly` and my `long_weekly` tibbles, I'm now ready to generate some more interesting multidimensional plots. Let's start with the same sort of time series of new cases we made for Illinois before so we can see how to replicate that with this new data structure:

```
long_weekly %>%
  filter(
    state == "Illinois" & variable == "new cases"
  ) %>%
  ggplot(aes(weekdate, value)) +
  geom_line()
```



Now we can easily plot Illinois cases against deaths from the same tibble:

```
long_weekly %>%
  filter(state == "Illinois") %>%
  ggplot(aes(weekdate, value, color = variable)) +
  geom_line()
```

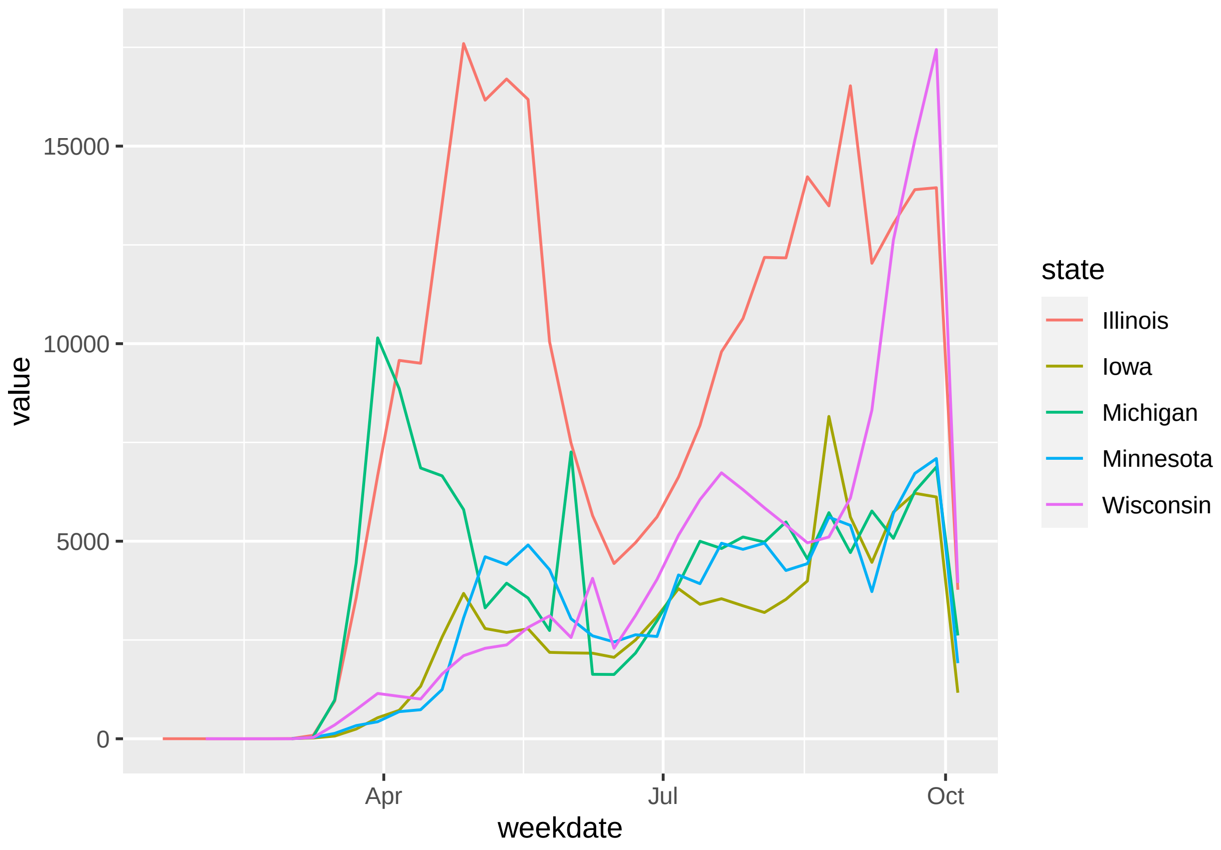


That plot isn't so great because the death counts are dwarfed by the case counts. Thank goodness!

Now let's compare Illinois case counts against some its neighbors in the upper midwest:

```r
upper_midwest <- c("Illinois", "Michigan", "Wisconsin", "Iowa", "Minnesota")

long_weekly %>%
  filter(state %in% upper_midwest & variable == "new cases") %>%
  ggplot(aes(weekdate, value, color = state)) +
  geom_line()
```

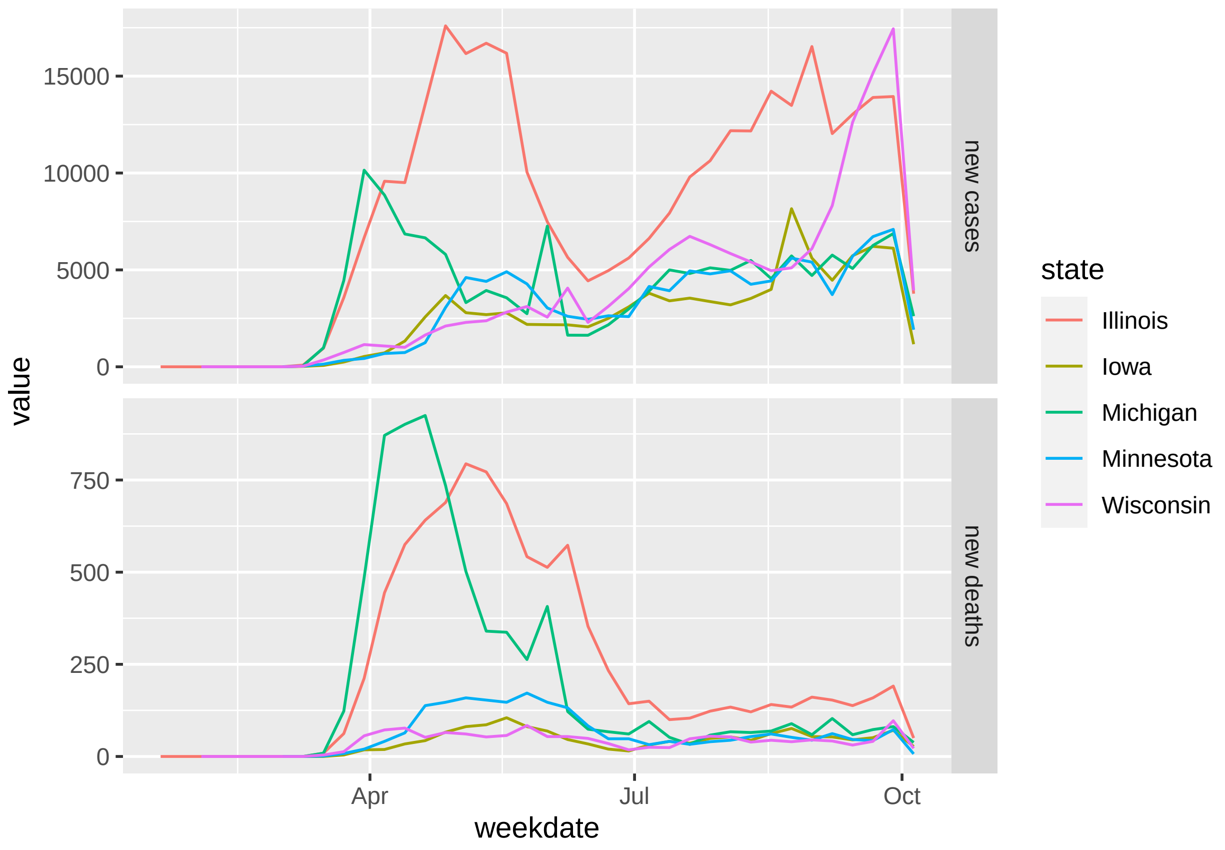


Now that's getting a bit more interesting.

What about finding some way to also incorporate the death counts? Well, ggplot has another layer option called "facets" that can help produce multiple plots and present them alongside each other (or in a grid). Here's an example that creates a faceted "grid" (really just a side-by-side comparison) of case counts and deaths for the same five states.

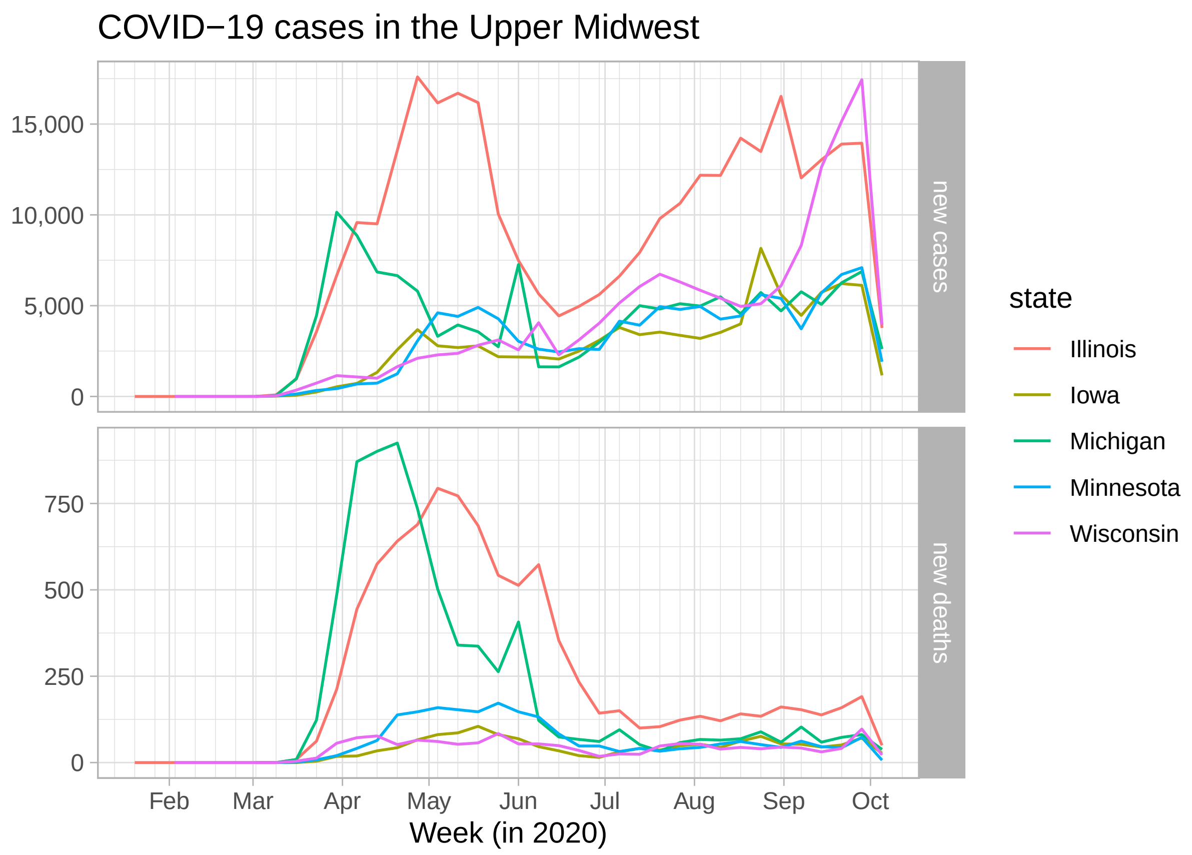
```
midwest_plot <- long_weekly %>%
  filter(state %in% upper_midwest) %>%
  ggplot(aes(weekdate, value, color = state)) +
  geom_line() +
  facet_grid(rows = vars(variable), scales = "free_y")

midwest_plot
```



Now we can clean up some of the other elements we worked on with the original plot (axes, title, etc.). I'll bake that into a single chunk below.

```
midwest_plot + scale_x_date(date_labels = "%b", date_breaks = "1 month", date_minor_breaks = "1 week")
```


COVID−19 cases in the Upper Midwest
15,000
10,000
5,000
0
new cases
750
500
250
0
new deaths
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Week (in 2020)
state
Illinois
Iowa
Michigan
Minnesota
Wisconsin