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 with some data on state-level COVID-19 in the United States published by *The New York Times* (*NYT*). You can access the data as well as details about the sources, measurement, and related available datasets via the *NYT* github repository.

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

```
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: 12,059 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 12,049 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. Also notice that the column names for cases and deaths don't reflect the fact that both variables are *cumulative* counts. Also also, notice that it looks like I need to convert the state variable to a factor. I'll start there and then get a quick sense of how much data I have for each state with a univariate table.

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

```
## 
##                  Alabama                   Alaska                  Arizona 
##                      209                      210                      256 
##                 Arkansas               California                 Colorado 
##                      211                      257                      217 
##              Connecticut                 Delaware     District of Columbia 
##                      214                      211                      215 
##                  Florida                  Georgia                     Guam 
##                      221                      220                      207 
##                   Hawaii                    Idaho                 Illinois 
##                      216                      209                      258 
##                  Indiana                     Iowa                   Kansas 
##                      216                      214                      215 
##                 Kentucky                Louisiana                    Maine 
##                      216                      213                      210 
##                 Maryland            Massachusetts                 Michigan 
##                      217                      250                      212 
##                Minnesota              Mississippi                 Missouri 
##                      216                      211                      215 
##                  Montana                 Nebraska                   Nevada 
##                      209                      234                      217 
##            New Hampshire               New Jersey               New Mexico 
##                      220                      218                      211 
##                 New York           North Carolina             North Dakota 
##                      221                      219                      211 
## Northern Mariana Islands                     Ohio                 Oklahoma 
##                      194                      213                      216 
##                   Oregon             Pennsylvania              Puerto Rico 
##                      223                      216                      209 
##             Rhode Island           South Carolina             South Dakota 
##                      221                      216                      212 
##                Tennessee                    Texas                     Utah 
##                      217                      239                      226 
##                  Vermont           Virgin Islands                 Virginia 
##                      215                      208                      215 
##               Washington            West Virginia                Wisconsin 
##                      261                      205                      246 
##                  Wyoming 
##                      211
```

Two things to point out here: (1) not all of our "states" are technically states (e.g., Puerto Rico, District of Columbia, Virgin Islands, Northern Mariana Islands, Guam). I prefer to think of this as the *NYT* data scientist team quietly reminding us that the United States maintains a number of colonial properties without formal political representation! The second thing (2) is that not all states have the same number of observations/rows. You can probably figure out exactly why this might be the case from the documentation of the data sources and or from thinking more carefully about the context (e.g., some states had cases much earlier in 2020 than others). Anyhow, just some things to be aware of as we move forward with our analysis.

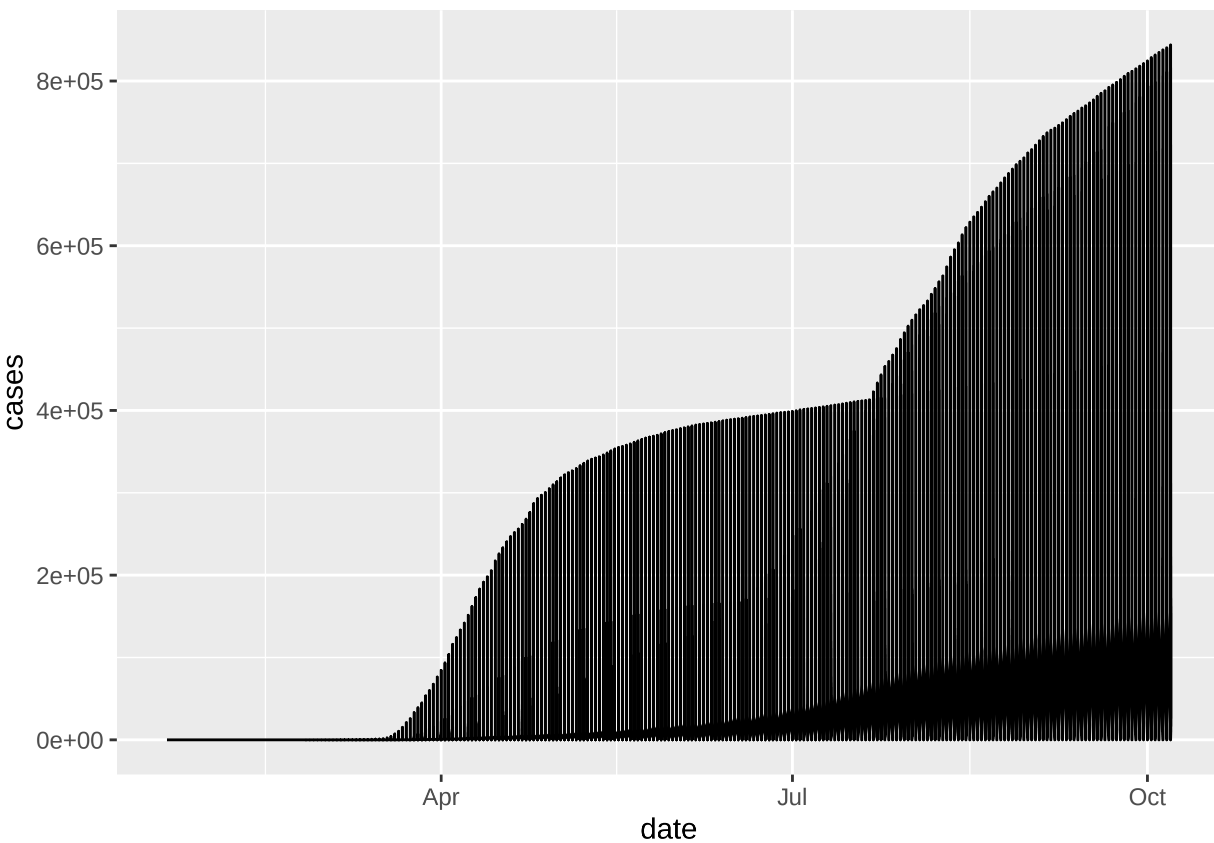
Plotting a univariate time series

A univariate time series is just a fancy term for a plot of a single variable for which you have repeated observations collected over time. I recommend using `geom_path()` (that's a hyperlink to the documentation) to create univariate time series plots. Specifically, I'll call `geom_line()`, which is a specialized (masked) 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 values of my y-values over time.

For a univariate example, let's build a plot of weekly case counts in Illinois.

I can start by just plotting the cumulative cases for all of the states and work towards the specific plot we want from there:

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



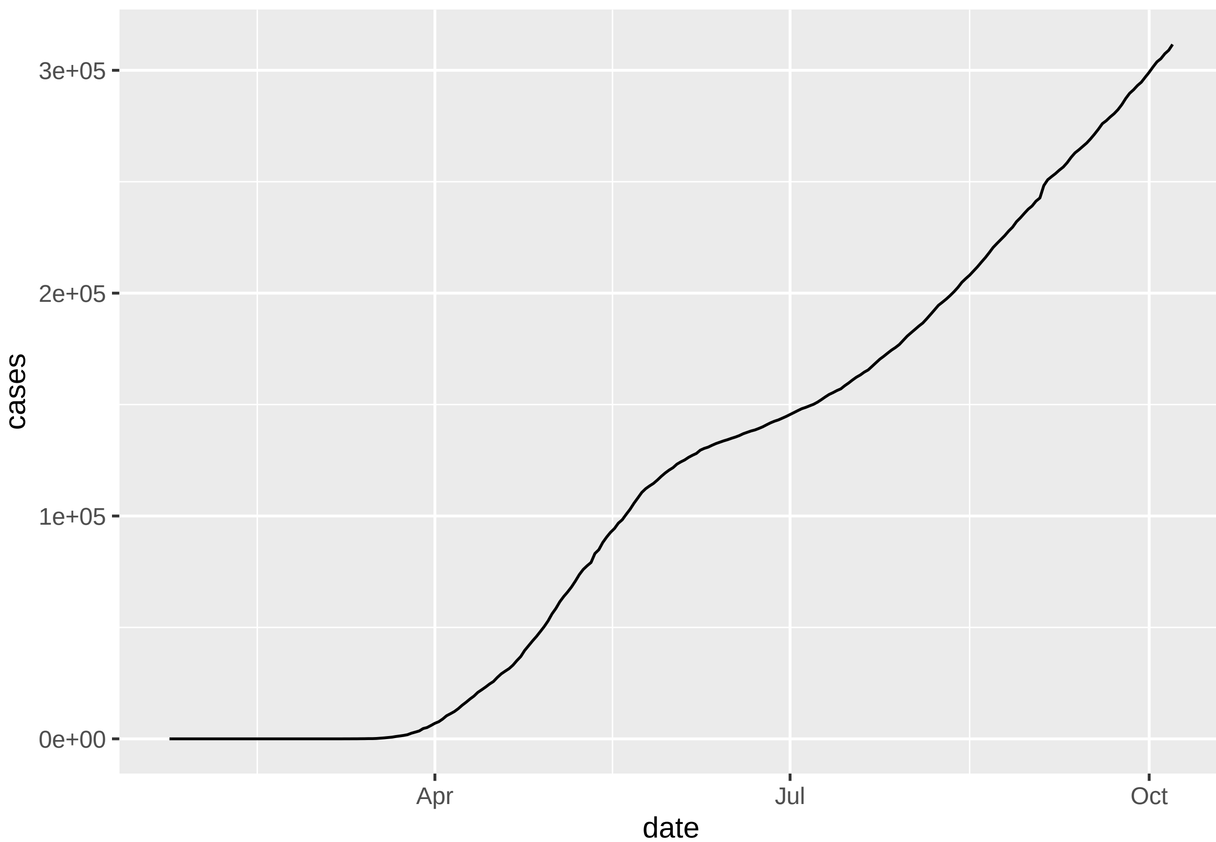
Notice that ggplot handles the `date` variable quite well by default! It recognizes the units of time and generates axis labels in terms of months. Also notice that ggplot handles the axis labels for the `cases` variable. . . less well. I don't know about you, but my brain doesn't parse scientific notation quickly/easily. Finally, the fact that this figure incorporates all the state-level observations as cumulative counts means that there is just a huge clutter of points/lines in this figure. It's impossible to really figure out what's going on, much less learn anything other than the cumulative number of cases within states appears to have increased over time (thanks for nothing, ggplot).

Tidying some timeseries data

Okay, let's get to work cleaning 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 and much clearer. It also looks plausibly accurate (it's always good to sanity check your data visualizations as you go—weird anomalies in a graph are usually a good indicator of something weird happening in the underlying code and/or data.

Now onwards to converting my cumulative case counts into weekly case counts. When I wrote this tutorial, the first way I thought to do this involved making calls to the Tidyverse `mutate`, `group_by`, and `summarize` verbs. After a little trial and error, I got it to work with the following code (which I'll walk through in detail below):

```
il_weekly_cases <- d %>%
  filter(state == "Illinois") %>%
  mutate(
```

```r
    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 so let's go through it verb-by-verb.

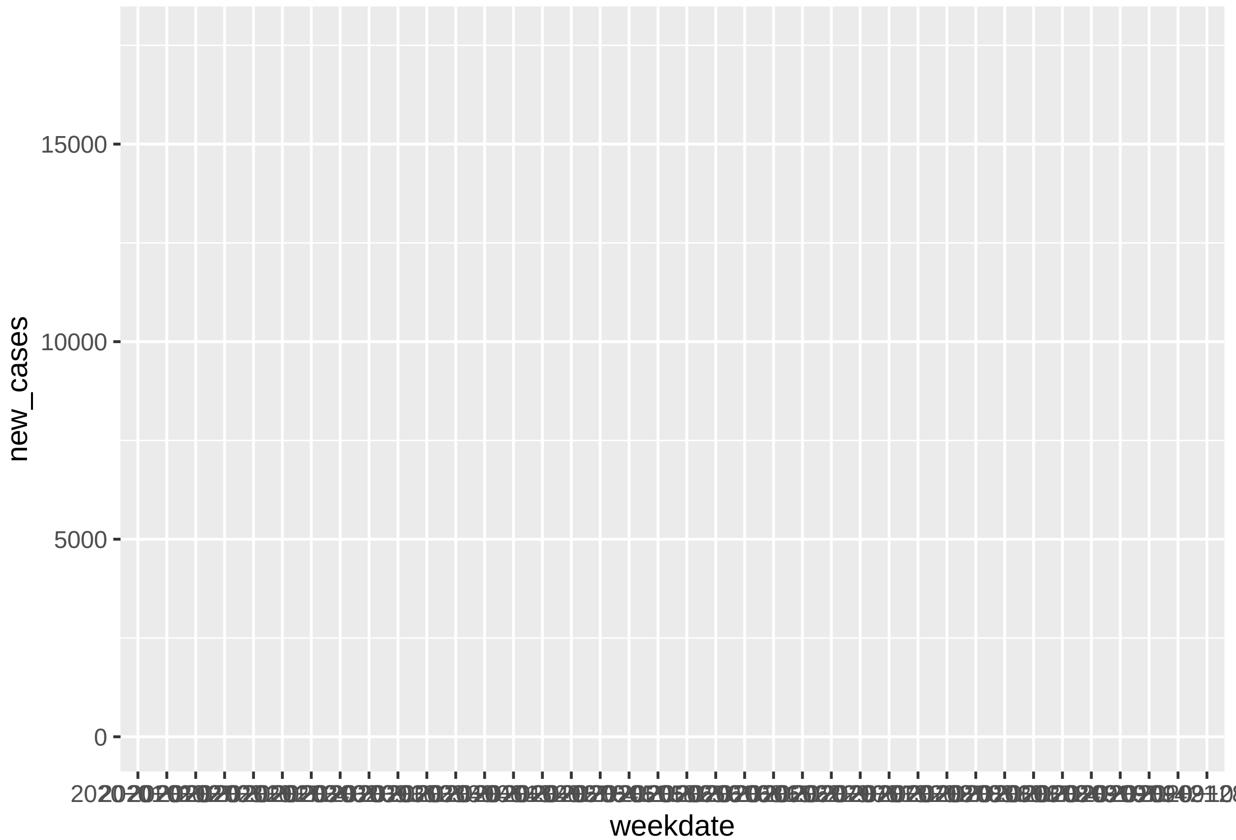
First, I `filter` my cases to restrict the set to Illinois data. Then I use `mutate` to create a `diff_cases` variable that disaggregates the cumulative values of `cases` (read the documentation for `diff` to learn more about this one). Differenced values alone wouldn't produce the correct number of items (try running `length(1:10)` and compare that with `length(diff(1:10, 1))` to see what I mean), so I store the first value of my `cases` variable and then append the differenced values after that (Note that this assumes and takes advantage of the fact that the data is sorted by date. I could add a call to `arrange(-desc())` before doing my mutation to ensure the correct ordering, but won't bother with that for now). 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. This is essentially creating conditional groupings of the data that I can then summarize in my next command.

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). The result is a brand new two-column tibble consisting of weekdates and weekly counts of new cases. Excellent!

Okay, let's see about plotting this now:

```r
il_weekly_cases %>%
  ggplot(aes(weekdate, new_cases)) +
  geom_line()
```



Hmm. looks like I have a problem here. My first guess is that there's something funny going on with my `weekdate` variable because it looks very different on the x-axis. Let's troubleshoot:

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

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

Whoops. Indeed, I need to convert that `weekdate` variable back 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 coercing `weekdate` to a character vector and then coercing that into a date using `as.Date` (and remember that it is sometimes easier to read these "nested" commands from the inside-out).

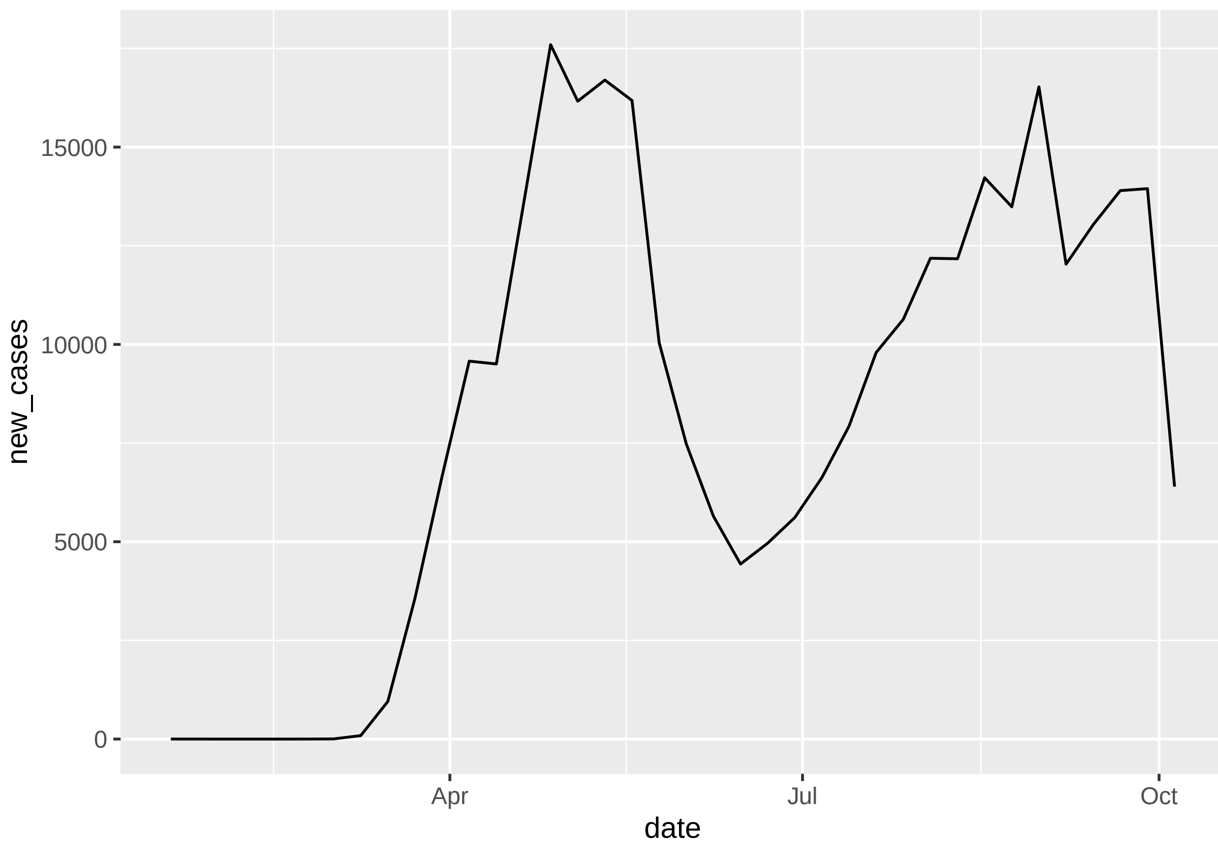
```
il_weekly_cases$date <- as.Date(as.character((il_weekly_cases$weekdate)))
il_weekly_cases
```

```
## # 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 for plotting now:

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

plot1
```



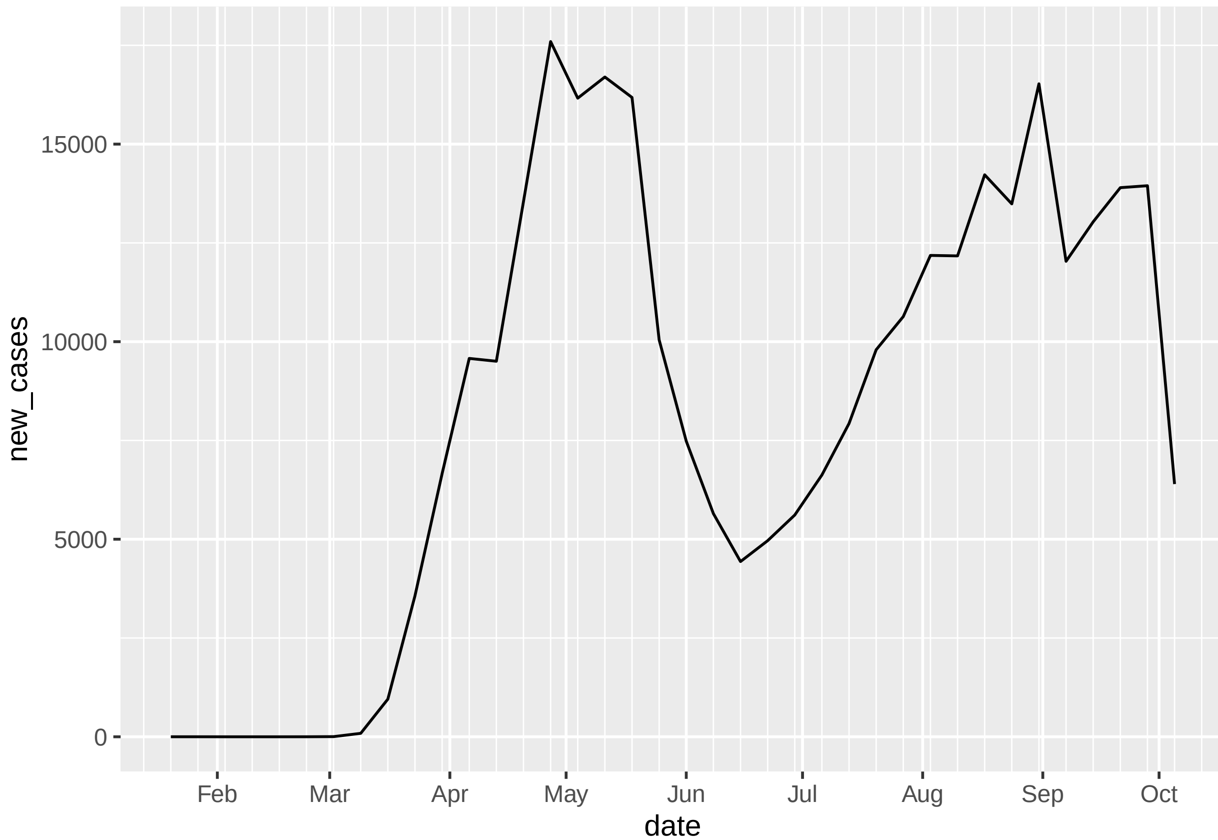
Much better! Notice that the final week of the data appears to fall off a cliff. That's just an artifact of the way that the *NYT* has published the data for part of the most recent week. Once it updates, the case count probably won't tumble like that (yikes).

Working on ggplot axis labels, titles, and scales

Now we can style the plot. As I mentioned briefly in class `ggplot2` treats labels, titles, and scales as "layers" within it's "grammar of graphics" (that sound you hear is me 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. Each of those have documentation and should appear on the `ggplot2` cheatsheet available via RStudio/Tidyverse.

To start, let's see whether there might be any way I want to improve the x-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 ("minor_breaks") that shows the weeks? Also, I like the date labels along the axis as abbreviations of the month names, so I'll keep that with a call to `date_labels`. Here's what all of that looks like:

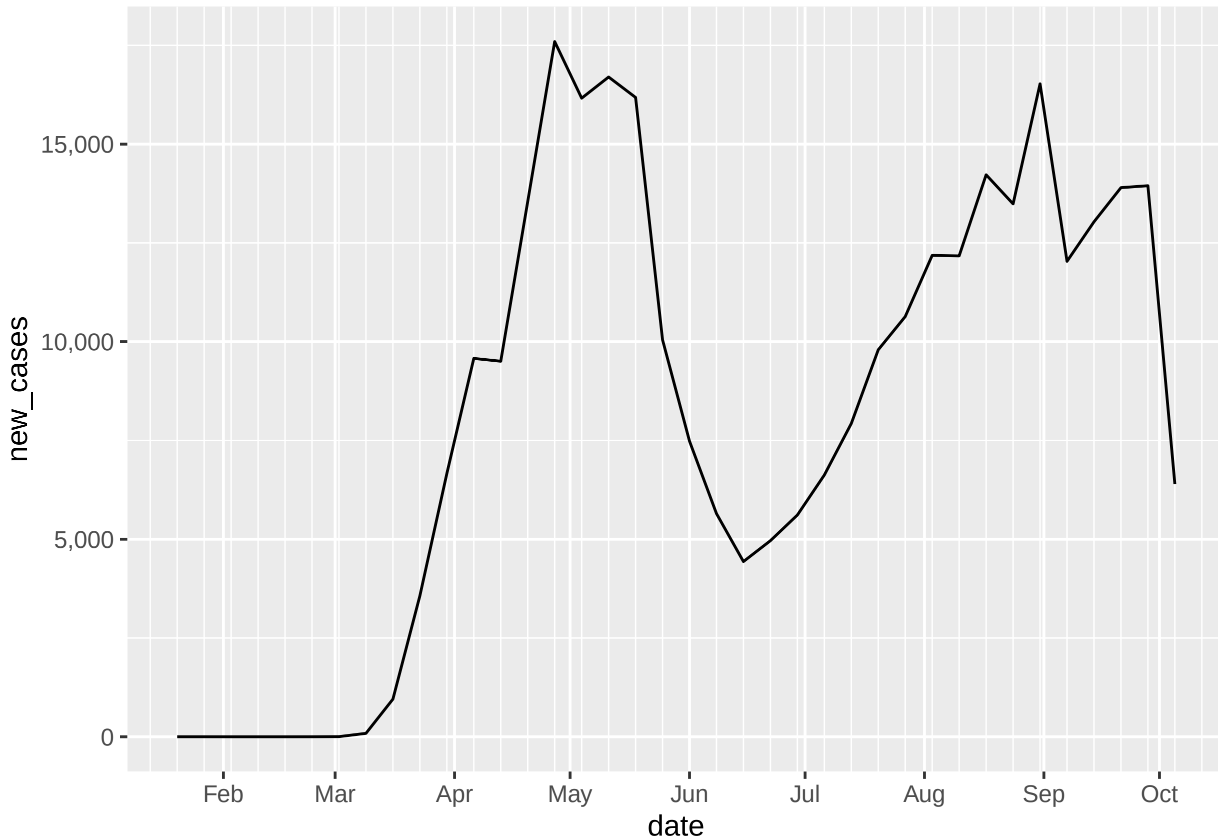
```
plot2 <- plot1 + scale_x_date(date_labels = "%b", date_breaks = "1 month", date_minor_breaks = "1 week")
plot2
```



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 on a single plot, adding new layers as I go without losing existing material along the way.

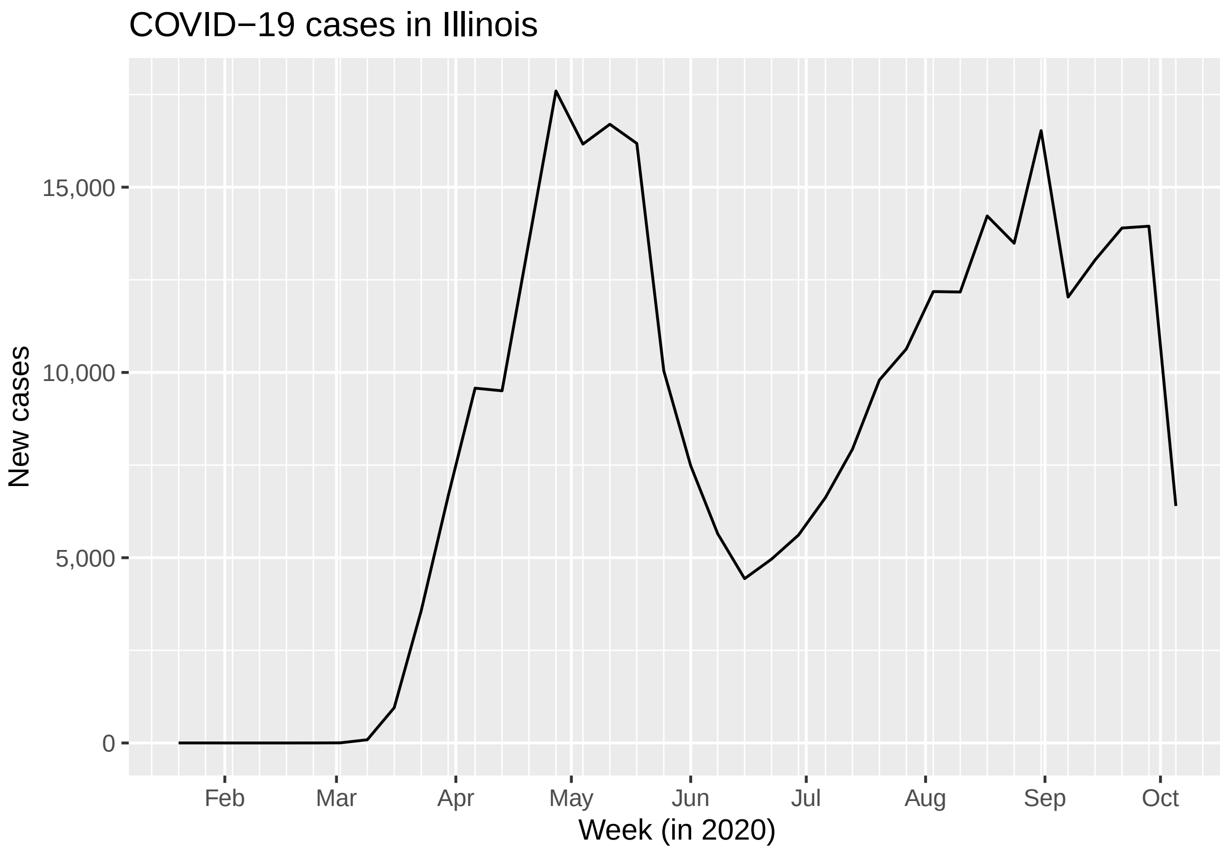
Now I can fix up the y-axis labels a bit using a call to the `labels` argument after I load the `scales` package (why doesn't ggplot support this kind of labeling itself? I have no clue).

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



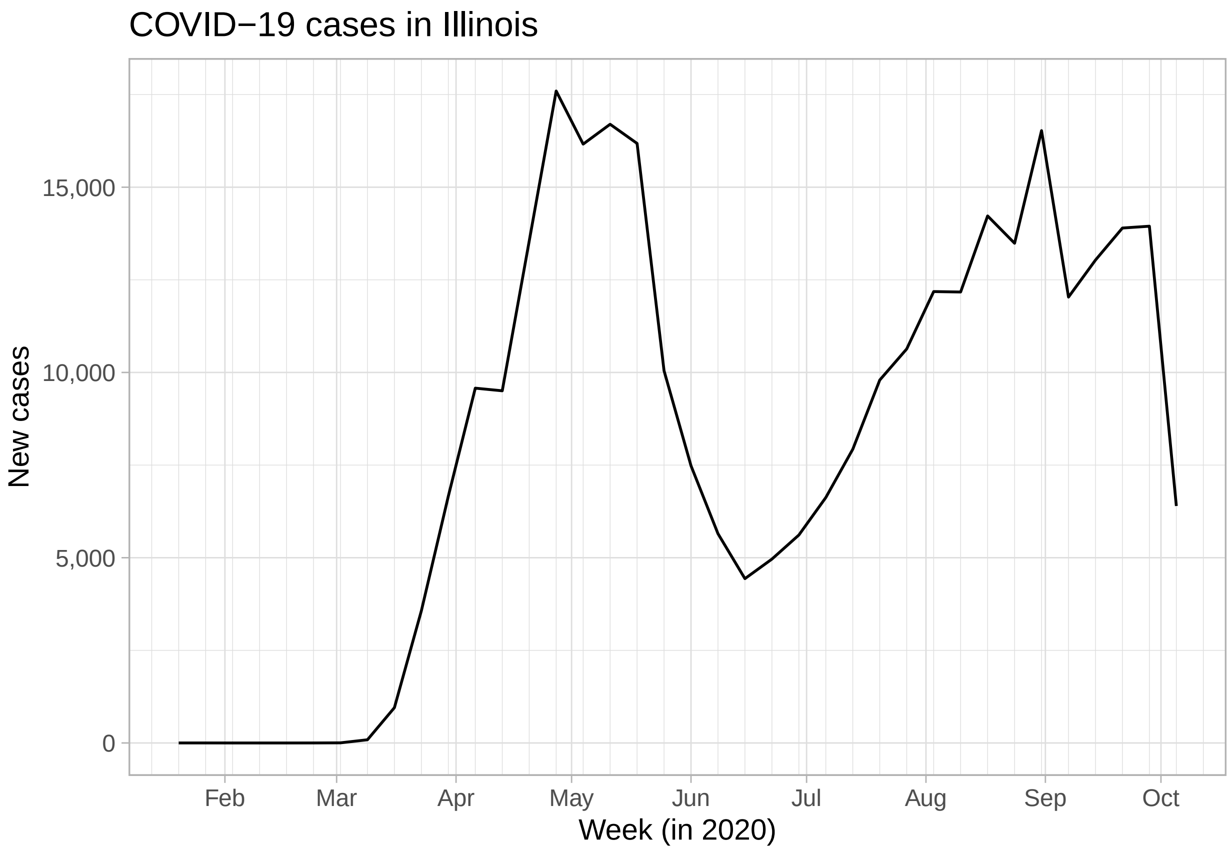
Nearly done. All that's left is a title and better axis names. I'll do that with yet another layer call to `labs`. The arguments here are pretty intuitive.

```
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 I apply that theme as... yet another layer:

```
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.

Multivariate and multidimensional time series plots

Okay, that's a lovely univariate time series plot. Now let's make this more sophisticated and interesting by incorporating more data, more dimensions, and more variables. In order to do that, I want to start with a little detour into data structures. Try to stay with me—this turns out to be super important for working more efficiently with tools like ggplot as well as learning to manage more complex statistical analysis strategies (that we won't really cover in the course, but so be it).

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

So now you want 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 "longer" 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 distinct variable (for which we usually have no more than one value attributed to any unit/observation). This often results in wider format data that is great for many things. However, it turns out that longer 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 pretty "long" format. Each row is a week and each column is a variable unique to that week (okay, I could consolidate my `weekdate` and `date` columns into just one, but that's not really the point here. The idea is that there's minimal redundant information in the rows and in the columns).

Our original dataframe was also pretty "long":

```
d
```

```
## # A tibble: 12,059 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 12,049 more rows
```

Here we have multiple observations per state (I think I would say the units or rows correspond to "state-dates" or something like that). It's not as "long" as possible, though, because we also have multiple columns corresponding to the two variables of interest: `cases` and `deaths`.

For the purposes of producing a multi-state and multivariate set of plots, the most important thing I want to do is consolidate my dataset into a format where I have the following columns: `date` (collapsed into weeks), `state`, `variable` (which will either have a value of `new cases` or `new deaths`), and a column for `value` that will hold the corresponding state-week count for the variable in each row. If that doesn't make sense, don't worry, we'll get there soon enough.

Doing this involves a different approach to tidying up my data. I'll start by dropping the step where I filtered by `state=="Illinois"` and replacing it with 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.

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

```
  ) %>%
  select(state, cases, deaths, weekdate)
weekly
```

```
## # A tibble: 12,059 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 12,049 more rows
```

Now I've got multiple observations for each state-week spread across multiple rows (because my rows were structured around a more granular measure of time). My next move is to collapse these into a single observation for each state-week. Remember that my `cases` and `deaths` variables are still cumulative counts, so as I do this aggregation by week I will only need to store the maximum value for each state-week in order to calculate the number of new cases per state-week.

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

Notice that the call to `group_by` groups by multiple variables. The order here matters! If I reversed it to read `group_by(weekdate, state)` the results would be very different. With the correct ordering, I have things bundled up into state-week sub-groups and then I move on to calculate the maximum value of cumulative cases within each bundle.

Next, I can fix up my `weekdate` variable again so that it is a Date object.

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

This will allow me to do some sorting within my state-week bundles to ensure things are in the proper order before I convert my weekly cumulative case count into weekly new case counts.

```
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
```

We're much closer to our goal now!

I can go ahead and drop the cumulative cases and deaths columns with a call to `select` in my next step. Then the big next (and nearly final) step is to "pivot" the data to organize the `new_cases` and `new_deaths` measures in the way I described above. To manage this, I'll use the `pivot_longer()` function (part of the `tidyr` package from the tidyverse):

```
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
```

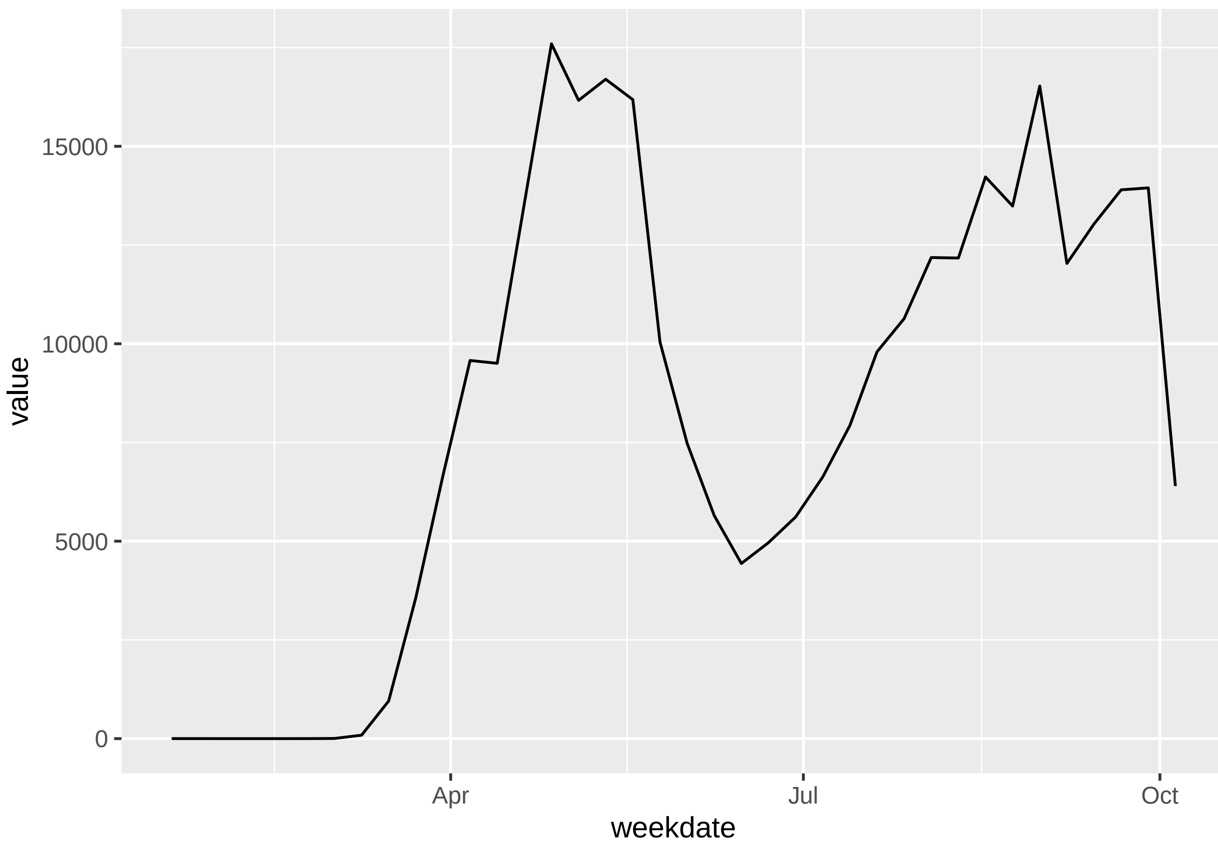
Can you see what that did? I now have two rows of data for every state-week. One row contains a value for `new_cases` and one contains a value for `new_deaths`. Both of those variables have been "pivoted" into a single `variable` column and their corresponding values recorded in another new column. Note that this makes our dataframe a little longer even though it does not technically reduce the "width" of this particular dataset (because we've taken two columns and pivoted them to create. . . two different columns). However, consider that we could accommodate as many additional numerical variables and values as we might like in this manner and you can start to see how this pivoting step could result in much longer data (the length becomes a function of the number of units in your dataset and the variables you include in your pivoting step).

Before we move forward I'm also going to clean up the values of `variable`. This turns out to be helpful later on when we're plotting, but makes more sense to implement here before I start creating any plot layers.

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

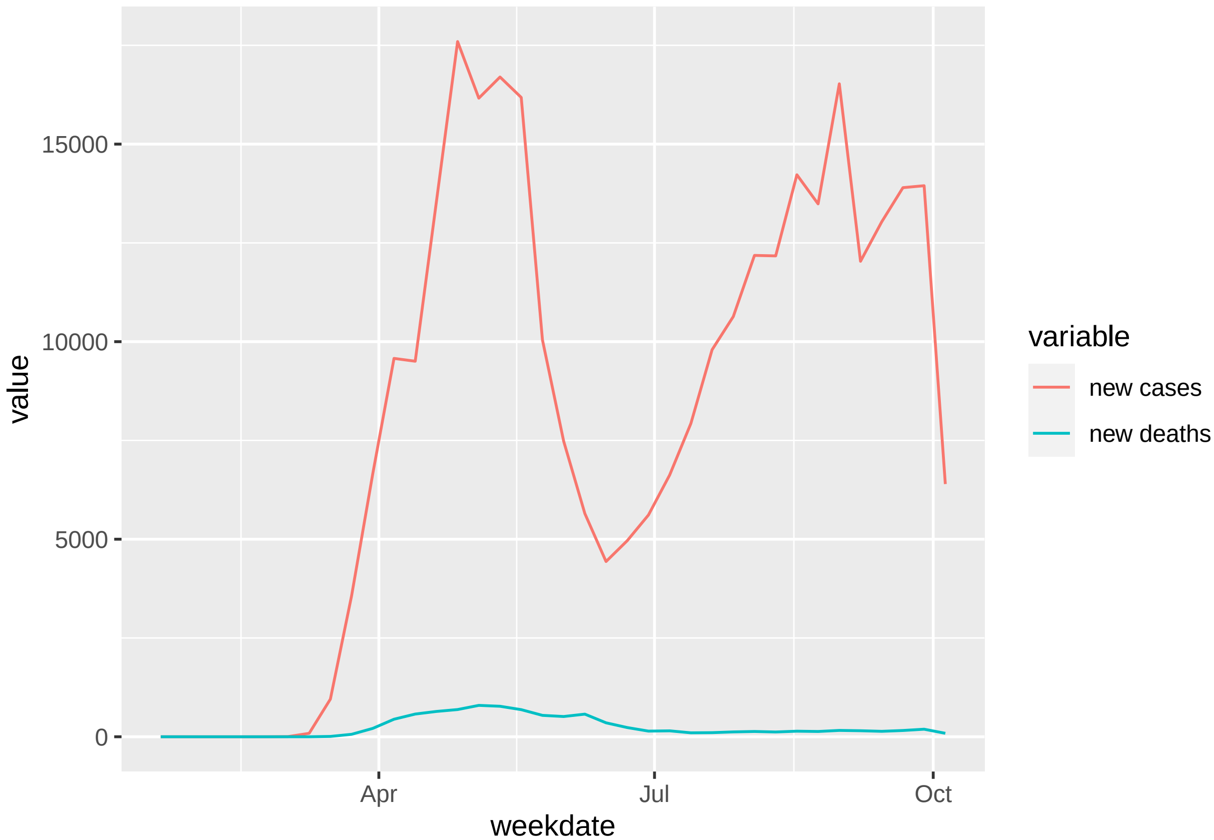
Okay, prepared with my `long_weekly` tibble, I'm now ready to generate some more interesting and multidimensional plots. Let's start with the same univariate time series of new cases we made for Illinois before so we can see how to replicate that figure with this new data structure:

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



With our "longer" data format, we can plot Illinois cases against deaths from the same tibble by incorporating a `color=variable` argument :

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

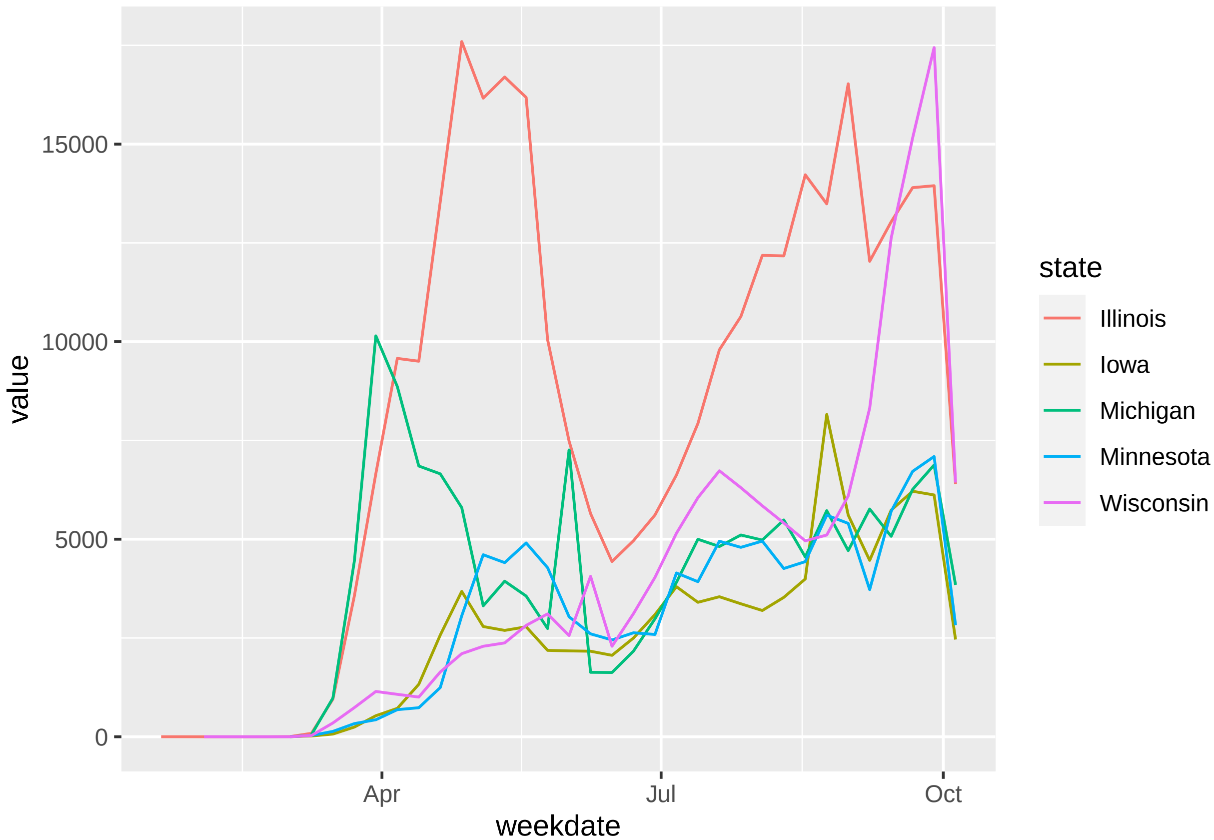


Unfortunately, that plot isn't so great because the death counts are dwarfed by the case counts (thank goodness!).

Now let's compare Illinois case counts against some the neighboring states in the upper midwest:

```
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()
```



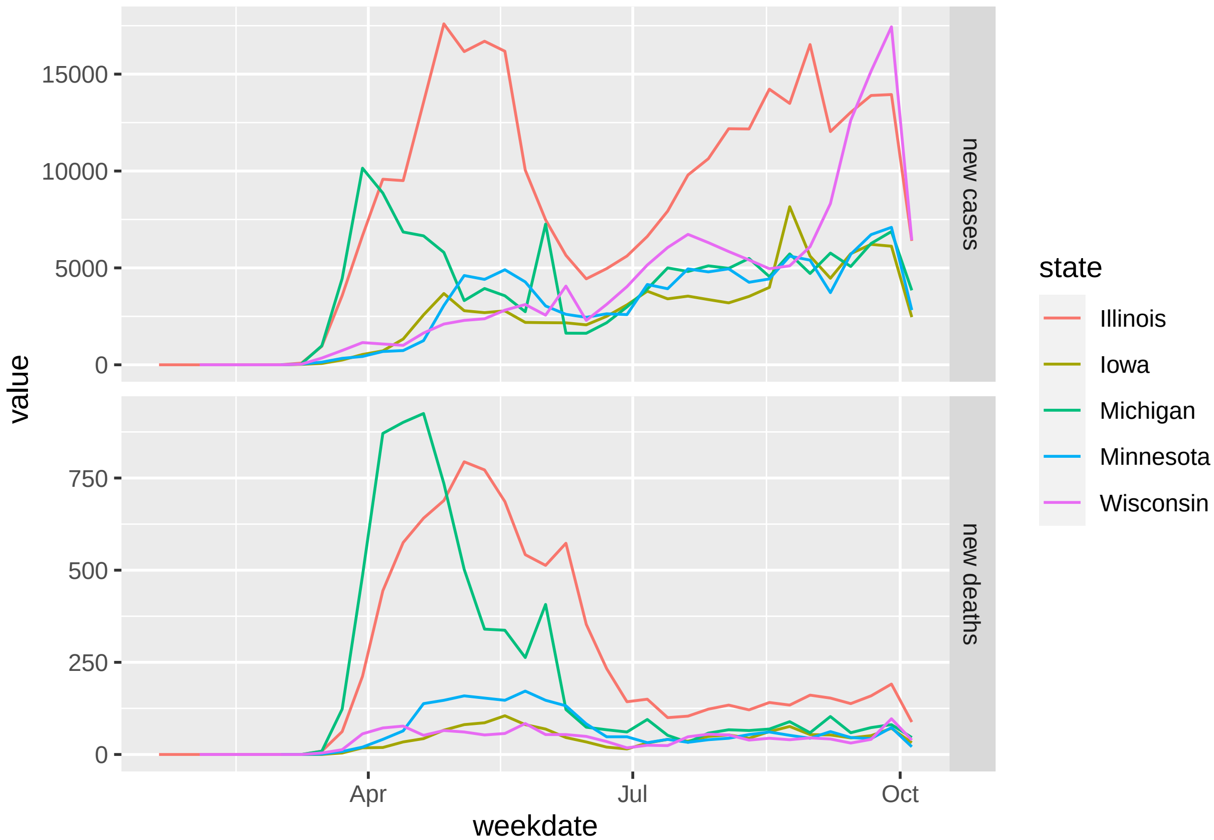
Notice that I use the `%in%` operator to filter for the values of the `state` vector that are "in" the `upper_midwest` vector (see `help(%in%)` for more).

Also notice that we now have ourselves a multivariate time series!

So now how about finding some way to also incorporate those death counts? If I just add them to this same plot we'll run into the same issue we did with the Illinois data because the death counts look tiny plotted on the same scale as the case counts. A good solution in such a situation is to create a second plot for weekly deaths that we can display together with this weekly cases plot that uses a differently scaled y-axis. The ggplot way to do this involves another type of layer called "facets." Here's an example that creates a faceted "grid" (noy much of a grid since there are only two variables or categories we're using to do the faceting) of weekly case counts and deaths for the same five states.

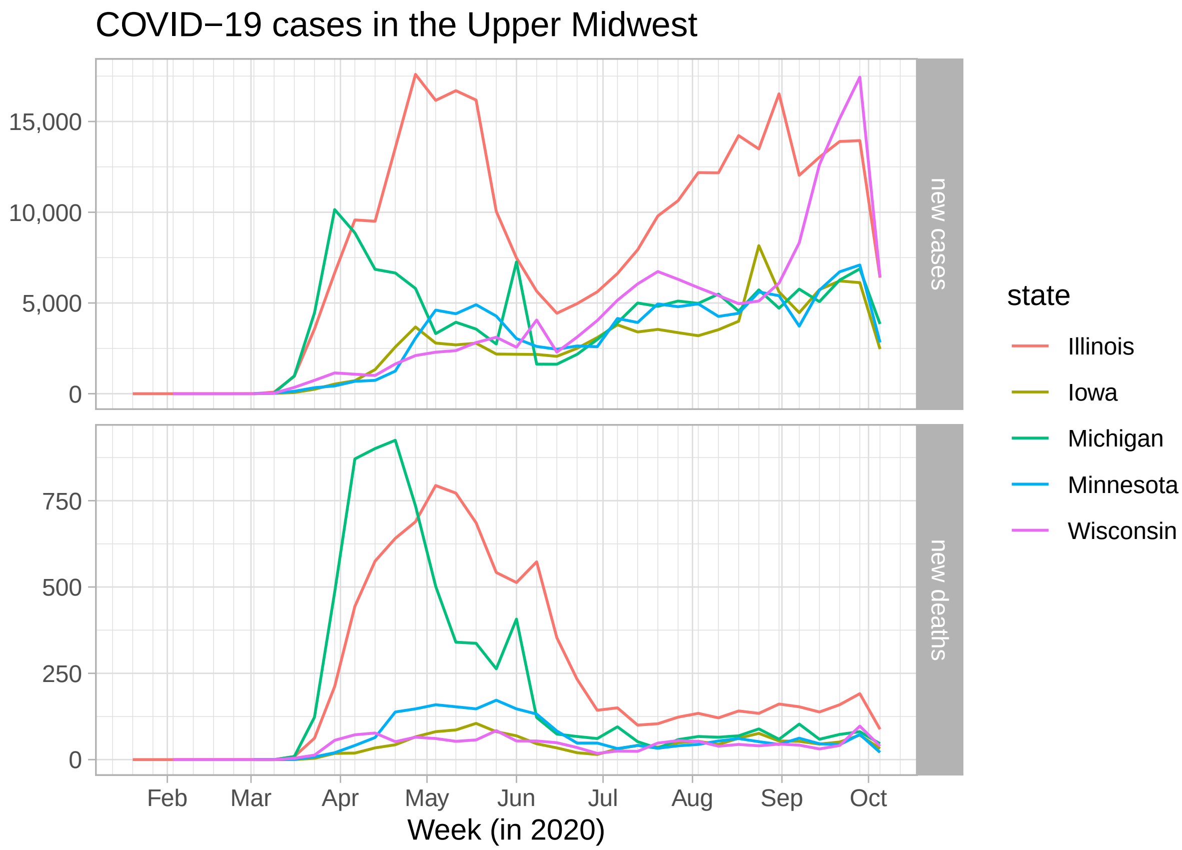
```
midwest_plot <- long_weekly %>%
  filter(state %in% upper_midwest) %>%
  ggplot(aes(weekdate, value, color = state)) +
  geom_line() +
  facet_grid(rows = vars(variable), scales = "free_y")

midwest_plot
```



Nice! 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") +
```



That's it! Mission accomplished. We've got ourselves a nice concise visualization of weekly COVID-19 cases and deaths across five upper midwest states over nearly 8 months of the pandemic.