An Introduction to Map Making in R

Today, we'll talk through the basics of making some maps in R. R has actually become an incredibly powerful map-making tool, and in a future course in this sequence, you'll see how to use its tools to make some more sophisticated maps. But today, we'll show you how to do some simple maps in R that can get you started. We'll mostly work with 2 main map-making packages, `maps` and `mapdata`, to draw maps. We'll also use the tools of the tidyverse, most notably `ggplot` to actually display our maps.

We'll first talk through the basics of how to utilize R's built-in map data, and then how to pair that with other data to show how different variables change across geographic units; such maps are known as choropleth maps.

Drawing a Map of a Country

Let's start at the beginning: let's try to draw a map of the United States of America. We'll do this in a few steps. We'll first load in the libraries (but make sure you've installed them first!).

```
## Load in the libraries
library(maps)
library(mapdata)
```

Now let's learn how to draw the map of the USA. A map is nothing more than a set of connected points making up the boundary of a geographic entity such as a county, state, or country. Map-makers represent these connected points as a set of longitude and latitude coordinates: every point on earth maps to a given longitude (distance east or west of the prime median in Greenwich, England) and a given latitude (distance north or south of the equator). So to draw a map of the USA, I need to know the latitude and longitude of every point along the US coastline.

Happily, this data is contained in the libraries we just loaded into R. While we'll focus on the USA today, R has parallel data for many other nations (and parts of nations) on earth; see the help files for more information. To get the data for the USA, we can run the following command:

```
## Get the data
usa <- map_data("usa")
head(usa)
```

```
##        long      lat group order region subregion
## 1 -101.4078 29.74224     1     1   main      <NA>
## 2 -101.3906 29.74224     1     2   main      <NA>
## 3 -101.3620 29.65056     1     3   main      <NA>
## 4 -101.3505 29.63911     1     4   main      <NA>
## 5 -101.3219 29.63338     1     5   main      <NA>
## 6 -101.3047 29.64484     1     6   main      <NA>
```

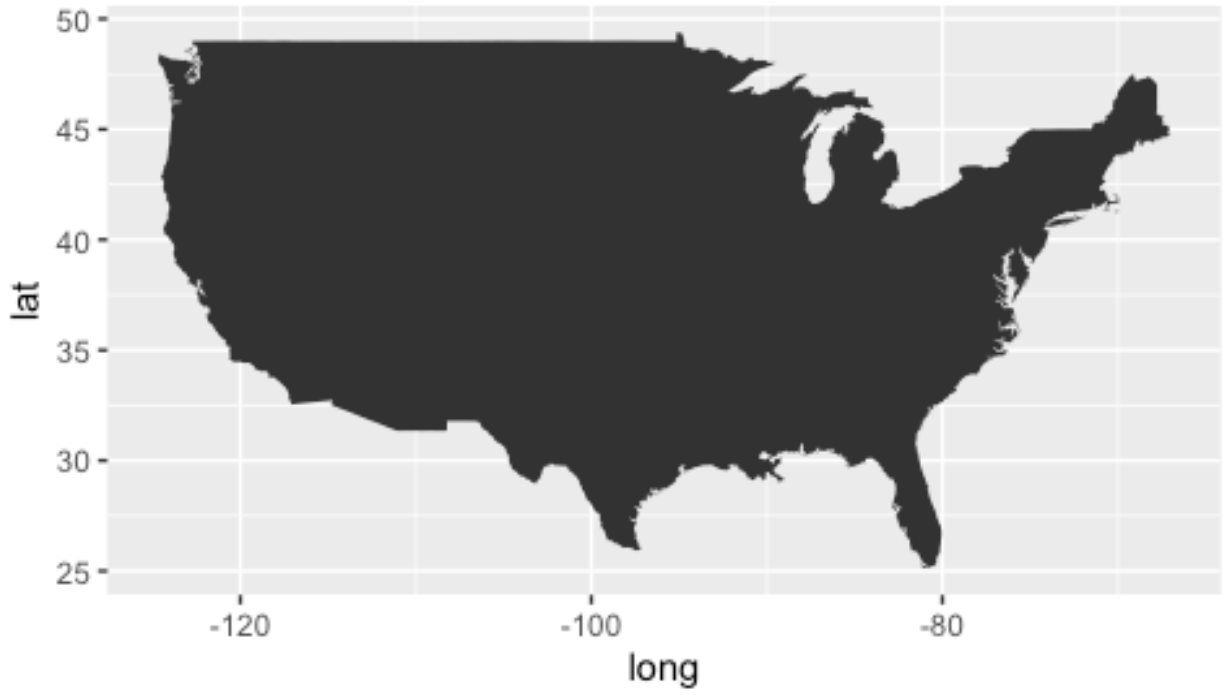
The `map_data` function just pulls out the information for the country in question (here, the US) and puts it into a format suitable for plotting with `ggplot`. Note that there are 6 variables in this dataset.

1. `long`: the longitude of a given boundary point

2. `lat`: the latitude of a given boundary point
3. `group`: This is an argument for ggplot to know how to connect points. Points in a group are connected, but points in different groups are not. In essence, when points are in different groups, `ggplot` lifts the pen (so to speak) when drawing them.
4. `order`: this tells R the order in which to draw the points
5. `region` and `subregion` tells which region to which the points belong. In the U.S., this is their state and county.

So basically, this data tells R how to plot the boundaries of the US (expressed in latitude and longitude) to make a map. Let's now actually plot this graph:

```
## Actually draw the plot
ggplot() +
  geom_polygon(data = usa, aes(x=long, y = lat, group = group)) +
  coord_quickmap()
```



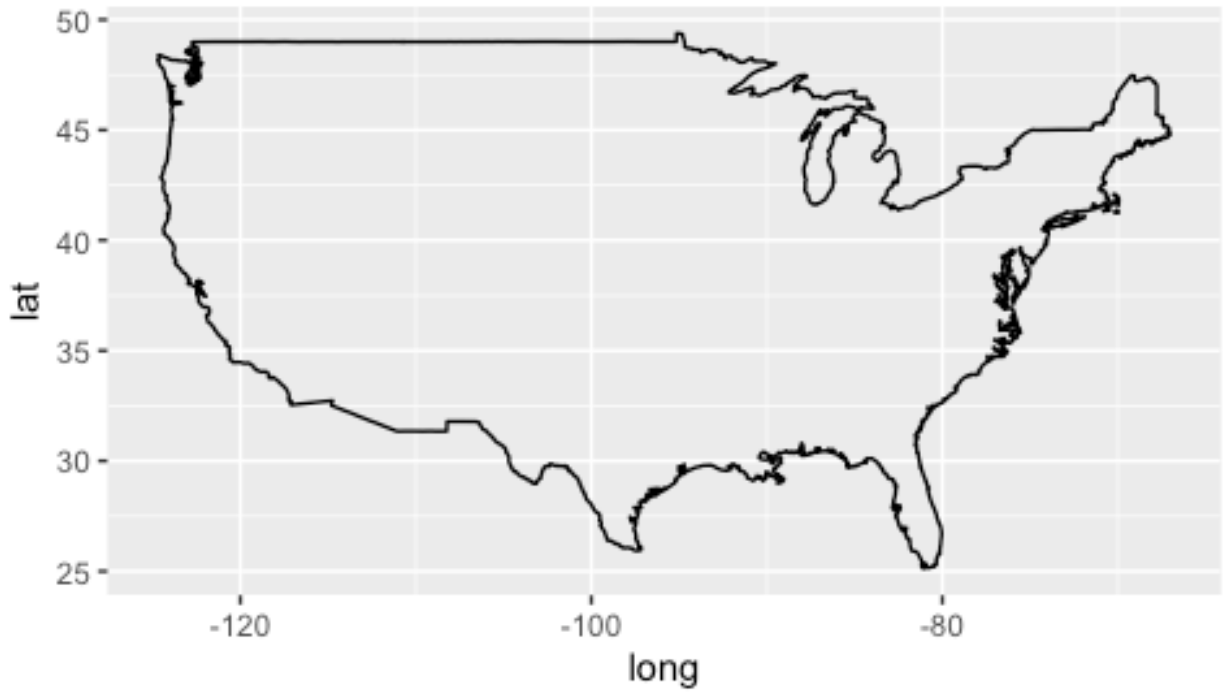
Cool, that gives us our map! The call to `ggplot` should look familiar: it's setting up our plotting environment, and then `geom_polygon()` is our geometry. We can think of a map as a complex polygon determined by the latitude and longitude of a large number of points corresponding to the boundary of the entity we're trying to plot; hence why we use this geometry.

The `coord_quickmap()` tells `ggplot` that we want our map to look like the Mercator projection. Because the earth is round, but our computer screen (or a piece of paper) is flat, you need to use a projection to determine how the map appears on the page (for more on this process, see here). The Mercator projection is the default projection that you've seen in maps throughout your lifetime. If you want another projection, you can use the `coord_map()` function in the `mapproj` library.

By default, `geom_poly()` fills in the polygon (the object you're mapping) with black (it produces a *filled* polygon). If you want to change that, you can edit your call to `geom_polygon`:

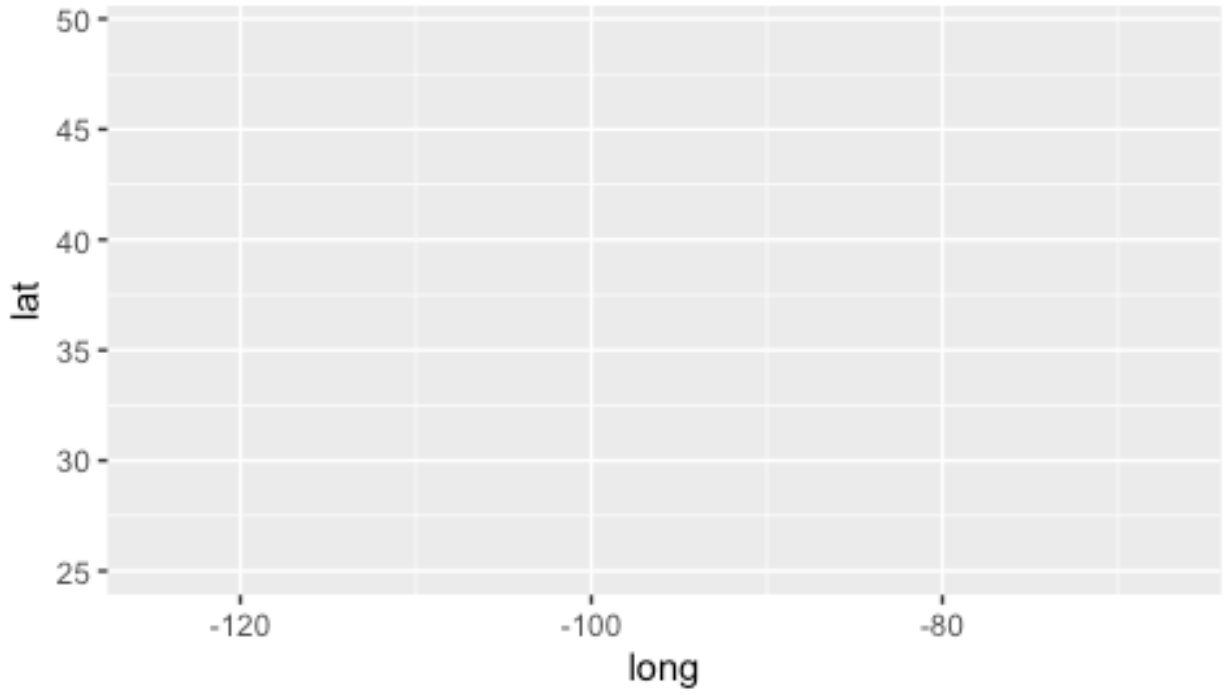
```
## Change the fill to blank
ggplot() +
  geom_polygon(data = usa, aes(x=long, y = lat, group = group), fill=NA,
```

```
color="black") +
  coord_quickmap()
```



It's worth noticing what happens if I run the following code chunk:

```
ggplot() +
  geom_polygon(data = usa, aes(x=long, y = lat, group = group), fill=NA) +
  coord_quickmap()
```



Note that we've just made one tiny change from the call to `geom_polygon()` above: we removed the call to color. Now we get a blank map! Why? Because `geom_polygon()` produces a filled polygon, and if we tell it `fill=NA`, then it draws out polygon with no fill. So if you're not going to fill in the map, then you need to draw the boundary line. The color of that boundary line is assigned by the color parameter.

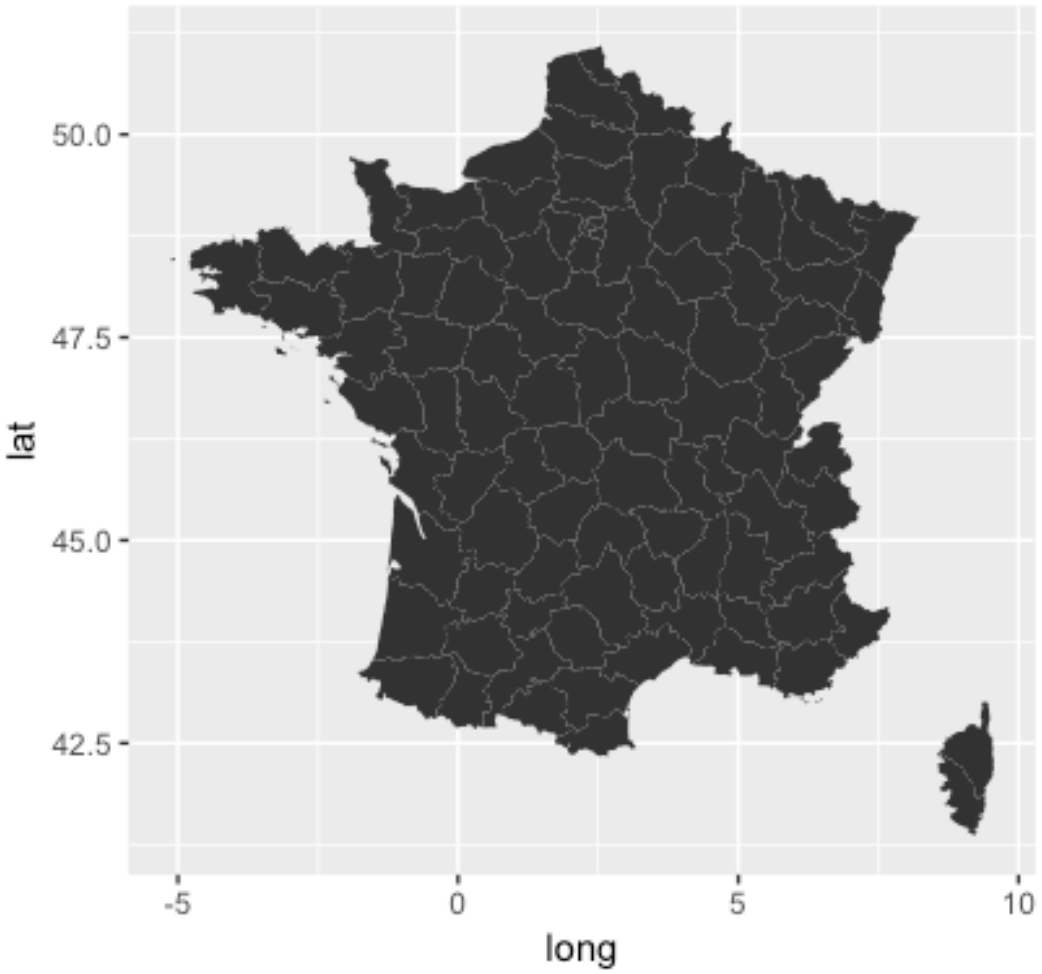
If you want to remove the `ggplot()` background (here, the latitude and longitude lines, since that's what we're plotting), you can use the `theme_void()` option:

```
ggplot() +
  geom_polygon(data = usa, aes(x=long, y = lat, group = group)) +
  coord_quickmap() +
  theme_void()
```

Again, this is an aesthetic choice, and whether you want the regular theme, or this blank white one, depends on your preferences. And of course, you can title the graph (or change other features of it) using the tools we learned in earlier lectures.

Typically, because my work focuses on the United States, I mostly draw maps of the US and various states/counties (we'll see how to do the state/county maps below). But of course, you can draw other countries, most of which are included in the maps package. For example, we can draw a map of France:

```
## Try a map of France
France <- map_data("france")
ggplot() +
  geom_polygon(data = France, aes(x=long, y = lat, group = group)) +
  coord_quickmap()
```



And likewise, you could draw other world countries of interest. Typically, you just reference the country by it's name. For a list of countries and other geographic entities included, you can run the following code chunk:

```
options(max.print=2000)
maps::map("world", namesonly=TRUE, plot=FALSE)
```

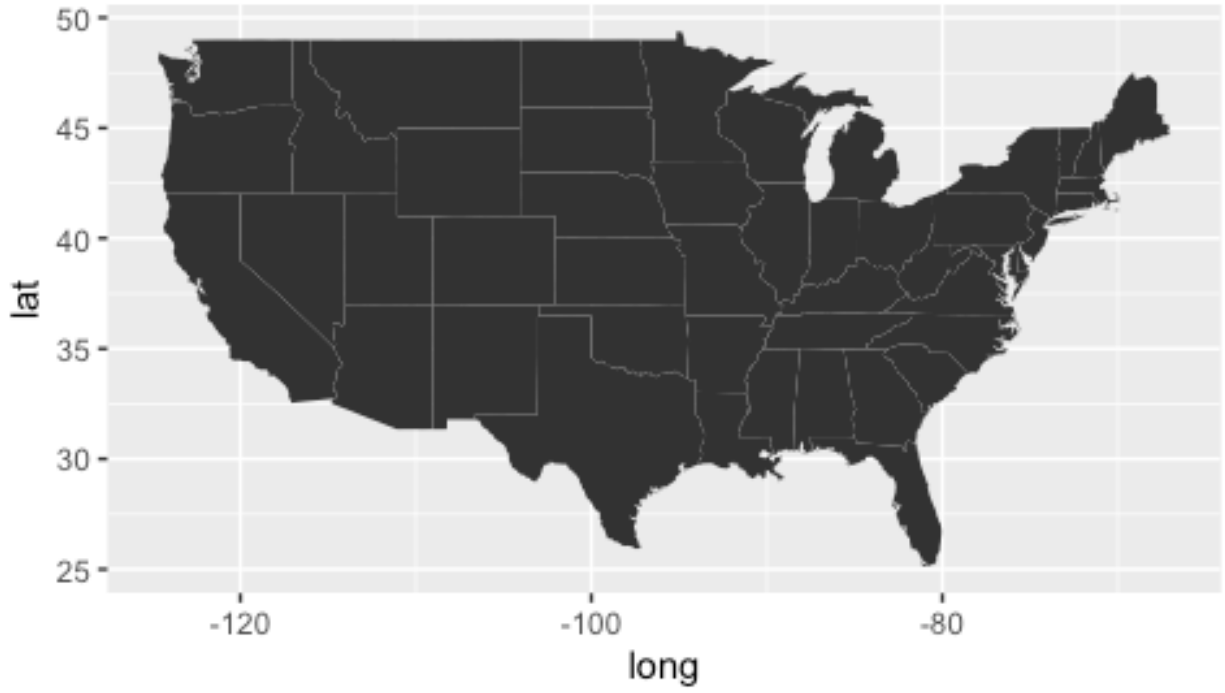
You need the first line since by default, R will only print 1,000 items to the console, and R has 1,627 built in countries and regions included in the maps dataset! It's quite likely that if you want to plot it, it is included in this library.

Drawing Maps of US States and Counties

The `maps` data also can draw maps of states and counties as well within the US. Let's start by seeing a state map in action:

```
## Draw a map of the states
states <- map_data("state")
ggplot(data = states) +
```

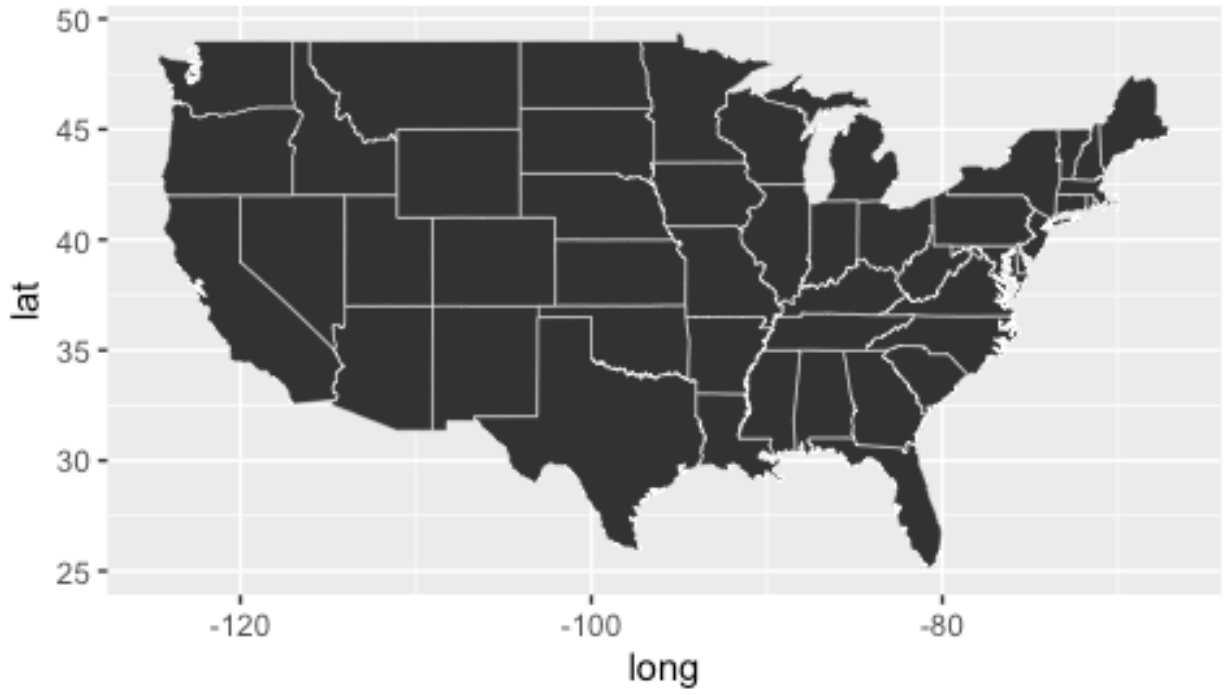
```
  geom_polygon(aes(x = long, y = lat, group = group)) +
  coord_quickmap()
```



Note that again, I'm using the `map_data` function to extract the data, and note that the `map` library is setup to know that the "states" argument will return the list of US states. The rest of the code is as above.

In this map, I find it hard to see the boundaries between states, so I'd like to emphasize those a bit more. I can do that by slightly altering my call to ggplot:

```
## better differentiate the states
ggplot(data = states) +
  geom_polygon(aes(x = long, y = lat, group = group), col="white", lwd=0.15)
+
  coord_quickmap()
```



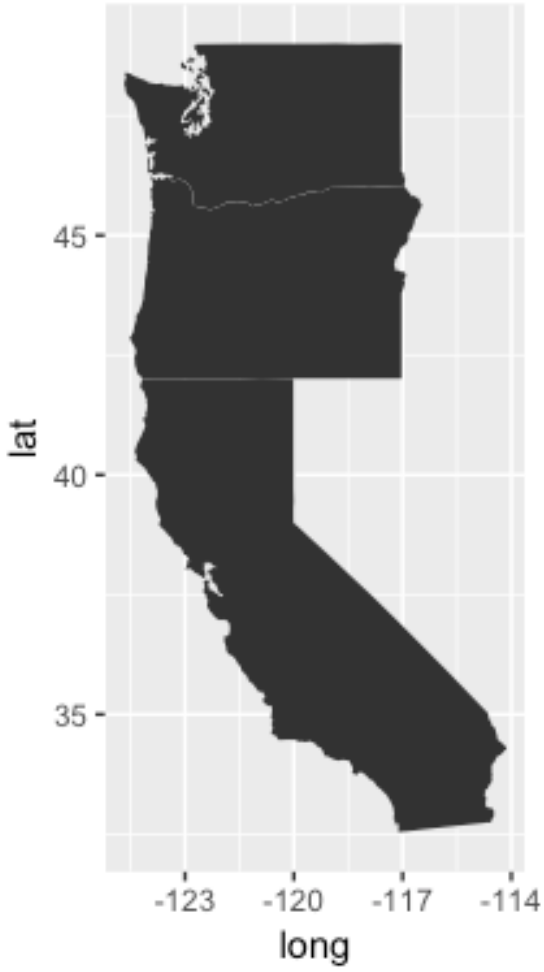
Here, I told R to draw a white line (rather than gray) for the state boundary, and I set it to be quite thin (lwd controls the line width; I played around with that until I found a value that I liked). I think that's a nicer map, but again, that's an aesthetic judgment. You can tweak the parameters as you see fit to make the map *you* think is best.

Note that this is giving me just the "lower 48" states: the states less Alaska and Hawaii. Adding Alaska and Hawaii to a plot is actually somewhat complicated, because the mapping here is done via latitude and longitude, and Alaska and Hawaii are quite far from the other U.S. states, so they'll appear to be quite far away on a map. If you want to include them on a map like this, you have to do some additional coding on your own. A good write-up of how to do this is available on Stack Overflow. You can also use the `tmap` package, but that's beyond the scope of this lecture.

You can also easily print just a subset of states as well:

```
## Draw just the pacific coast states
west_coast <- filter(states, region %in%
c("california","oregon","washington"))
ggplot(data = west_coast) +
```

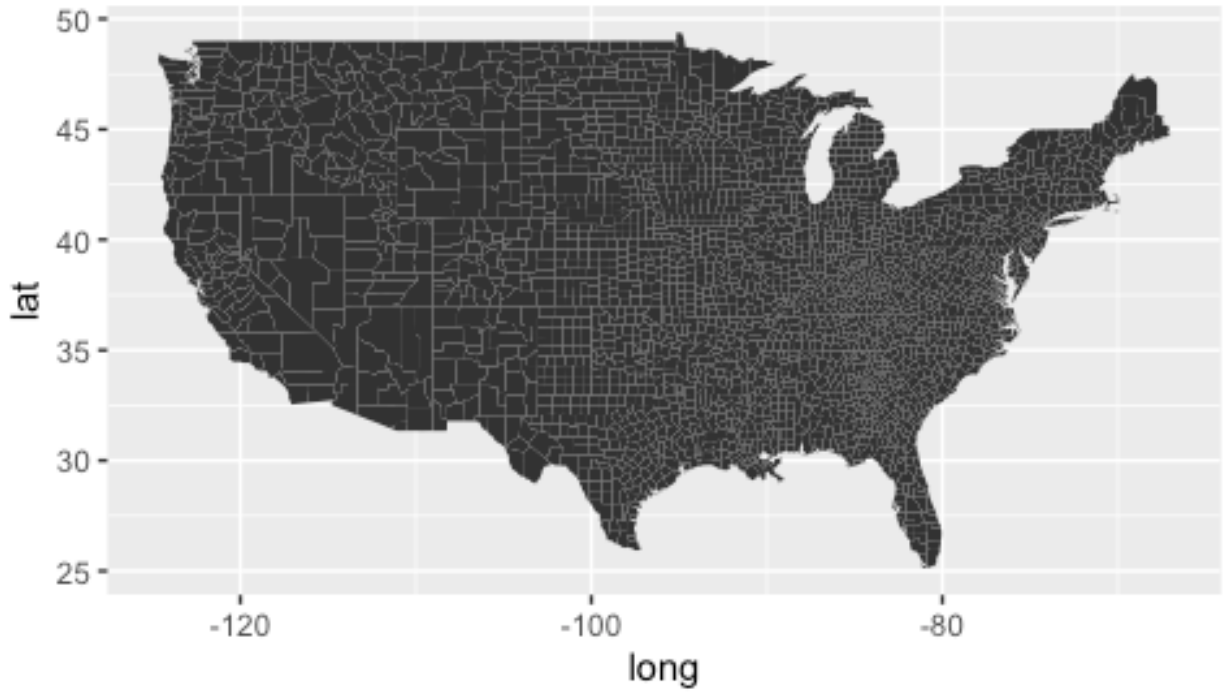
```
  geom_polygon(aes(x = long, y = lat, group = group)) +
  coord_quickmap()
```



Note that the key change here was just to use the `filter` command from `dplyr` to select the data from California, Washington, and Oregon. As we noted above, the `region` variable records U.S. states for the U.S. map data, so we use that variable to select the states of interest.

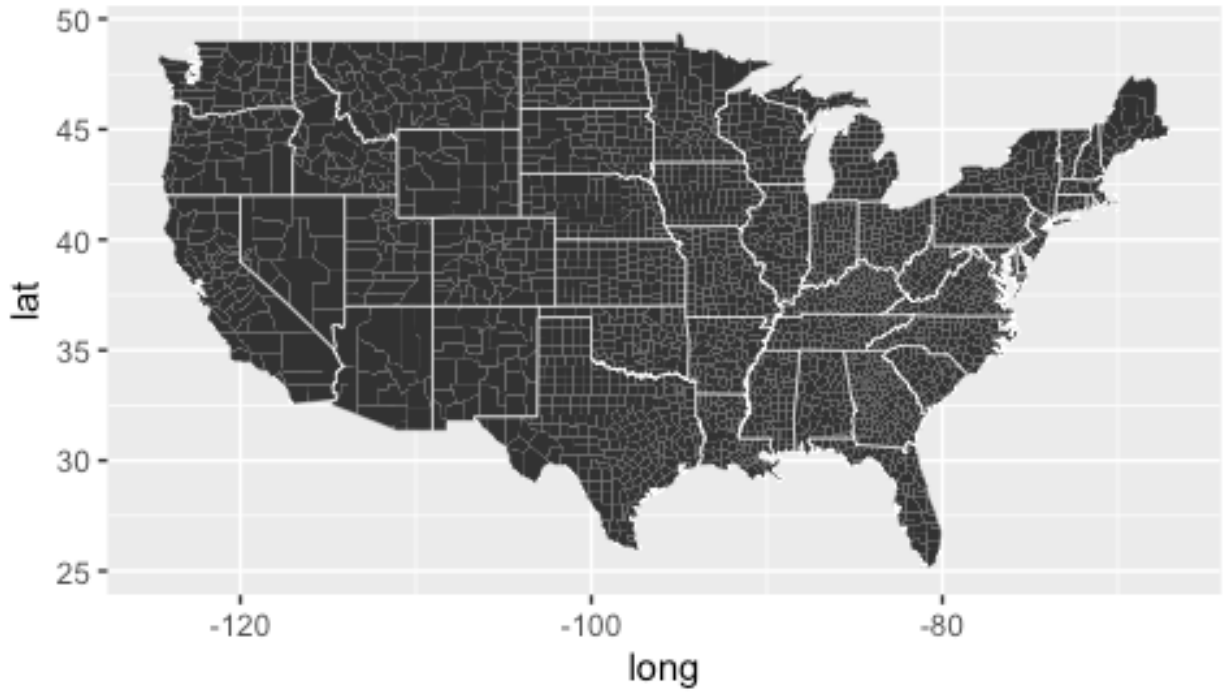
We could also plot a county-level map of the US:

```
## Draw a map of counties
counties <- map_data("county")
ggplot(data = counties) +
  geom_polygon(aes(x = long, y = lat, group = group)) +
  coord_quickmap()
```



But again, part of making a map is thinking about aesthetics. I might try adding the state boundaries in to see if that makes the map easier to read. We can do that with two calls to `geom_polygon()`:

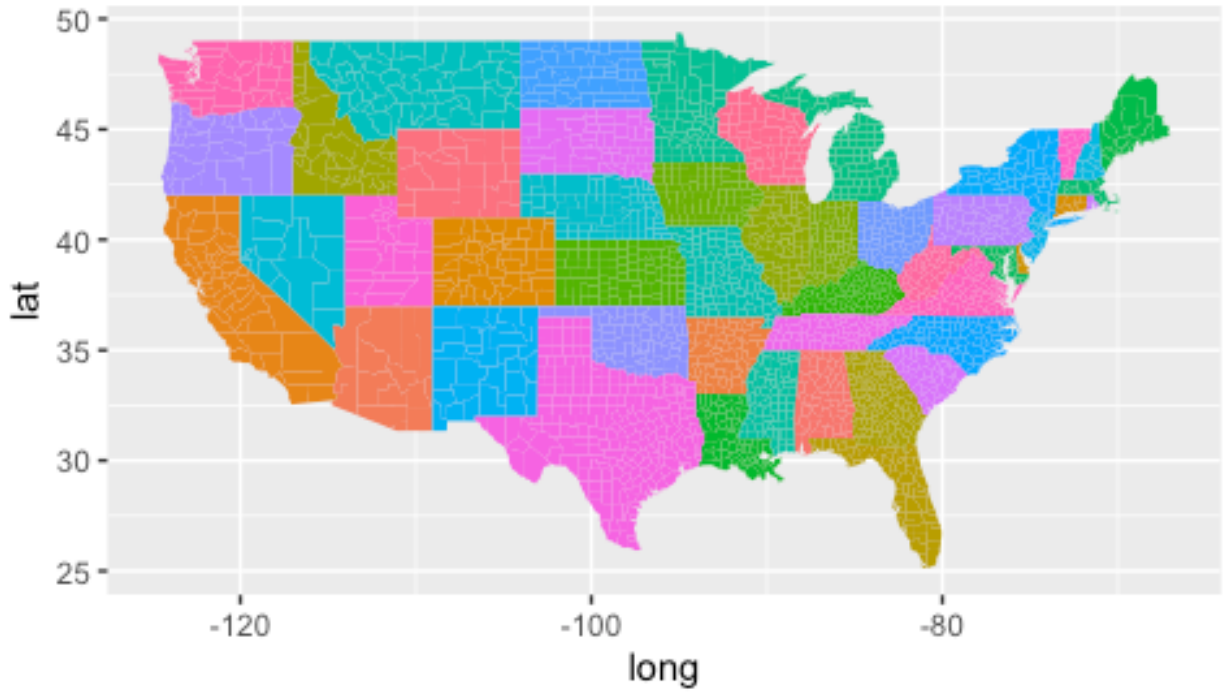
```
## show state boundaries to make it eaiser
ggplot() +
  geom_polygon(data=counties, aes(x=long, y=lat, group=group)) +
  geom_polygon(data=states, aes(x=long, y=lat, group=group), fill=NA,
col="white", lwd=0.15) +
  coord_quickmap()
```



So note that the first call to `geom_polygon()` draws the county-level map, and then the second call to `geom_polygon()` adds the state boundaries on top. Note that in my second call, I set `fill=NA` to avoid filling in the states, which would over-write the county boundaries!

Another option that some people like is to use color, rather than state boundaries, to better differentiate the states:

```
ggplot(data=counties, aes(long,lat, group = group)) +
  geom_polygon(aes(fill = region)) +
  theme(legend.position="none") +
  coord_quickmap()
```



Here, the call to `theme(legend.position="none")` is just suppressing the legend telling us how color maps into state (i.e., that CA is orange, NY is blue, etc.). Given that most people know which state is which, I don't really think it's necessary here.

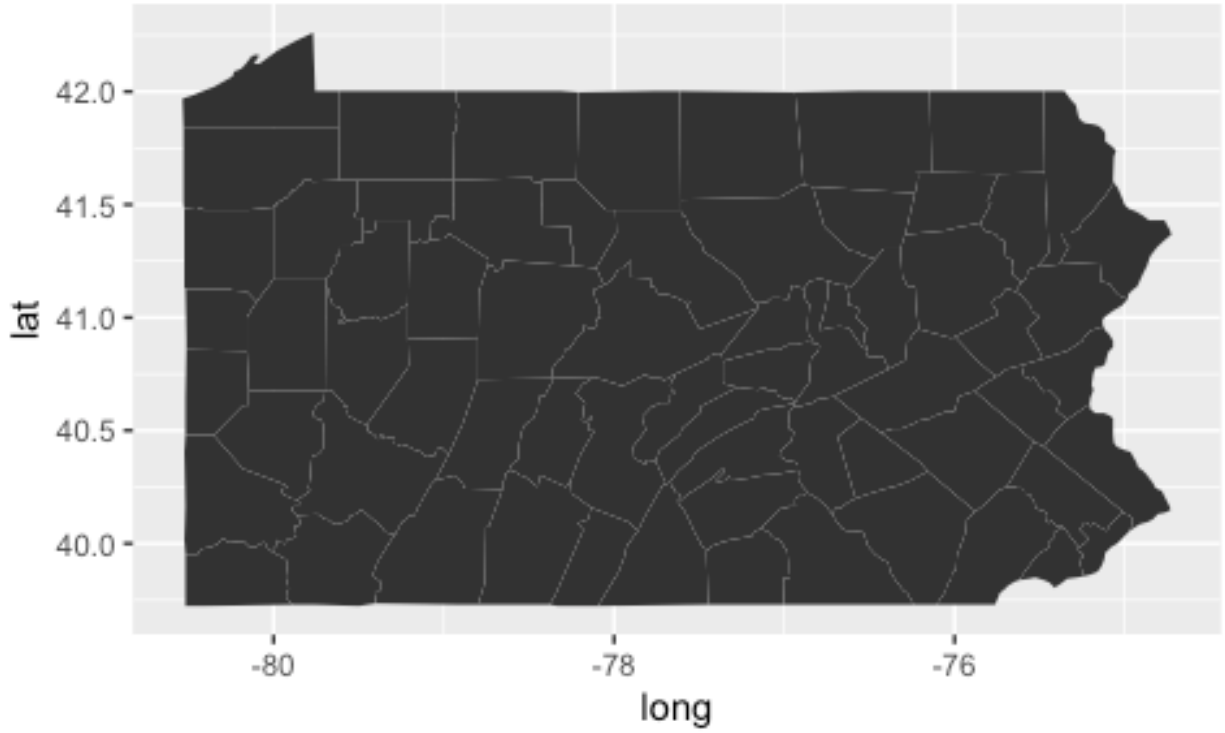
In the end, it's really your call about whether adding the state lines or colors makes it easier to read the map: that's an aesthetic judgement.

Test Yourself: County-level map of Pennsylvania

Question: How would you draw a county-level map of Pennsylvania?

Answer:

```
pa_counties <- filter(counties,region=="pennsylvania")
ggplot() +
  geom_polygon(data=pa_counties, aes(x=long, y=lat, group=group)) +
  coord_quickmap()
```



Note that this is just combining a call to `filter` to get the data just from Pennsylvania, and then plotting it using our same syntax.

Adding in other data

Of course, if this is all we could do, it wouldn't be terribly useful. What makes this useful is that we can create maps showing different state and county-level information. The Census department gathers us a tremendous amount of information about Americans. For example, they provide information about median income (by county) and the percent of residents in each county who reside in poverty, which can be found here. I've downloaded the 2017 data, cleaned it for you, and saved it to Canvas as `census_poverty.csv`.

But now we need a way of telling R how to match up the county data from this new dataset to the county data we're using for map making. We saw how to do this in our lecture on joining data, and one of the crucial components is a key variable that tells us how each observation in our Census data maps onto the observations in our map-making data. When you have state or county data, this is done through something called a FIPS code, which is a code produced by the U.S. government to identify each county or county-equivalent in the United States (FIPS stands for Federal Information Processing Specification).

Happily, the maps package in R comes with a dataset of FIPS codes pre-loaded within it, and our Census data includes FIPS codes as well. We'll use them to join these two datasets together. To do this,we'll need to proceed in a few steps. First, let's take a look at the FIPS code data that comes included in the maps library:

```
data(county.fips)
head(county.fips)
```

```
##   fips        polyname
## 1 1001 alabama,autauga
## 2 1003 alabama,baldwin
## 3 1005 alabama,barbour
## 4 1007    alabama,bibb
## 5 1009  alabama,blount
## 6 1011 alabama,bullock
```

Note here that the state and county are combined into one variable called polyname. Our mapping data also contains state and county data, but in a different format: state and county are in separate variables:

```
## does this match our data?
head(counties)
```

```
##        long      lat group order  region subregion
## 1 -86.50517 32.34920     1     1 alabama   autauga
## 2 -86.53382 32.35493     1     2 alabama   autauga
## 3 -86.54527 32.36639     1     3 alabama   autauga
## 4 -86.55673 32.37785     1     4 alabama   autauga
## 5 -86.57966 32.38357     1     5 alabama   autauga
## 6 -86.59111 32.37785     1     6 alabama   autauga
```

Here, state and county are two separate variables (region and sub-region). So to merge this into our mapping data, we'll need to create this polyname variable in the counties data so we can merge the two together:

```
## do a mutate to match and join
county_with_fips <- counties %>%
  mutate(polyname = paste(region,subregion,sep=",")) %>%
  left_join(county.fips, by="polyname")
head(county_with_fips)
```

```
##        long      lat group order  region subregion        polyname fips
## 1 -86.50517 32.34920     1     1 alabama   autauga alabama,autauga 1001
## 2 -86.53382 32.35493     1     2 alabama   autauga alabama,autauga 1001
## 3 -86.54527 32.36639     1     3 alabama   autauga alabama,autauga 1001
## 4 -86.55673 32.37785     1     4 alabama   autauga alabama,autauga 1001
## 5 -86.57966 32.38357     1     5 alabama   autauga alabama,autauga 1001
## 6 -86.59111 32.37785     1     6 alabama   autauga alabama,autauga 1001
```

Let's parse that code. We took the counties dataset, and then using mutate, we made a new variable (polyname) that is the state (region) and county (subregion) separated by a

comma to match the format of that variable in the `county_fips` data. We then used `left_join` to combine our datasets, matching them up by the polyname variable. For more on what `left_join` does, you can refer back to our handout on combining data.

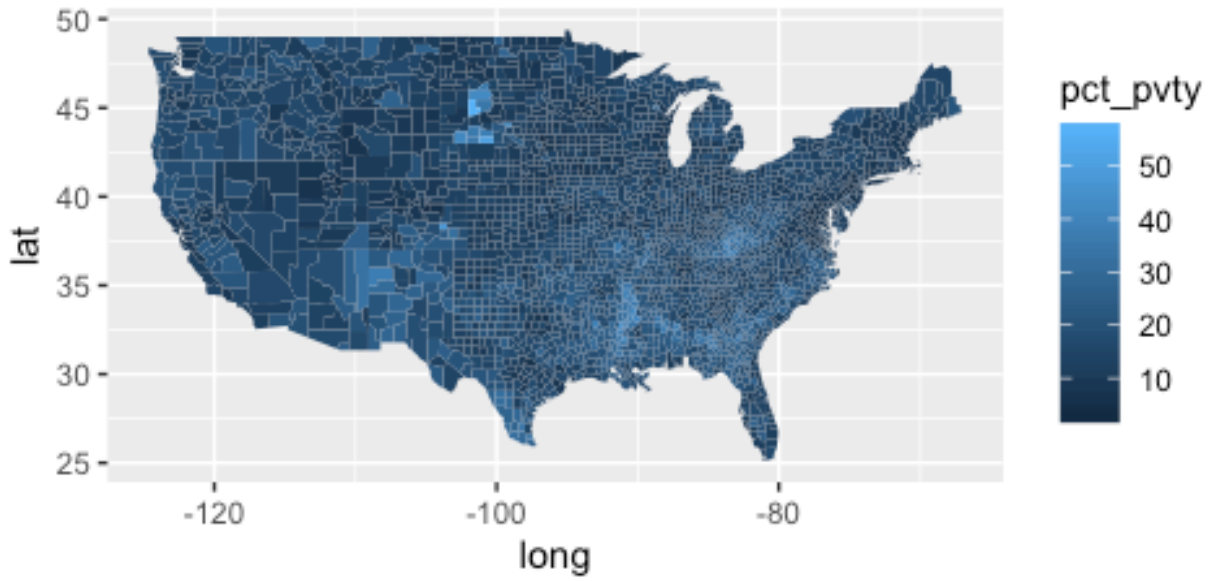
I then used the `head()` command just to print out the first few lines of the dataset so that I could verify that I'd correctly merged in the FIPS codes, and I did. Huzzah!

Now we need to read in the data from the census and merge it with our mapping data:

```
## Read in the new data
setwd("~/Dropbox/DATA101")
census_income <- read_csv(file="Data/Processed/Census_Poverty.csv")
## merge with our data
county_with_income <- inner_join(county_with_fips,census_income,
                                 by=c("fips"="fips_code"))
```

Happily, this is all syntax we've seen before: `read_csv` to read in our data, and `inner_join` to combine our datasets (with `by` indicating the variable for matching in each dataset; please review the handout on joining datasets if you need a refresher on this command). And now we have one dataset that contains the county latitudes and longitudes and the data on income and poverty from the Census, so we're ready to plot!

```
## now plot!
ggplot(data=county_with_income) +
  geom_polygon(aes(x=long, y=lat, group=group, fill=pct_pvty)) +
  coord_quickmap()
```



So note how this syntax changes: I use the `fill` aesthetic to tell R to indicate the percent of people living in poverty in each county. Substantively, what did we learn? We see clearly the deep swath of poverty in part of the south, and in parts of the rural Midwest and west, and the much lower levels in parts of the Northeast and Pacific Coast. So with just a few lines of code, we can learn quite a lot about the geography of poverty!

By the way, if you were wondering, this type of map is a called a choropleth map, from the Greek for "area" and "multitude." It's a general name for a map that uses color to show how some variable (here, poverty) varies by geography. Use that fact at your next cocktail party!

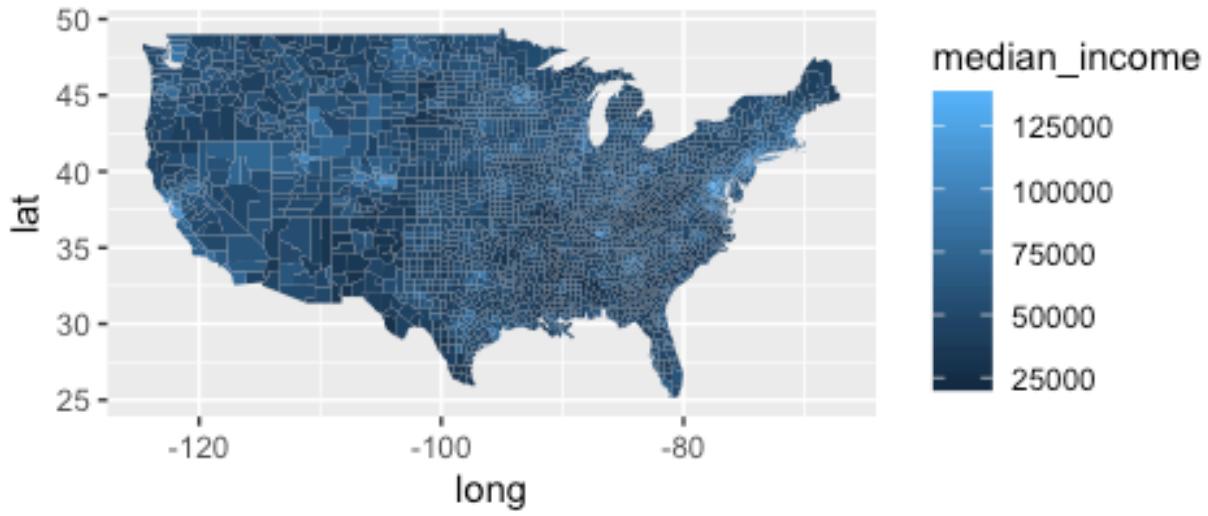
Also, note that we made a county-level map, but using the same tools, we could easily make a state-level map as well. The logic is identical, the syntax just changes very slightly.

Test Yourself: Median Income by County

In the dataset above, you'll find a variable `median_income` that gives the median income for each county in the U.S. Reproduce the map using this variable.

Answer:

```
ggplot(data=county_with_income) +
  geom_polygon(aes(x=long, y=lat, group=group, fill=median_income)) +
  coord_quickmap()
```

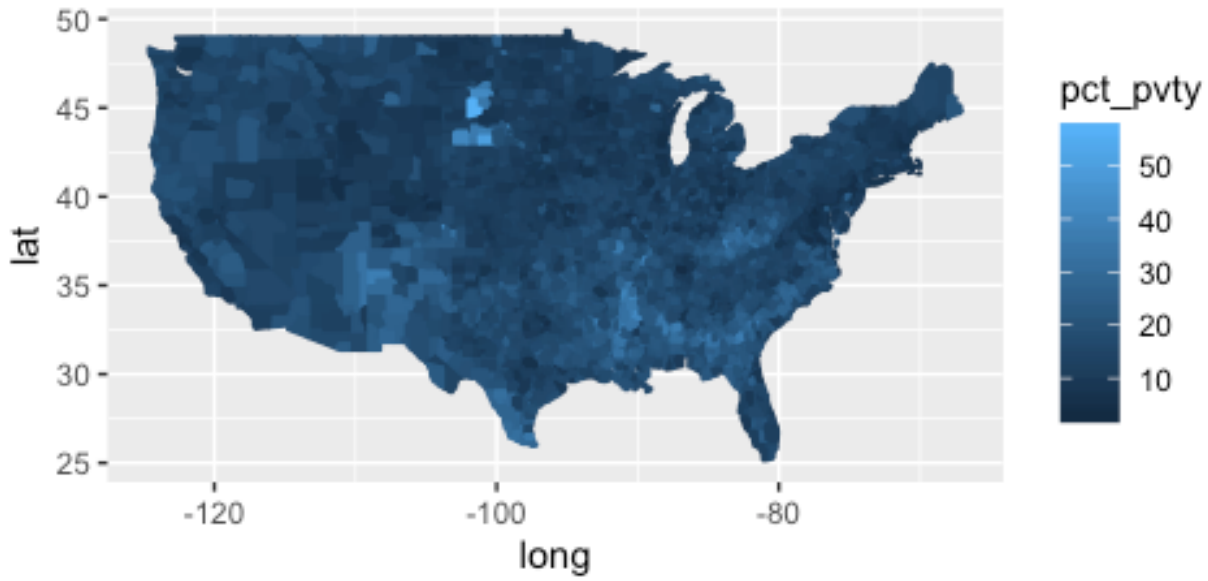


Going back to our original plot, there three aesthetic choices that you might not like in it:

1. Given the size of the counties, you might not like the county boundary lines.
2. `ggplot` defaults to a dark blue to light blue color scheme, and you might want another option.
3. The color scheme doesn't really highlight differences between high and low values very effectively.

To change #1, you simply change the color option in your call to ggplot:

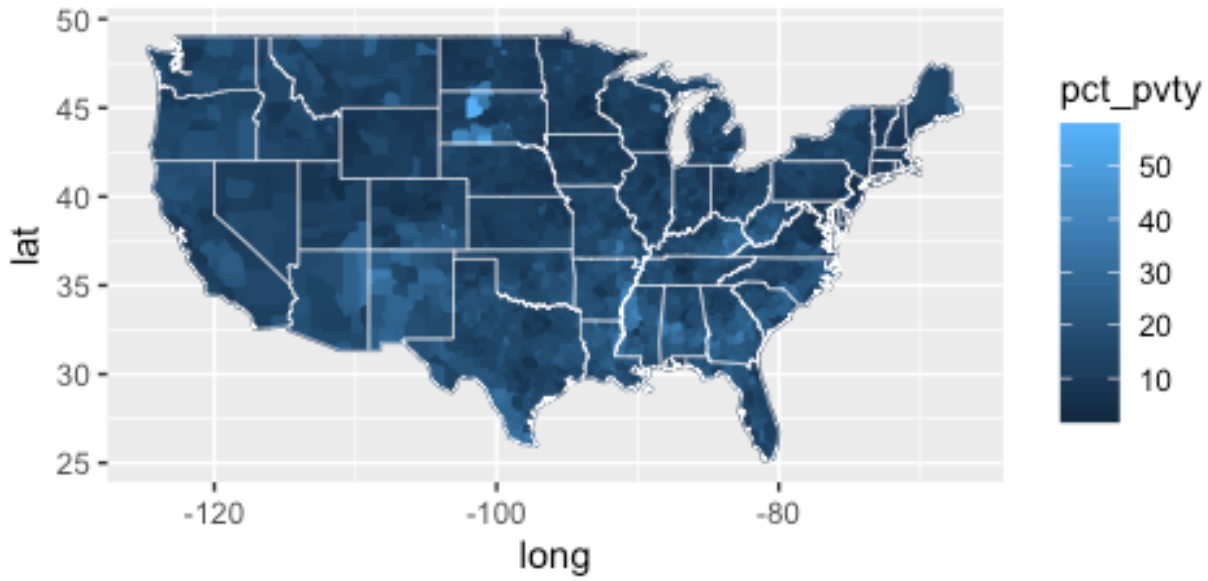
```
ggplot(data=county_with_income) +
  geom_polygon(aes(x=long, y=lat, group=group,
                   fill = pct_pvty, color=pct_pvty))  +
  coord_quickmap()
```



So note here that I map both `fill` and `color` to my variable of interest. That means that the interior of each county, and the border of each county, are plotted in the exact same color, thereby eliminating the gray boundaries in our default plot.

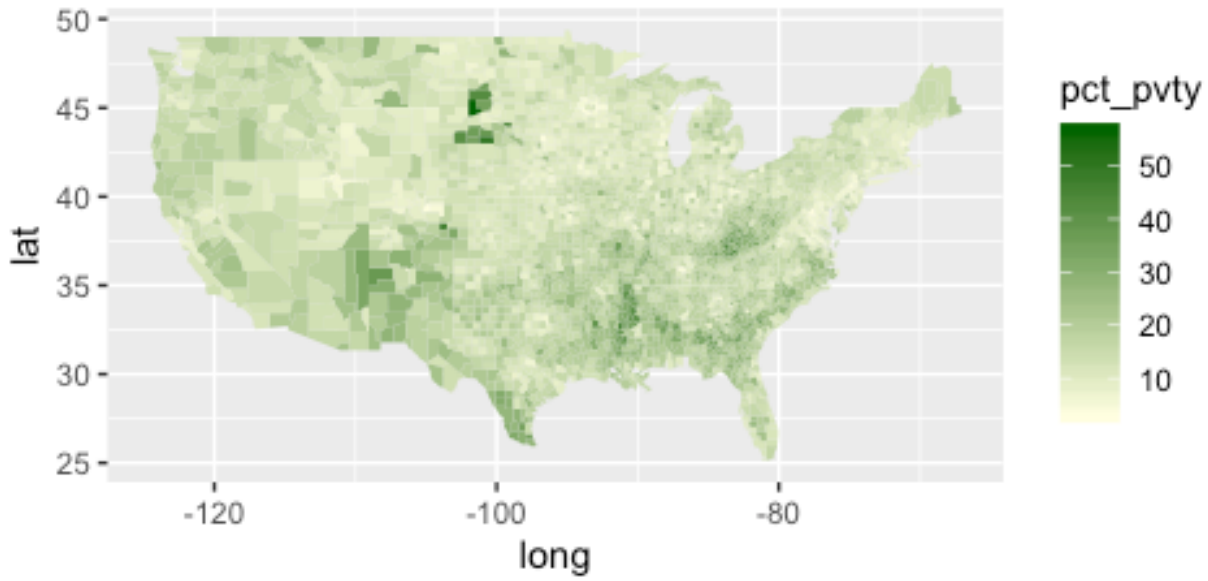
Of course, I could also add back in the state-boundaries as well, though it's not clear to me it's really helping much here:

```
ggplot(data=county_with_income) +
  geom_polygon(aes(x=long, y=lat, group=group,
                   fill = pct_pvty, color=pct_pvty))  +
  geom_polygon(data=states, aes(x=long, y=lat, group=group), fill=NA,
col="white", lwd=0.15) +
  coord_quickmap()
```



To adjust #2, you can use the `scale_fill_gradient` option in `ggplot`:

```
ggplot(data=county_with_income) +
  geom_polygon(aes(x=long, y=lat, group=group, fill=pct_pvty)) +
  scale_fill_gradient(low="lightyellow", high="darkgreen") +
  coord_quickmap()
```



If you want to see a list of colors available in R, just type `colours()` in the R console, and it will print out a list.

To adjust # 3, I sometimes find it helpful to treat my variable as discrete, rather than continuous. Why? Because if I choose a discrete color palette, I find I'm sometimes able to better see certain kinds of geographic trends.

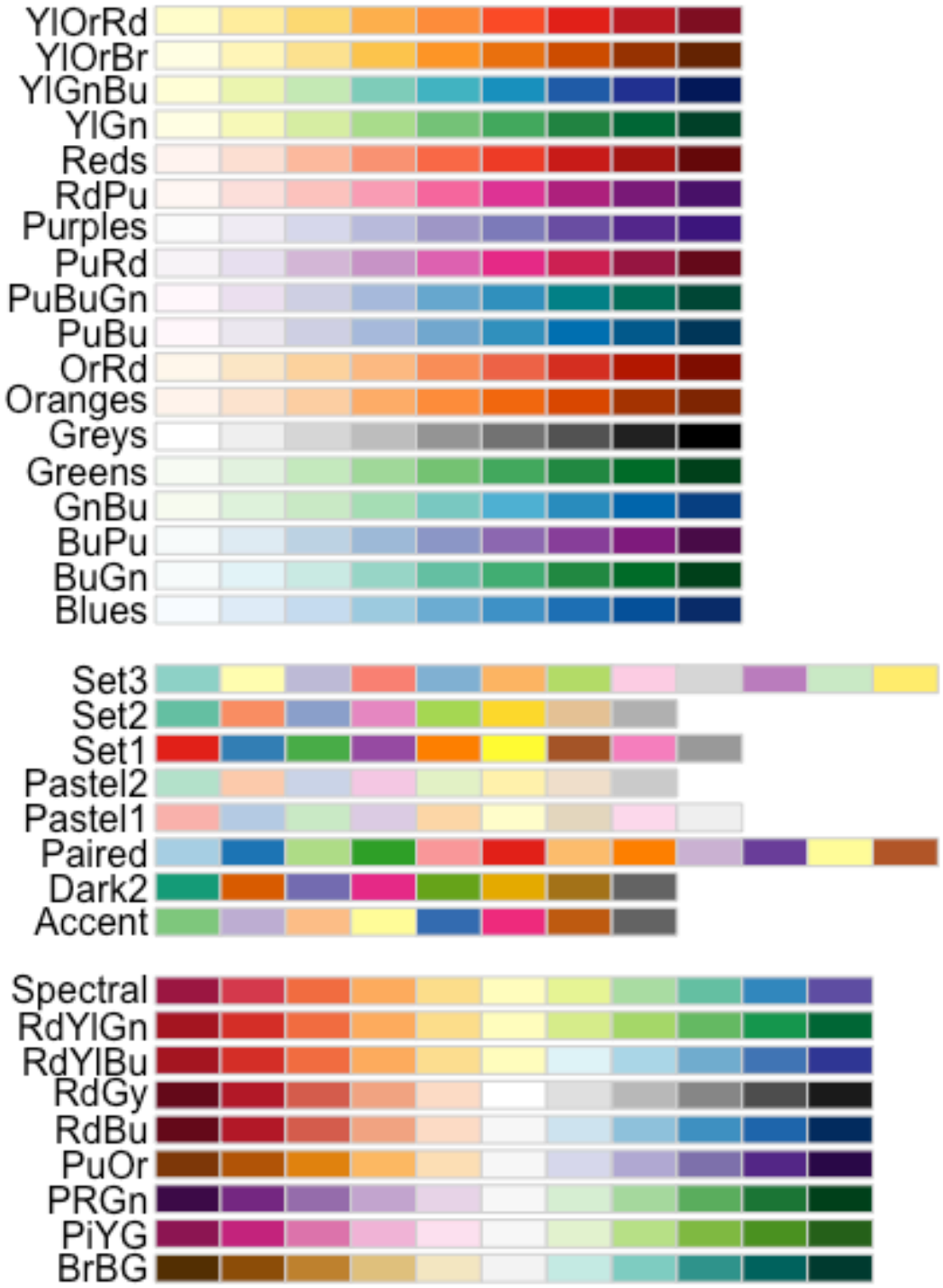
To do this, you'll need to make your continuous variable into a discrete one. Here, I've done this for you already for median income, and the variable is called `medinc_quart`. This variable takes median income and divides it into four equally sized groups. In our data, the four groups are:

1. 25% of counties have a median income of less than $42,275
2. 25% of counties have a median income between $42,275 and $48,885
3. 25% of counties have a median income between $48,885 and $56,696
4. 25% of counties have a median income above $56,696

We can use this new variable to plot a version of our data that will effectively highlight a few key geographic trends.

To do this, we need to choose a color palate that will highlight the differences in our data. Selecting a color scheme to do this can be a bit harder than you think! Luckily, Cynthia Brewer, a Professor of Geography at Penn State University, has studied this problem extensively, and has designed a set of color schemes for map-making. These can be found in the `RColorBrewer` package. To see the examples of color schemes in her package, you can use the command `display.brewer.all()`:

```
library(RColorBrewer)
display.brewer.all()
```

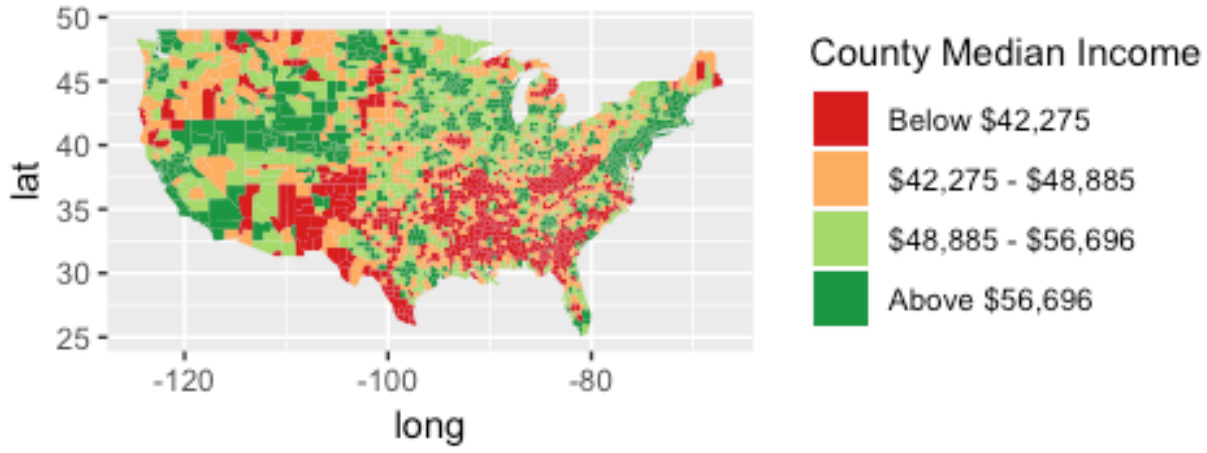

YlOrRd
YlOrBr
YlGnBu
YlGn
Reds
RdPu
Purples
PuRd
PuBuGn
PuBu
OrRd
Oranges
Greys
Greens
GnBu
BuPu
BuGn
Blues
Set3
Set2
Set1
Pastel2
Pastel1
Paired
Dark2
Accent
Spectral
RdYlGn
RdYlBu
RdGy
RdBu
PuOr
PRGn
PiYG
BrBG

There are 3 types of palettes : sequential, qualitative, and diverging, corresponding to the three groups in the graph:

1. The first chunk are sequential palettes, which are best for data that progresses from low to high along a gradient. For example, "YlOrRd" shows the data running from yellow, to orange, to red, making a visually appealing gradient.
2. The second chunk are qualitative palettes, which are best for nominal or categorical data, where the categories are distinct. The "Set" palettes are a good example of this.
3. The final chunk are the divergent values, which highlight extreme values are either end. "RdYlGn" is an example, which shows high (green) and low (red) values in offsetting colors, so you can see where the more extreme cases are.

All three types of scales have their uses, but I personally like the divergent palettes the best. Why? I find them helpful for highlight patterns of high and low cases. Let's see that in action for the median income map:

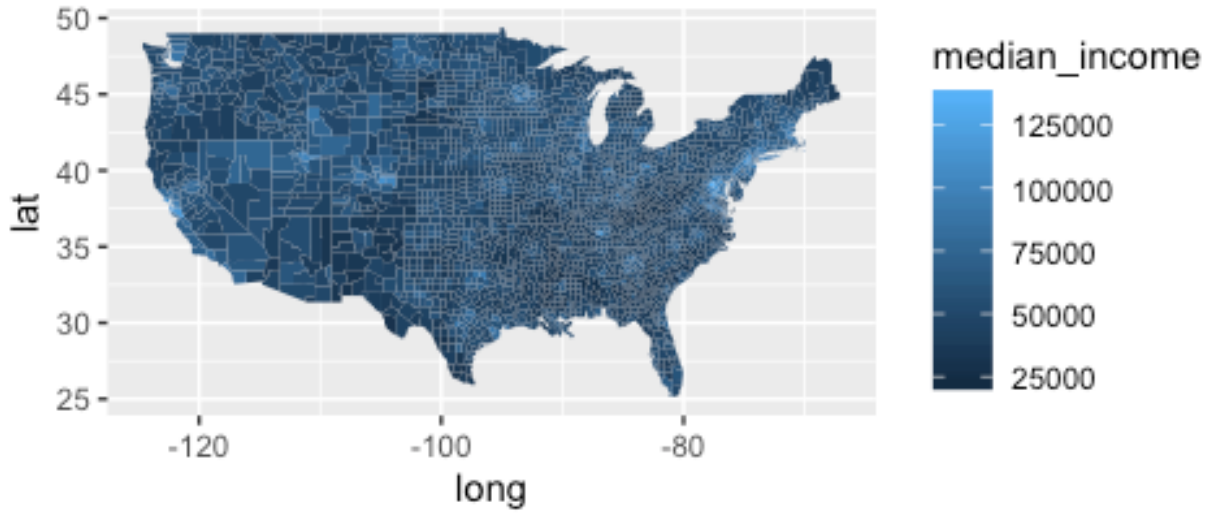
```
## Divergent color scheme
ggplot(data=county_with_income) +
  geom_polygon(aes(x=long, y=lat, group=group, fill=as.factor(medinc_quart)))
+
  coord_quickmap() +
  scale_fill_brewer(palette = "RdYlGn",
                    name="County Median Income",
                    labels=c("Below $42,275",
                             "$42,275 - $48,885",
                             "$48,885 - $56,696",
                             "Above $56,696"))
```



Note that here, I'm using the `scale_fill_brewer()` option because I'm using the color palette from the Brewer library. I've also added the legend labels as options just to make it clear what's happening substantively.

What I like about this graphic is that we can quickly see the geographic dispersion of median income. We see the richer areas along the coasts, and in part of the Mountain West, but then we see larger pockets of much lower incomes in the South and Appalachia.

We can contrast that with our original graph from above:



The same pattern is there (since it's the same underlying data), but we can see the geographic gradient of high/low incomes in the divergent graph much better (that is its aim after all). So while I wouldn't publish the divergent graph, I would make it for myself and use it to then decide how I might further analyze my data.

Of course, this is just scratching the surface of how to create maps and spatial visualizations. There are many more sophisticated packages in R, most notably `ggmap`, `sf`, and `tmap`. `ggmap` allows you to do "reverse geolocating" where you turn street addresses into latitudes and longitudes for plotting, and the other packages allow you to work with more complex mapping tools. But working with these packages requires some additional R knowledge, so we'll save them for a future course in this sequence. Happy map-making!

Appendix: All code used to make this document

```
knitr::opts_chunk$set(echo = TRUE)
knitr::opts_knit$set(root.dir = '~/Dropbox/DATA101')
library(tidyverse)
## Load in the libraries
library(maps)
library(mapdata)
```

```
## Get the data
usa <- map_data("usa")
head(usa)
## Actually draw the plot
ggplot() +
  geom_polygon(data = usa, aes(x=long, y = lat, group = group)) +
  coord_quickmap()
## Change the fill to blank
ggplot() +
  geom_polygon(data = usa, aes(x=long, y = lat, group = group), fill=NA,
color="black") +
  coord_quickmap()
ggplot() +
  geom_polygon(data = usa, aes(x=long, y = lat, group = group), fill=NA) +
  coord_quickmap()
ggplot() +
  geom_polygon(data = usa, aes(x=long, y = lat, group = group)) +
  coord_quickmap() +
  theme_void()
## Try a map of France
France <- map_data("france")
ggplot() +
  geom_polygon(data = France, aes(x=long, y = lat, group = group)) +
  coord_quickmap()
options(max.print=2000)
maps::map("world", namesonly=TRUE, plot=FALSE)
## Draw a map of the states
states <- map_data("state")
ggplot(data = states) +
  geom_polygon(aes(x = long, y = lat, group = group)) +
  coord_quickmap()
## better differentiate the states
ggplot(data = states) +
  geom_polygon(aes(x = long, y = lat, group = group), col="white", lwd=0.15)
+
  coord_quickmap()
## Draw just the pacific coast states
west_coast <- filter(states, region %in%
c("california","oregon","washington"))
ggplot(data = west_coast) +
  geom_polygon(aes(x = long, y = lat, group = group)) +
  coord_quickmap()
## Draw a map of counties
counties <- map_data("county")
ggplot(data = counties) +
  geom_polygon(aes(x = long, y = lat, group = group)) +
  coord_quickmap()
## show state boundaries to make it eaiser
ggplot() +
  geom_polygon(data=counties, aes(x=long, y=lat, group=group)) +
```

```
  geom_polygon(data=states, aes(x=long, y=lat, group=group), fill=NA,
col="white", lwd=0.15) +
  coord_quickmap()
ggplot(data=counties, aes(long,lat, group = group)) +
  geom_polygon(aes(fill = region)) +
  theme(legend.position="none") +
  coord_quickmap()
pa_counties <- filter(counties,region=="pennsylvania")
ggplot() +
  geom_polygon(data=pa_counties, aes(x=long, y=lat, group=group)) +
  coord_quickmap()
data(county.fips)
head(county.fips)
## does this match our data?
head(counties)
## do a mutate to match and join
county_with_fips <- counties %>%
  mutate(polyname = paste(region,subregion,sep=",")) %>%
  left_join(county.fips, by="polyname")
head(county_with_fips)
## Read in the new data
setwd("~/Dropbox/DATA101")
census_income <- read_csv(file="Data/Processed/Census_Poverty.csv")
## merge with our data
county_with_income <- inner_join(county_with_fips,census_income,
                                 by=c("fips"="fips_code"))
## now plot!
ggplot(data=county_with_income) +
  geom_polygon(aes(x=long, y=lat, group=group, fill=pct_pvty)) +
  coord_quickmap()
ggplot(data=county_with_income) +
  geom_polygon(aes(x=long, y=lat, group=group, fill=median_income)) +
  coord_quickmap()
ggplot(data=county_with_income) +
  geom_polygon(aes(x=long, y=lat, group=group,
                   fill = pct_pvty, color=pct_pvty))  +
  coord_quickmap()
ggplot(data=county_with_income) +
  geom_polygon(aes(x=long, y=lat, group=group,
                   fill = pct_pvty, color=pct_pvty))  +
  geom_polygon(data=states, aes(x=long, y=lat, group=group), fill=NA,
col="white", lwd=0.15) +
  coord_quickmap()
ggplot(data=county_with_income) +
  geom_polygon(aes(x=long, y=lat, group=group, fill=pct_pvty)) +
  scale_fill_gradient(low="lightyellow", high="darkgreen") +
  coord_quickmap()
library(RColorBrewer)
display.brewer.all()
## Divergent color scheme
```

```r
ggplot(data=county_with_income) +
  geom_polygon(aes(x=long, y=lat, group=group, fill=as.factor(medinc_quart)))
+
  coord_quickmap() +
  scale_fill_brewer(palette = "RdYlGn",
                    name="County Median Income",
                    labels=c("Below $42,275",
                             "$42,275 - $48,885",
                             "$48,885 - $56,696",
                             "Above $56,696"))
ggplot(data=county_with_income) +
  geom_polygon(aes(x=long, y=lat, group=group, fill=median_income)) +
  coord_quickmap()
```