# Geographic Data

R as a GIS

Image: Crater Lake, © the author

# Last time

- **Cartography** and thinking about map design
- **Elements of maps**: points, lines, polygons
- **Principles of color**
- **Drawing** basic political maps

# This time

- **Managing** geographic data
- Geographic data **types** and **sources**
- Different kinds of maps:
  - **Political**
  - **Road**
  - **Physical**

# Geography and R

- GIS is not the same thing as “ArcGIS” (or even QGIS)
    - People often use GIS to mean “draw me a map in Arc”
        - *There are tissue brands that are not Kleenex, plastic zip bags that are not Ziplocs, swabs that are not Qtips, you can search in other places than “Google”, etc*
    - And there are some good reasons for this.  Geographic data brings a host of additional complications to it, including:
        - Access, availability, detail and resolution, size, universality

# R is a GIS

- However, data is data
  - It exists in files
  - Those files are accessed on your drive or over the network
  - We are still doing the same things as ever before: manipulating cells of numbers that represent things in the world and representing it visually
- Geographic data is not “special” - it just has its own series of constraints, complications (and opportunities!)

# Types of geographic data

- **Vector** data
  - Data representing fundamental forms - points, lines, polygons
  - Typically used for *political, social and economic* maps
- **Raster** data
  - Data for representing spatial continuous phenomena, like elevation
  - Typically used for *physical* maps
- Can be used **together** if you have similar coordinate systems

# Vector data

- Vector data is data that describes a series of points, lines, and polygons
- Most commonly, you will see this as a **shape** file (although there are lots of other formats, too)
    - Think of a shape file as providing **shape** to the polygons we care about (cities, states, counties, etc)
    - And we then merge in **attributes** onto those shapes for visualization

# The Bosnian Civil War

- For instance:
  - “ged.csv” is a dataset on violent events in the Bosnian civil war
  - Includes information on start time, location (lat/long), and casualty numbers

```
> events <- read.csv("bosnia/ged.csv")
> summary(events)
       id          date_start           latitude       longitude          best        
 Min.   :199077   Length:1136        Min.   :42.71   Min.   :15.78   Min.   :   0.00  
 1st Qu.:199690   Class :character   1st Qu.:43.85   1st Qu.:18.10   1st Qu.:   1.00  
 Median :200136   Mode  :character   Median :43.85   Median :18.38   Median :   3.00  
 Mean   :200106                      Mean   :44.11   Mean   :18.16   Mean   :  17.29  
 3rd Qu.:200503                      3rd Qu.:44.54   3rd Qu.:18.38   3rd Qu.:   6.00  
 Max.   :200874                      Max.   :45.19   Max.   :19.54   Max.   :8106.00
```

# The Bosnian Civil War

- This is still just a regular data frame!  If we want R to do something different with it, we have to declare it appropriately. We declare both what the coordinates are and what the coordinate reference system is

```
events <- st_as_sf(events, coords = c("longitude", "latitude"), crs = 4326)
events
Simple feature collection with 1136 features and 3 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 15.78194 ymin: 42.71194 xmax: 19.53556 ymax: 45.18944
Geodetic CRS:  WGS 84
First 10 features:
        id date_start best                    geometry
1   199885 1993-02-01    6 POINT (18.80833 44.87278)
2   199767 1994-03-03    1 POINT (15.91861 44.84444)
3   200575 1995-03-12    1    POINT (18.38333 43.85)
```

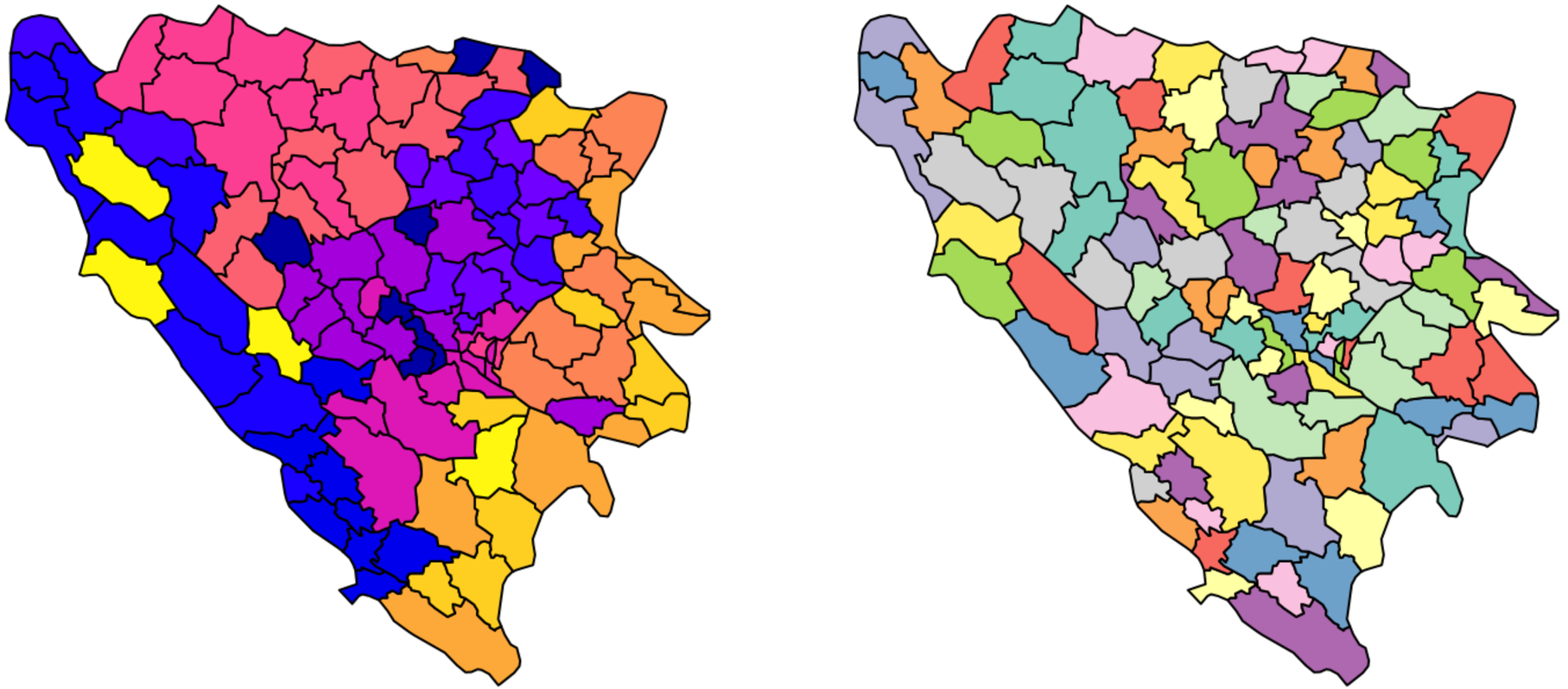
# The Bosnian Civil War

- We also need to read in the “shape” file, which gives us borders

```
> bosnia <- st_read("bosnia/bosnia.shp", crs = 4326)
> bosnia
Simple feature collection with 109 features and 2 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: 15.74059 ymin: 42.56583 xmax:
19.61979 ymax: 45.26595
Geodetic CRS:  WGS 84
First 10 features:
     id      name                                geometry
1   108 Trebinje POLYGON ((18.02122 42.93654...
2    17      Neum POLYGON ((17.83138 43.01822...
3   116  Ljubinje POLYGON ((18.15364 43.0523...
```

# Let’s plot!

```
plot(bosnia)
```



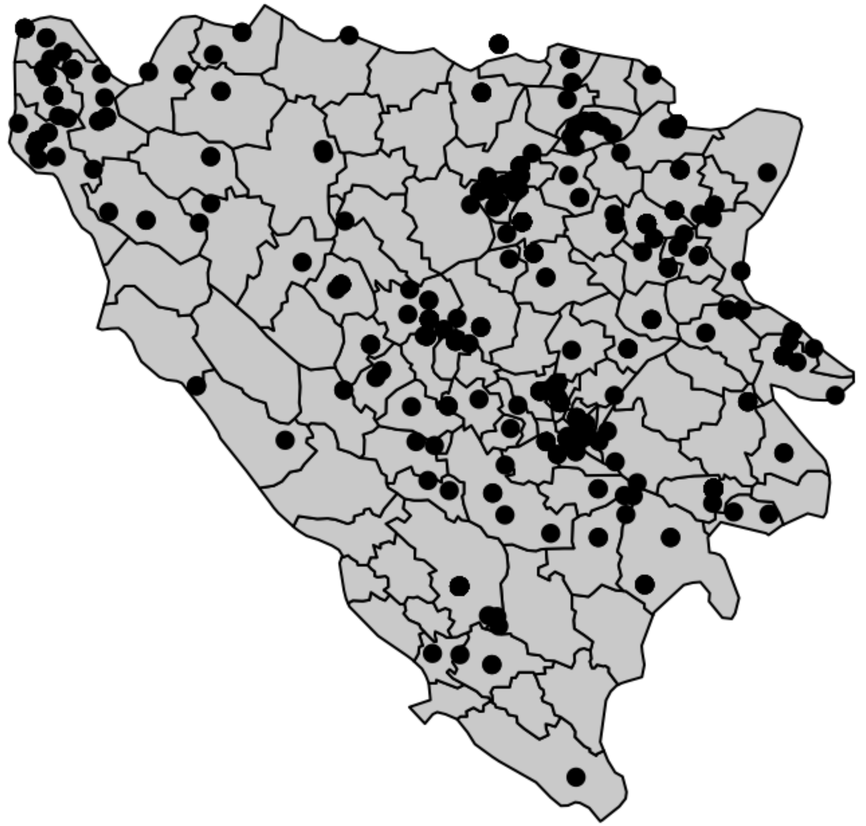

- Gibberish!

# Let’s be more specific

```
plot(st_geometry(bosnia), col = "lightgrey")
```

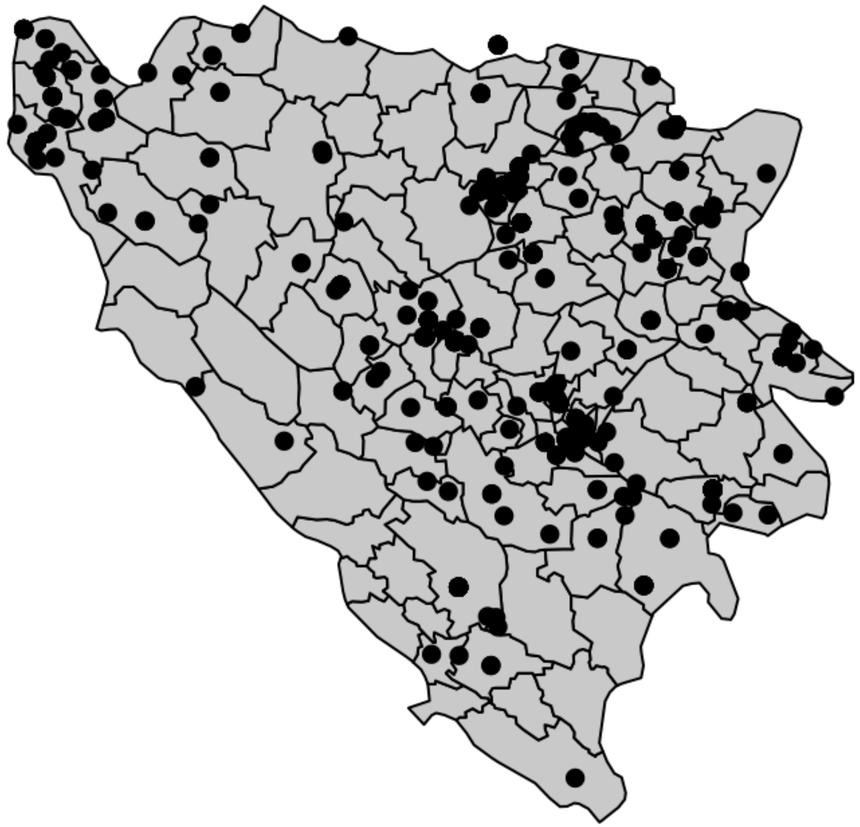
# Add the events themselves

```
plot(st_geometry(bosnia), col = "lightgrey")
plot(st_geometry(events), pch = 16, add = T)
```



# Add the events themselves

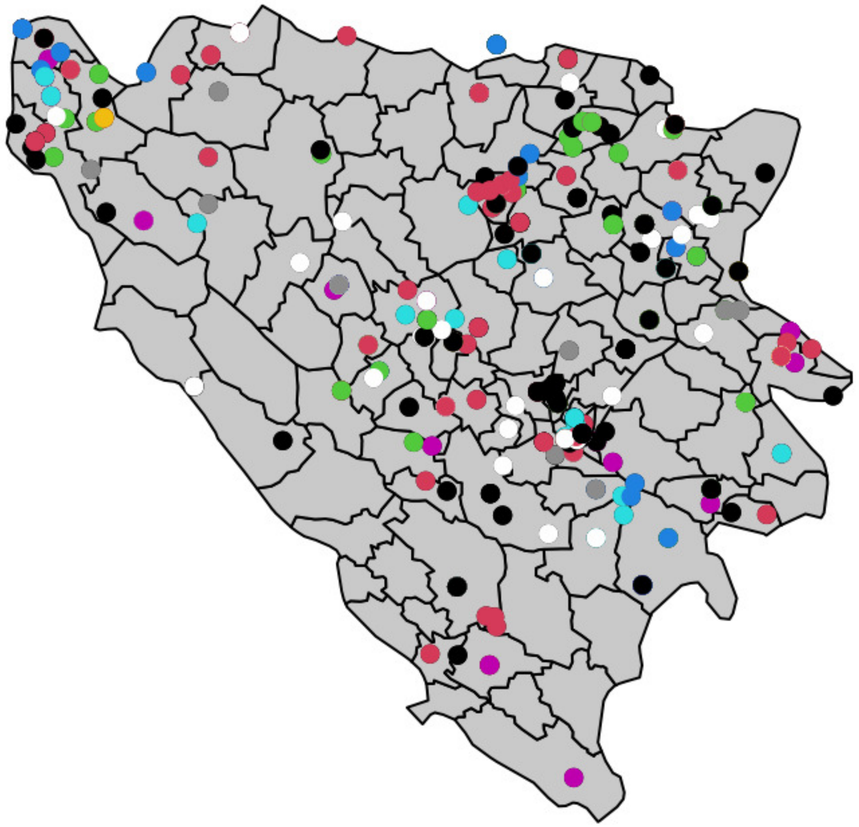
- Wow!  That was . . . easy?
  - Again . . . not so fast there, skippy.
  - The last 10% takes 90% of the effort.
- What do we want to do here?
  - Color events by severity
  - Title
  - Legend



# Add the events themselves

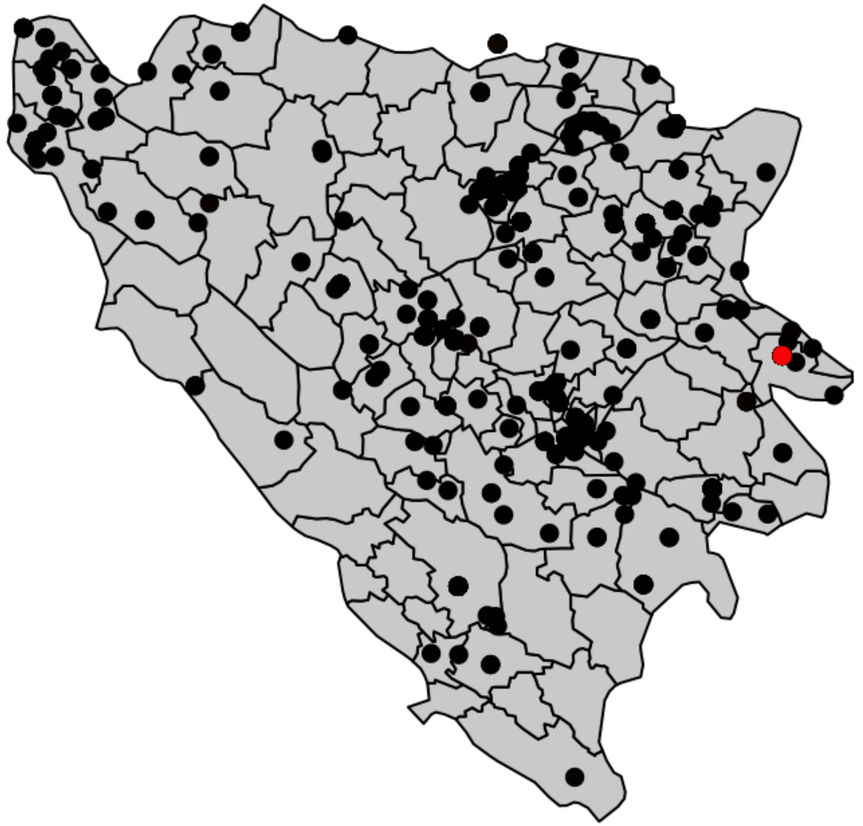
```
plot(st_geometry(bosnia), col = "lightgrey")
plot(st_geometry(events), col = events$best, pch = 16, add = TRUE)
```

- Drat.
  - Those colors are random, and not tied to severity.



# Let’s create a color scale!

```
library(scales)
black_red <- gradient_n_pal(c("black", "red"))

# numeric vector to color
vals <- events$best
rng  <- range(vals, na.rm = TRUE)    # data range
z    <- rescale(vals, to = c(0, 1), from = rng)
ptcol <- black_red(z)        # black to red colors for each point

# plot
plot(st_geometry(bosnia), col = "lightgrey")
plot(st_geometry(events), col = ptcol, pch = 16, add = TRUE)
```

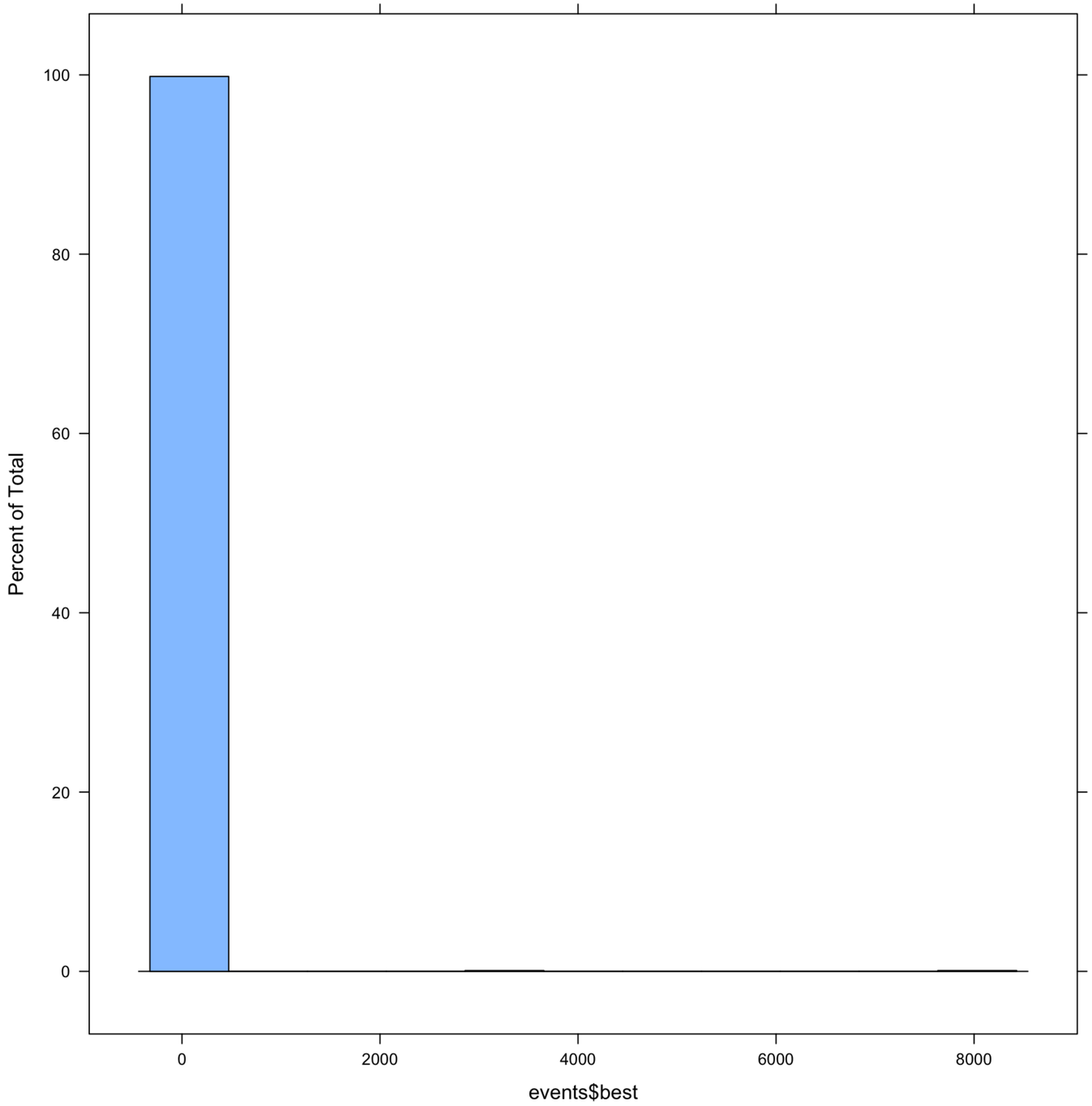


- But why no color?
  - We have at least one severe outlier.
  - Let’s investigate

# Let’s create a color scale!

```
library(lattice)
histogram(events$best)
```

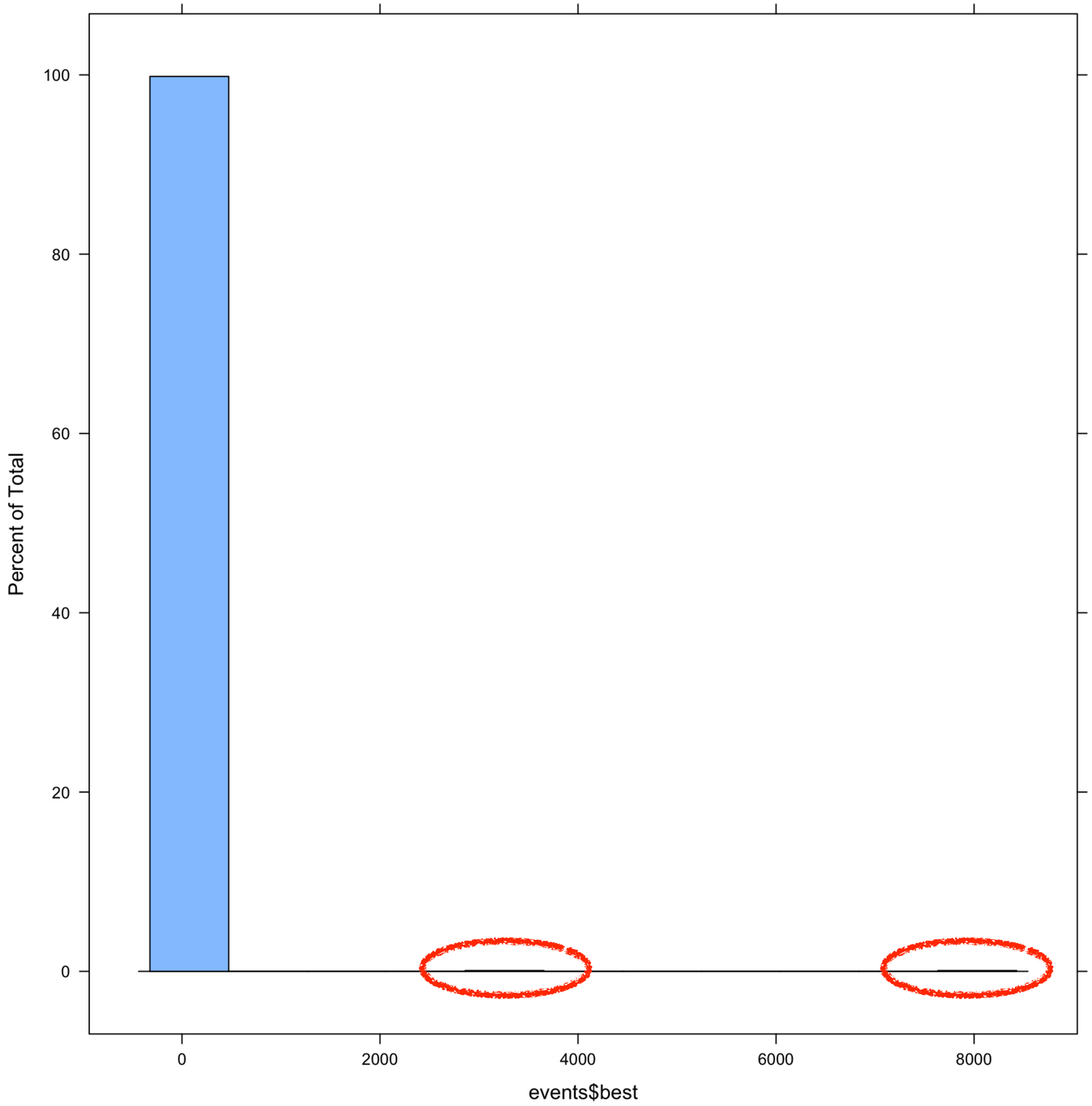
- This is what a histogram looks like what there are a lot of events at the bottom and some (in this case, very terrible) outliers



# Let’s create a color scale!

```
library(lattice)
histogram(events$best)
```

- This is what a histogram looks like what there are a lot of events at the bottom and some (in this case, very terrible) outliers



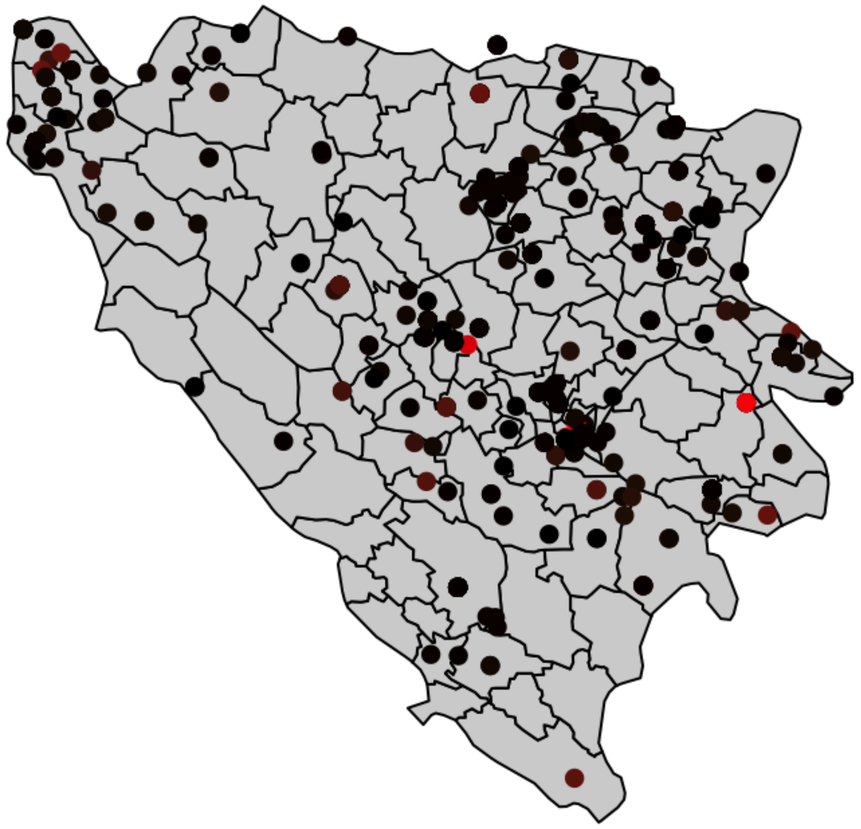
# Let’s create a color scale!

```
> events_sorted <- events[order(events$best), ]
> tail(events_sorted$best, 10)     # 10 lowest
 [1]  100  101  101  144  150  197  270  300 3000 8106
```

- We have a 8106, a 3000, a few between 100 and 300, and then everything else
- New strategy:
  - Create different channels (normal, large, outliers) and represent with different shapes

# Let’s create a color scale!

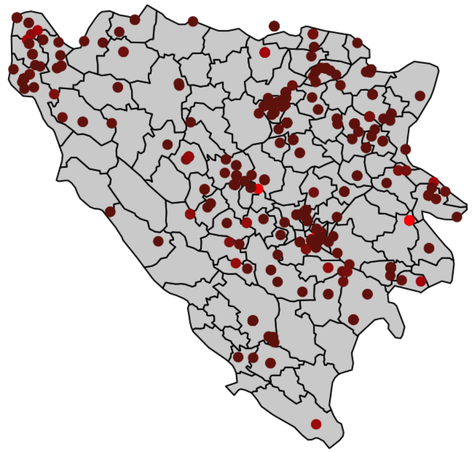
```
large_vals    <- c(144, 150, 197, 270, 300)
outlier_vals  <- c(3000, 8106)

ev_out   <- events[!is.na(events$best) & events$best %in% outlier_vals, ]
ev_large <- events[!is.na(events$best) & events$best %in% large_vals, ]
ev_reg   <- events[!is.na(events$best) &
                   !(events$best %in% c(large_vals, outlier_vals)) &
                   events$best >= 0 & events$best <= 101, ]

vals <- ev_reg$best
rng  <- range(vals, na.rm = TRUE)           # data domain
z    <- rescale(vals, to = c(0, 1), from = rng)
ptcol <- black_red(z)

plot(st_geometry(bosnia), col = "lightgrey")
plot(st_geometry(ev_reg), col = ptcol, pch = 16, add = TRUE)
```

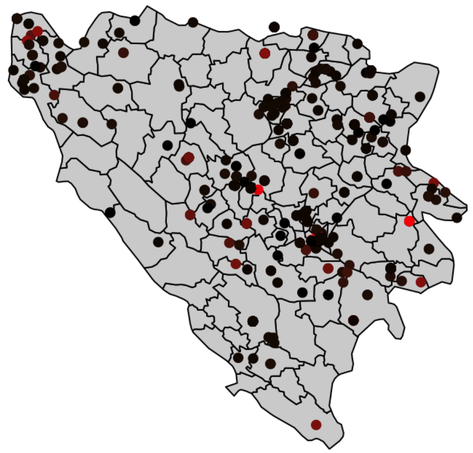


- Better?

# Let’s create a color scale!

```
#Still dark.  what if we made the mean of the color scale
the mean of the variable?
mean(ev_reg$best)
median(ev_reg$best)

z <- rescale_mid(vals, to = c(0, 1), from = rng, mid = 7)
ptcol <- black_red(z)

plot(st_geometry(bosnia), col = "lightgrey")
plot(st_geometry(ev_reg), col = ptcol, pch = 16, add = TRUE)

z <- rescale(vals, to = c(0, 1), from = rng)
z <- z ^ 0.7  # lower exponent -> shifts color toward red
earlier
ptcol <- black_red(z)

plot(st_geometry(bosnia), col = "lightgrey")
plot(st_geometry(ev_reg), col = ptcol, pch = 16, add = TRUE)
```

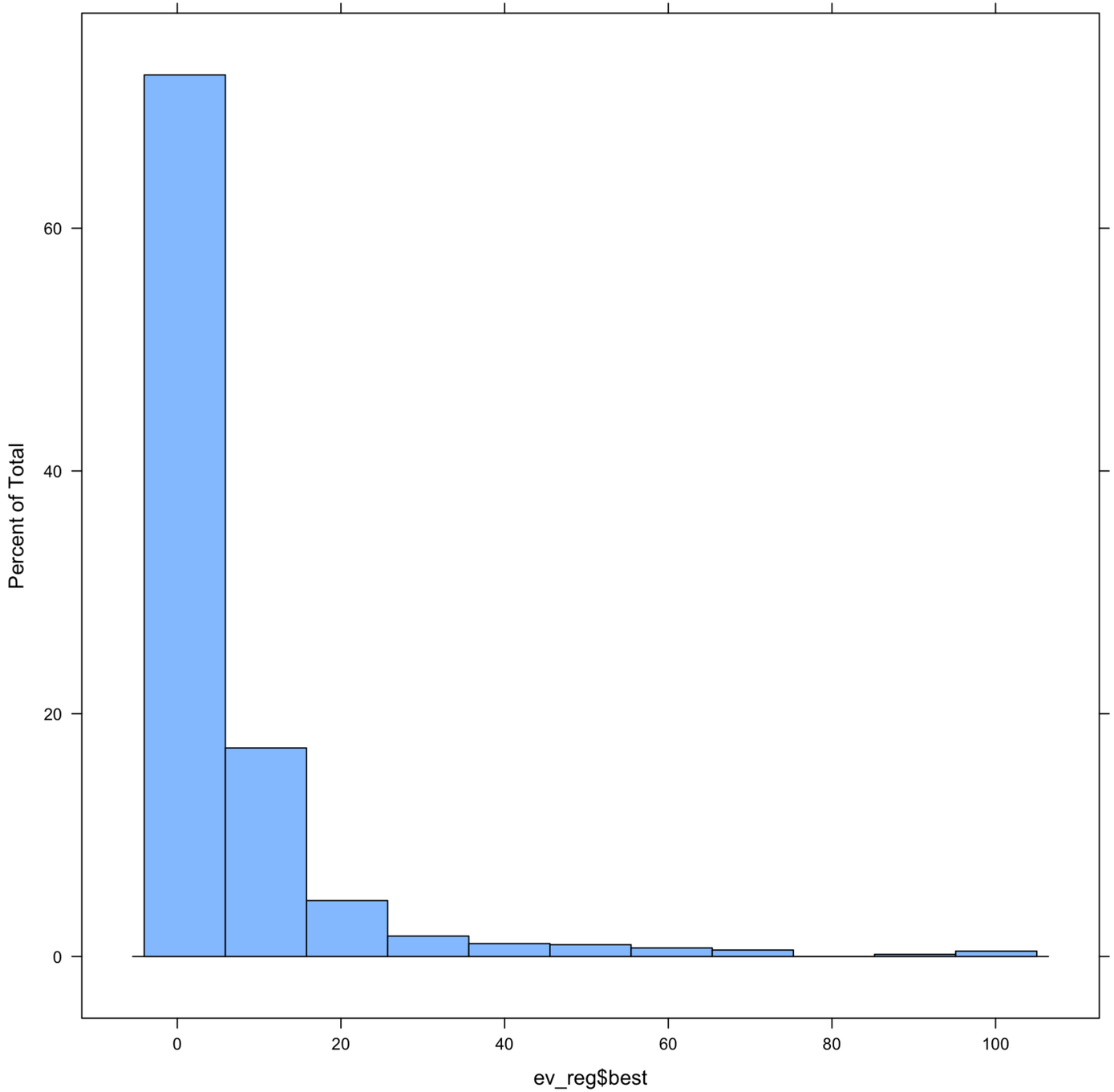




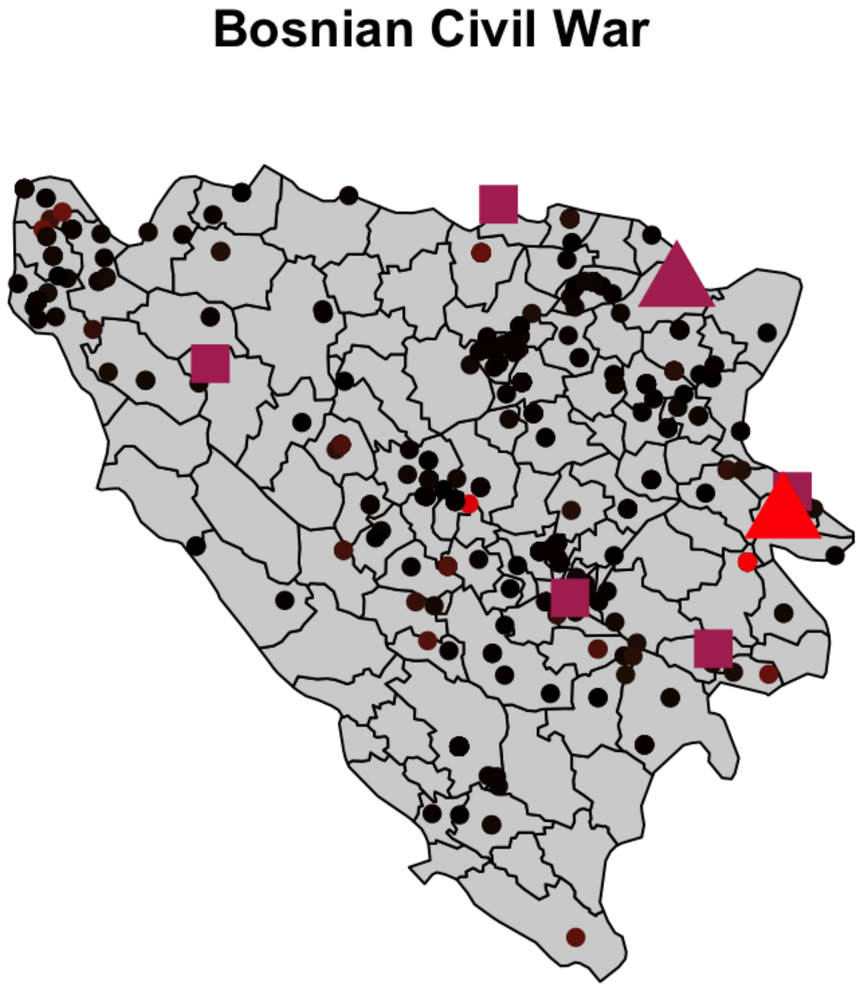
# Let’s create a color scale!

```
library(lattice)
histogram(ev_reg$best)
```

- It’s still just a very skewed dataset.  Could keep playing with the color through transformation and midpoints, but just a tough challenge
- Let’s just add our second and third data layers

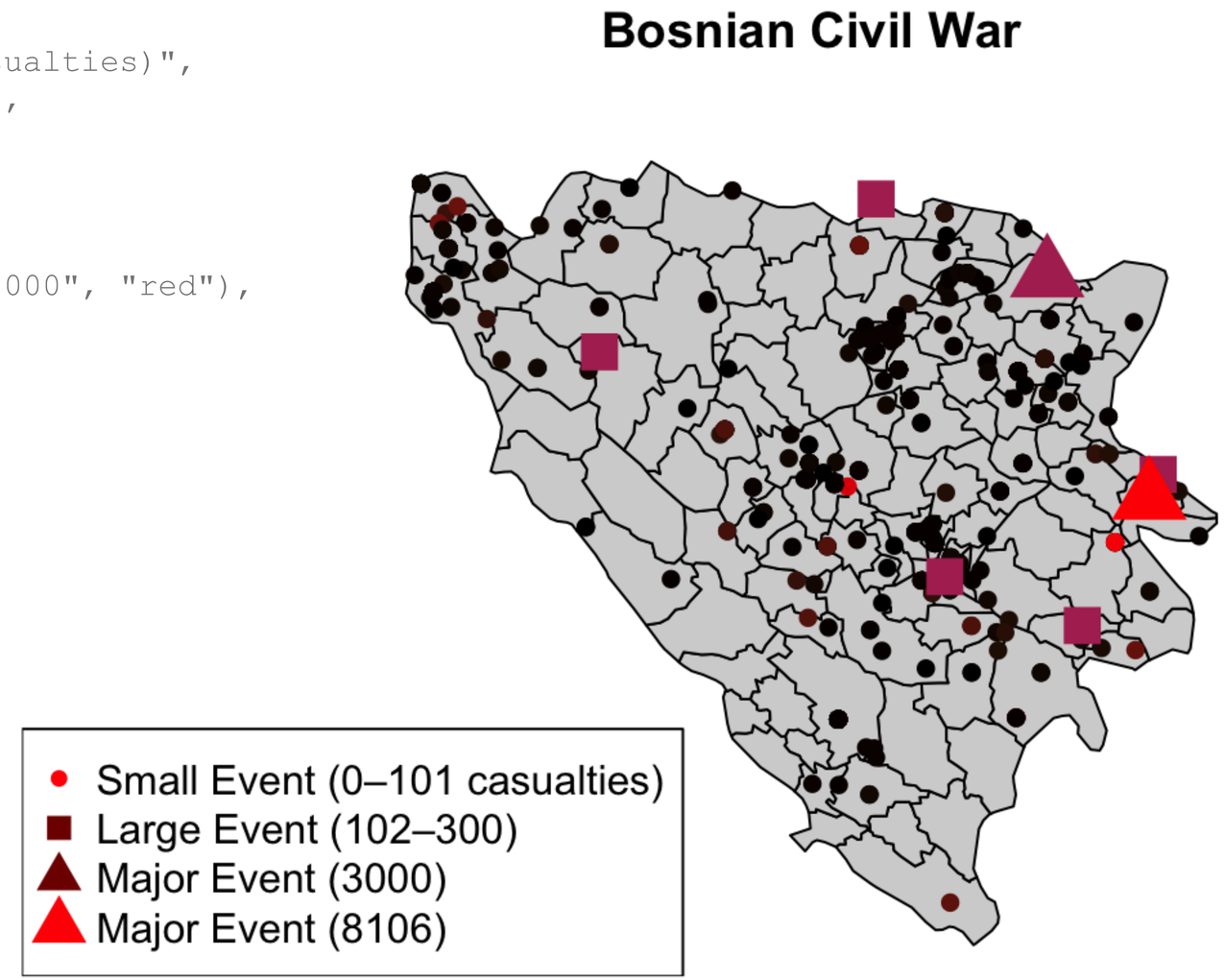


```
plot(st_geometry(bosnia), col = "lightgrey", main="Bosnian Civil War")
plot(st_geometry(ev_reg), col = ptcol, pch = 16, add = TRUE)
plot(st_geometry(ev_large), col = "maroon", pch = 15, cex=2, add = TRUE)
plot(st_geometry(ev_out[1,]), col = "maroon", pch = 17, cex=3, add = TRUE)
plot(st_geometry(ev_out[2,]), col = "red", pch = 17, cex=3, add = TRUE)
```



```
plot(st_geometry(bosnia), col = "lightgrey", main="Bosnian Civil War")
plot(st_geometry(ev_reg), col = ptcol, pch = 16, add = TRUE)
plot(st_geometry(ev_large), col = "maroon", pch = 15, cex=2, add = TRUE)
plot(st_geometry(ev_out[1,]), col = "maroon", pch = 17, cex=3, add = TRUE)
plot(st_geometry(ev_out[2,]), col = "red", pch = 17, cex=3, add = TRUE)
```

```
legend("bottomleft",
       legend = c("Small Event (0-101 casualties)",
                  "Large Event (102-300)",
                  "Major Event (3000)",
                  "Major Event (8106)"),
       pch    = c(16, 15, 17, 17),
       pt.cex = c(0.9, 1.3, 1.8, 2.2),
       col    = c("red", "#800000", "#800000", "red"),
       bty    = "o")  # <-- draw box
```



# So what's familiar to us here?

- Most things!
  - We are layering
  - We are playing with colors (exhaustively)
  - We are making choices about data representation
- The only major difference is tweaking how we manage the data on the front end, and working with a geographic shape file as a data source

# We can do a few other things to enable better data management

- We can merge the two datasets together to form a single object:

```
joined <- st_join(events, bosnia)
```

- This allows us to calculate statistics more effectively in the tidyverse:

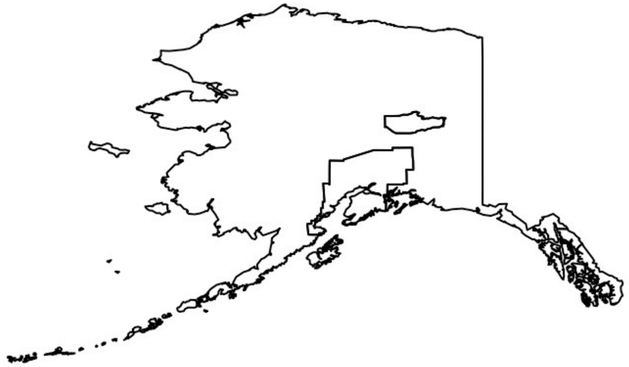
```
joined <- st_join(events, bosnia)
eventcounts <- joined |>
  as.data.frame() |>
  group_by(id.y) |>
  count(name = "num_events")
eventcounts
```

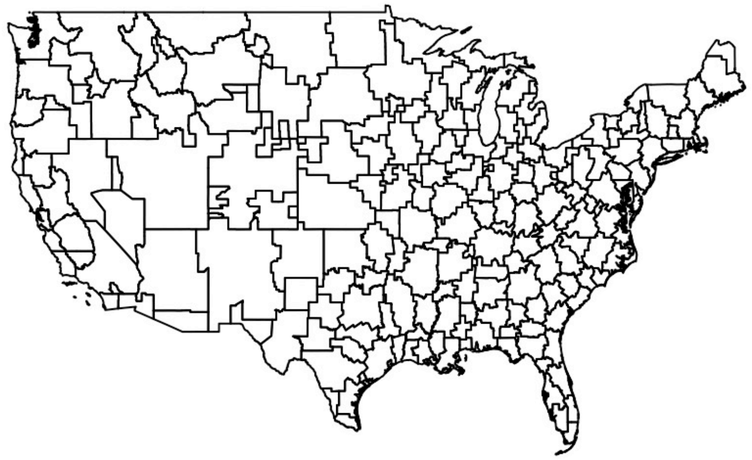
```
> eventcounts
# A tibble: 79 × 2
# Groups:   id.y [79]
    id.y num_events
   <int>      <int>
 1     1          1
 2     2          7
 3     4          1
 4     6          1
```

# Geographic data sources

- This is where you spend a lot of your time. Some valuable ones:
  - The US Census Bureau (data and shape files)
  - Open Street Routing (roads, some geographic features)
  - Shape files you find floating around

# Here, for instance, is a shape file of media markets I just found and plotted with `st_read()` and `plot()`





# Layering map visualizations

1. Identify a source for your shape data
2. Identify a merge key to bring things in/along
3. Identify data for what you want to color in the shapes with
4. Identify what/if you want to layer on top for points, lines, etc
5. Manage, merge, and plot
6. Edit, edit, edit

# Example: let’s plot something from the census by county

1. Identify a source for your shape data
2. Identify a merge key to bring things along
3. Identify data for what you want to color in the shapes with
4. Identify what/if you want to layer on top for points, lines, etc
5. Manage, merge, and plot
6. Edit, edit, edit

# Example: let's plot something from the census by county

1. Identify a source for your shape data Census
2. Identify a merge key to bring things along FIPS code, if needed
3. Identify data for what you want to color in the shapes with Census
4. Identify what/if you want to layer on top for points, lines, etc
5. Manage, merge, and plot
6. Edit, edit, edit

# Some simple census API code with tidy census

```
library(tidycensus)
library(ggplot2)
library(dplyr)
library(sf)

# (Run once per computer; then restart R)
#census_api_key("YOUR KEY", install = TRUE)

# Load 2020 population by county, with geometry
counties <- get_decennial(
  geography = "county",
  variables = "P1_001N",   # total population
  year = 2020,
  geometry = TRUE
)
```

- *Loads population and geometry! Pre-merged!*

# Wait, is **`tidycensus`** really this awesome?

```
> counties
Simple feature collection with 3221 features and 6 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -179.1467 ymin: 17.88328 xmax: 179.7785 ymax: 71.38781
Geodetic CRS:  WGS 84
# A tibble: 3,221 × 7
   GEOID NAME                        variable   value                                                geometry area_km2
density
 * <chr> <chr>                       <chr>      <dbl>                                  <MULTIPOLYGON [°]>    <dbl>
<dbl>
 1 13031 Bulloch County, Georgia     P1_001N    81099 (((-82.02684 32.55516, -82.02527 32.55836, -82.0158…    1783.   45.5
 2 13121 Fulton County, Georgia      P1_001N  1066710 (((-84.84931 33.51318, -84.84429 33.51469, -84.8410…    1384.  771.
 3 13179 Liberty County, Georgia     P1_001N    65256 (((-81.8244 32.01488, -81.81338 32.01623, -81.80993…    1401.   46.6
 4 13189 McDuffie County, Georgia    P1_001N    21632 (((-82.64852 33.60838, -82.64409 33.60685, -82.6428…     690.   31.4
 5 13213 Murray County, Georgia      P1_001N    39973 (((-84.94433 34.68004, -84.9431 34.68043, -84.94017…     898.   44.5
 6 13209 Montgomery County, Georgia P1_001N     8610 (((-82.65581 32.3015, -82.65291 32.30334, -82.65243…     634.   13.6
 7 13139 Hall County, Georgia        P1_001N   203136 (((-84.06225 34.16845, -84.05949 34.17551, -84.0587…    1112.  183.
 8 13245 Richmond County, Georgia    P1_001N   206607 (((-82.3503 33.3148, -82.33028 33.32901, -82.29418 …     851.  243.
 9 13049 Charlton County, Georgia    P1_001N    12518 (((-82.42027 30.80053, -82.42018 30.80344, -82.4151…    2029.
6.17
10 13141 Hancock County, Georgia     P1_001N     8735 (((-83.27522 33.19132, -83.27413 33.1968, -83.27401…    1238.
7.06
# ℹ 3,211 more rows
# ℹ Use `print(n = ...)` to see more rows
```

- ***Yes, in fact.  Yes, is is.***

# LET'S GRAPH!! IT'S SO EASY!!!

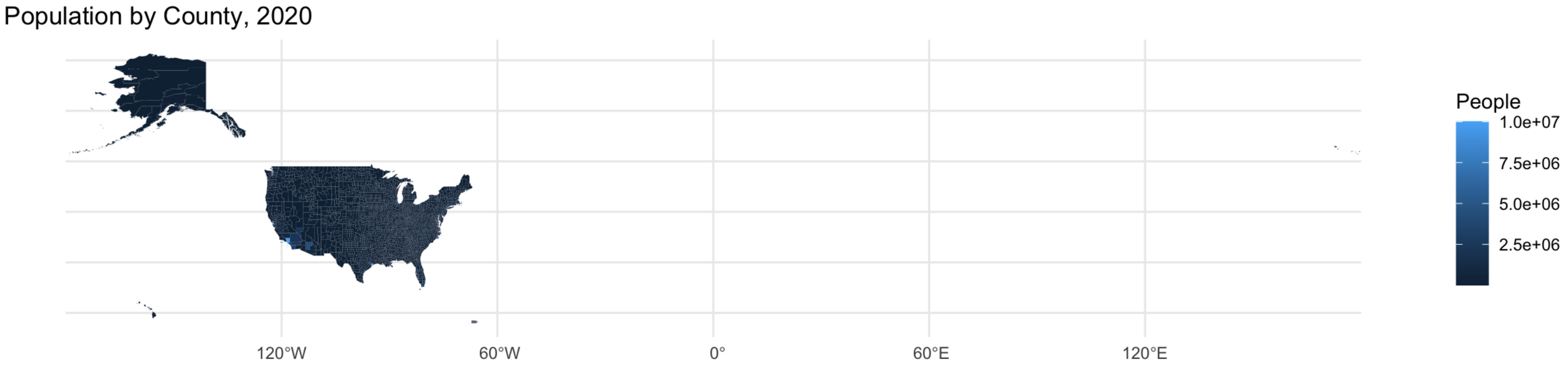
- Using what we learned in lab last week:

```
ggplot(counties) +
  geom_sf(aes(fill = value), color = NA) +
  scale_fill_viridis_c() +
  labs(title = "Population by County, 2020",
       fill = "People") +
  theme_minimal()
```

- **Can it really be this easy??**

# Well, yes . . .

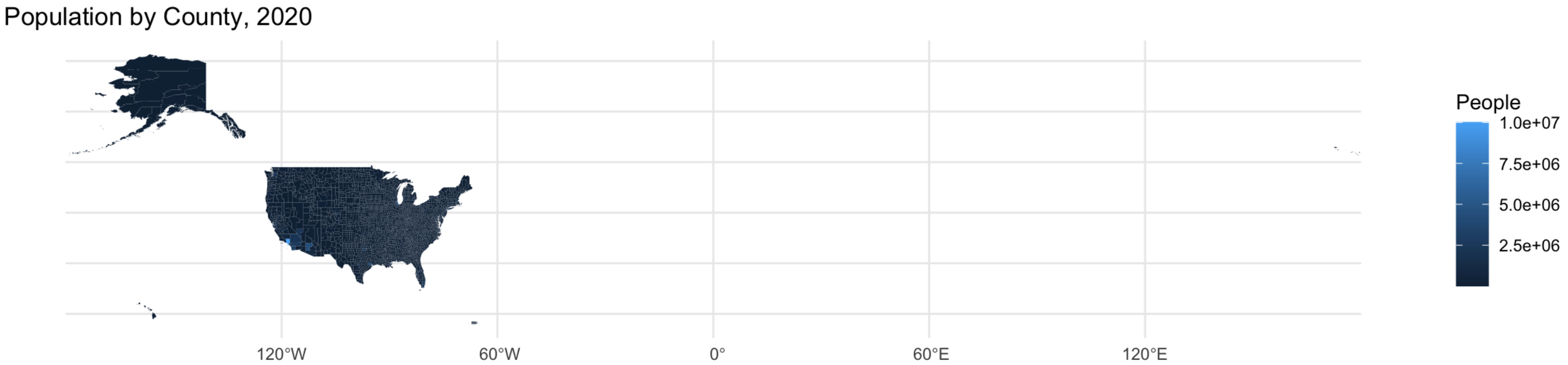
- and also no




- What are our problems?

# Problems

- At minimum:
  - Color scale
  - Plot area. *I'm looking at you, trailing Hawaiian islands*

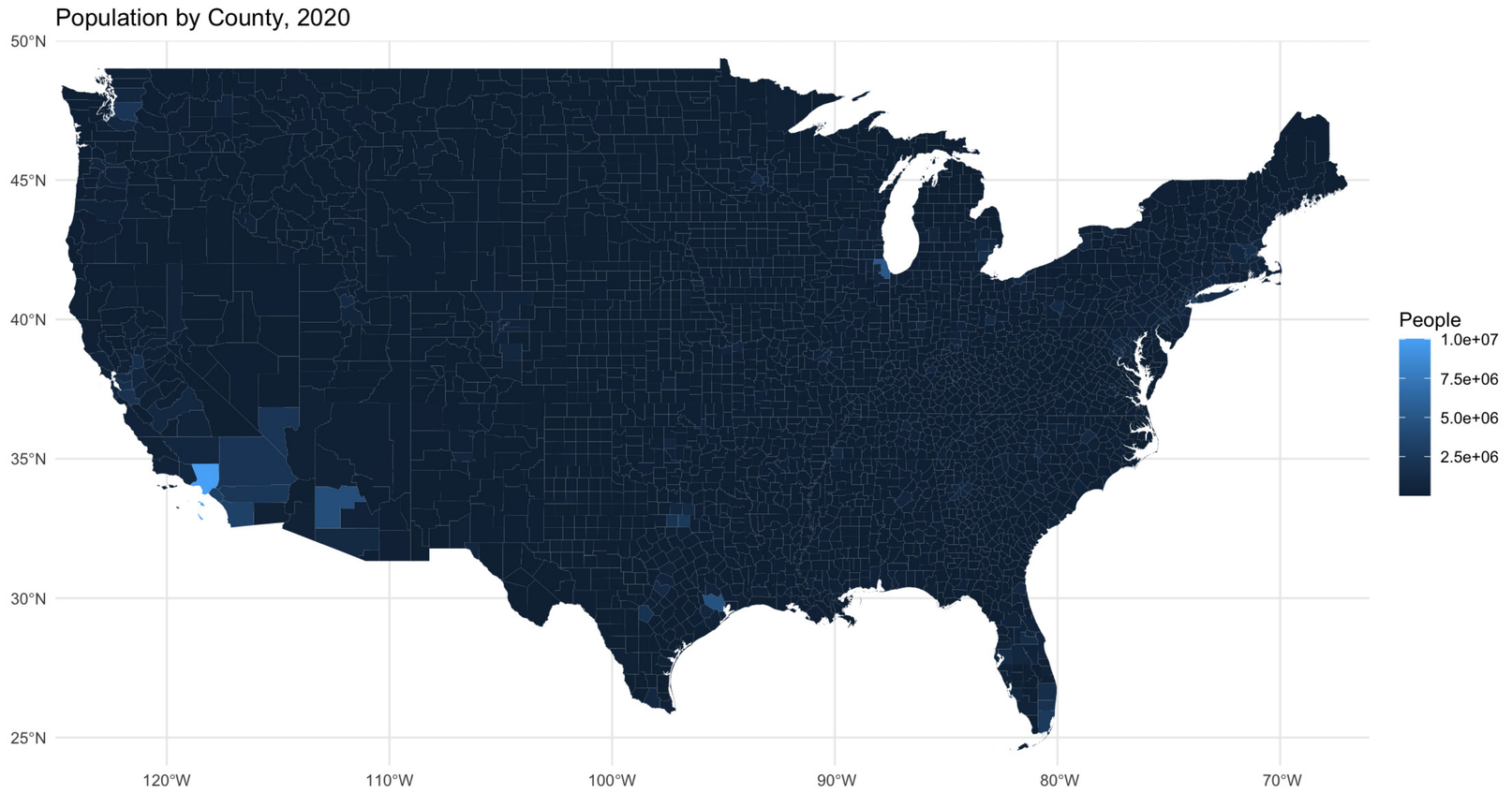


# Two ways to think of re-displaying here

- 1. Limit the *view window*
- 2. Constrain the *data itself*

# View windows

```
ggplot(counties) +
  geom_sf(aes(fill = value), color = NA) +
  coord_sf(
    xlim = c(-125, -66),    # longitude range
    ylim = c(24, 50),       # latitude range
    expand = FALSE
  ) +
  labs(title = "Population by County, 2020",
       fill = "People") +
  theme_minimal()
```



# Constraining data itself

- We have GEOID, which is a FIPS code for county.  We're going to take out the first two integers there to get the State FIPS, not the county FIPs

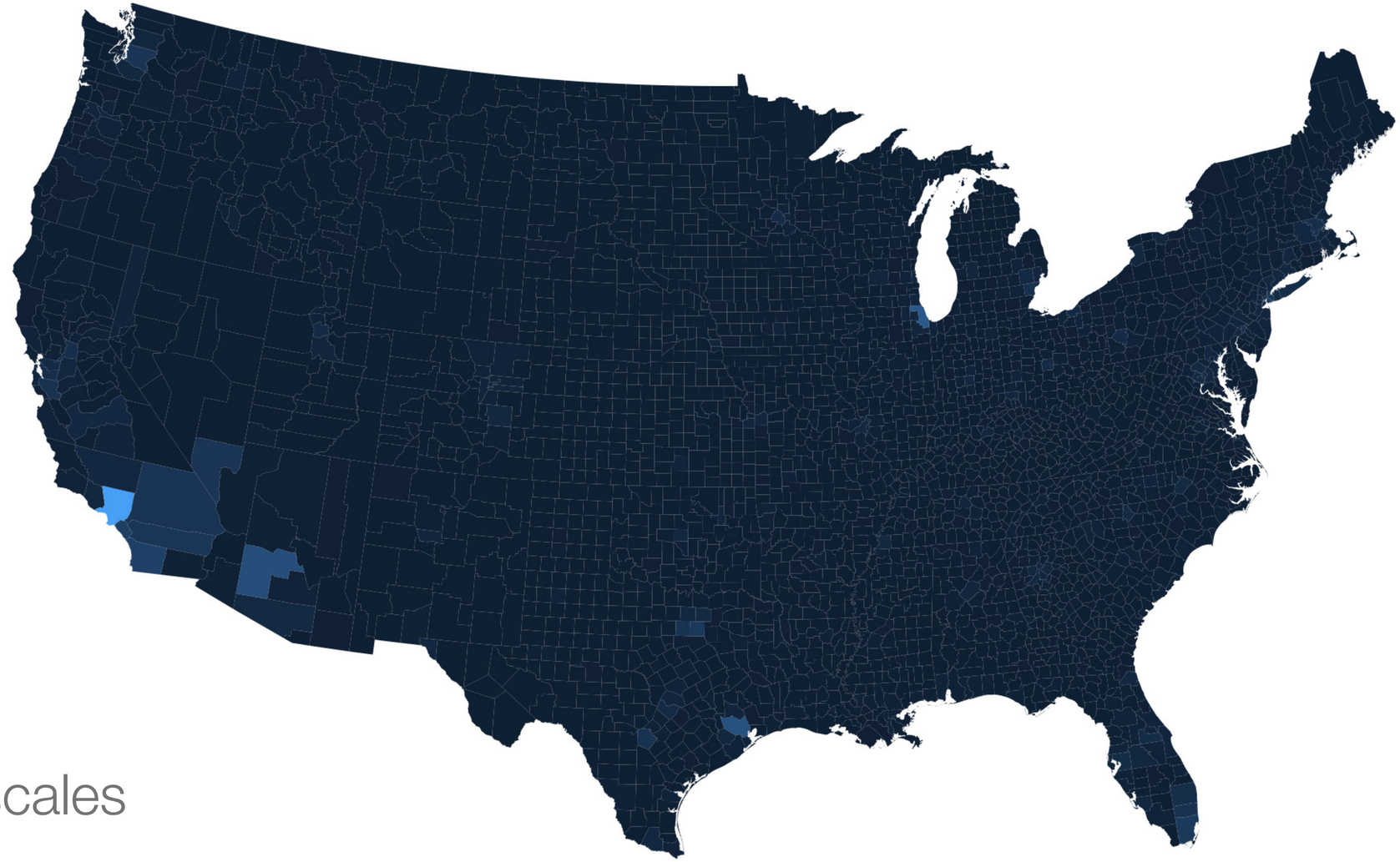
```
counties <- counties |>
  mutate(STATEFP = substr(GEOID, 1, 2)) |>
  filter(!STATEFP %in% c("02", "15", "72"))  # AK, HI, PR

lower48 <- counties |>
  filter(!STATEFP %in% c("02", "15", "72"))  # 02 = AK, 15 = HI, 72 = PR
```

- Now, we'll also start the "make nice" process with a projection

```
# Add a nice projection for U.S. maps (Albers equal area = 5070)
lower48 <- st_transform(lower48, 5070)
```

# Well, this is getting better

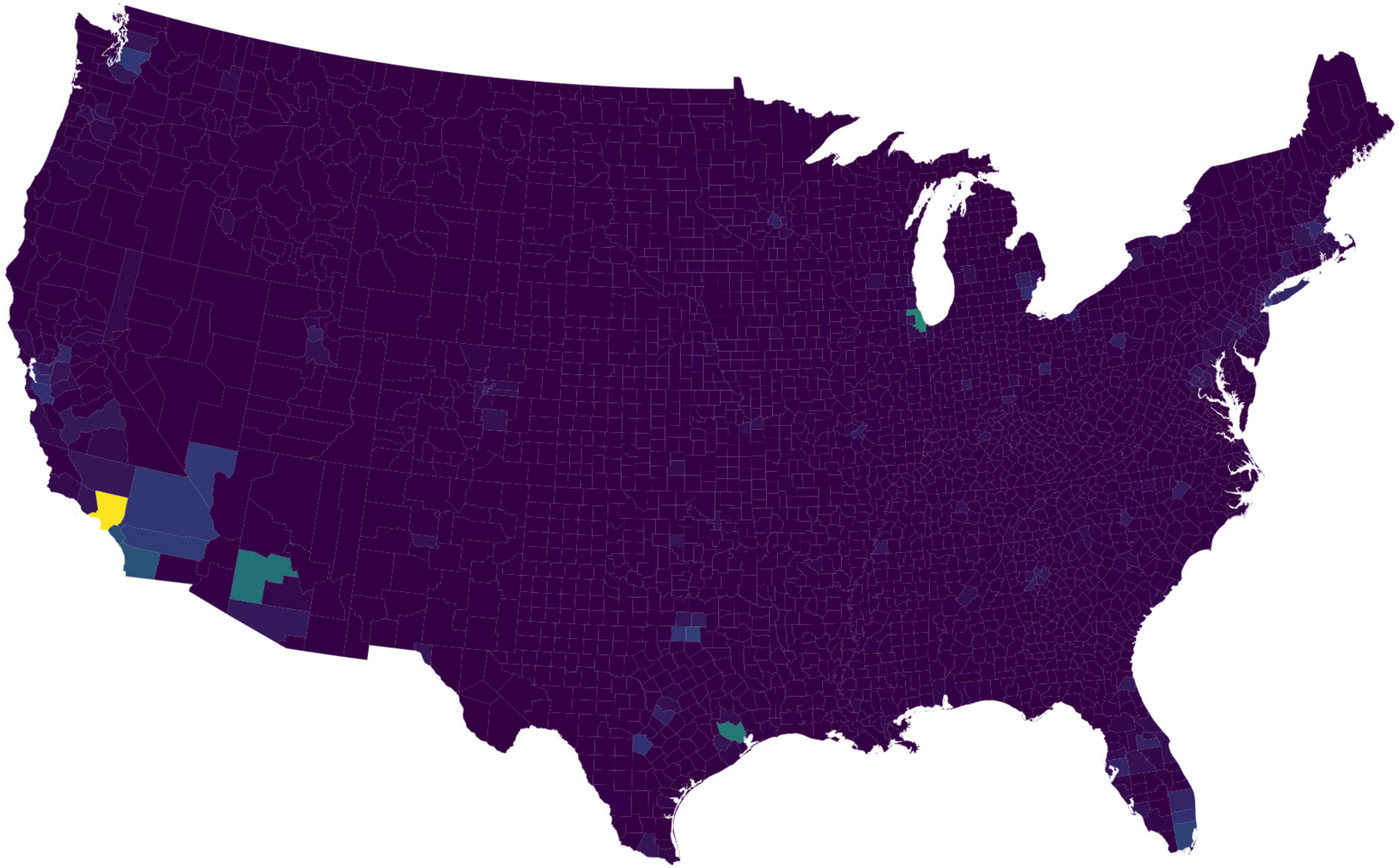
```
ggplot(lower48) +
  geom_sf(aes(fill = value), color = NA) +
  coord_sf(datum = NA) +
  labs(title = "Population by County, 2020",
       fill = "People") +
  theme_minimal()
```



- But . . . scales

# Well, this is getting better

```
ggplot(lower48) +
  geom_sf(aes(fill = value), color = NA) +
  scale_fill_viridis_c(labels = scales::comma) +
  coord_sf(datum = NA) +
  labs(title = "Population by County, 2020",
       fill = "People") +
  theme_minimal()
```



- Doesn’t get better with a standard scale . . . we have to customize a bit

# Let's look at some code

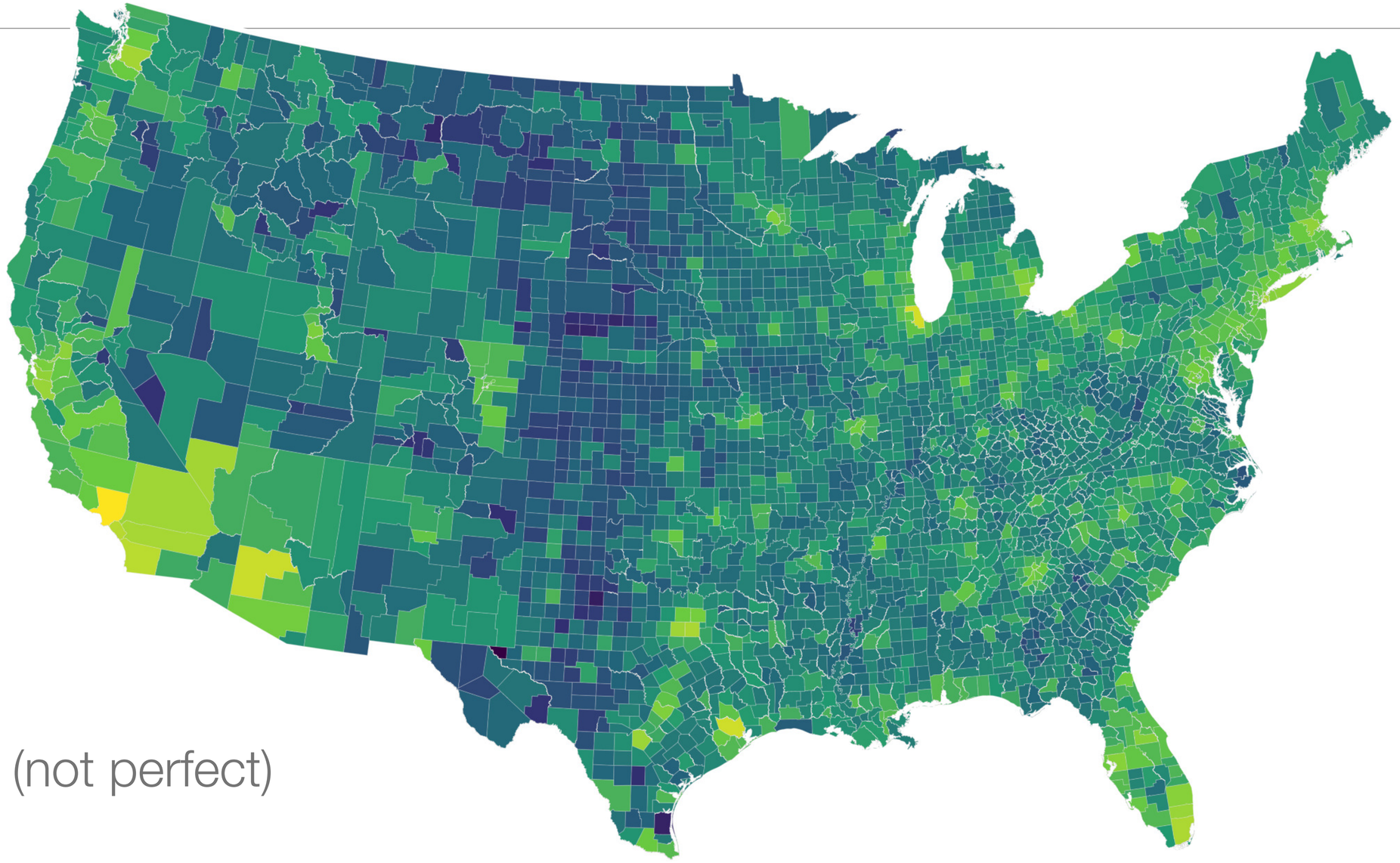
```
p  <- ggplot(lower48, aes(fill = value)); p
p1 <- p + geom_sf(color = "gray90", linewidth = 0.05) + coord_sf(datum = NA); p1
p2 <- p1 +
  scale_fill_viridis_c(
    trans = "log10",
    breaks = c(1e4, 5e4, 1e5, 5e5, 1e6, 5e6, 1e7),
    labels = label_number(scale_cut = cut_short_scale())
  ) +
  labs(title = "Population by County, 2020", fill = NULL); p2
p2 + theme_map() + guides(fill = guide_colorbar(barheight = unit(60, "pt")))
```

- **`viridis()`**
  - A color scaling package that works to try to create contrast-y scales

```
scale_fill_viridis_c(
    trans = "log10",
    breaks = c(1e4, 5e4, 1e5, 5e5, 1e6, 5e6, 1e7),
    labels = label_number(scale_cut = cut_short_scale())
```

  - **`scale_fill_viridis_c`**
    - use a continuous scale
  - **`trans = "log10"`**
    - apply a logarithmic transformation (pull the outliers in)
  - **`breaks = c(1e4, 5e4, 1e5, 5e5, 1e6, 5e6, 1e7)`**
    - breaks shown on the legend
  - **`labels = label_number(scale_cut = cut_short_scale())`**
    - give me human readable scales

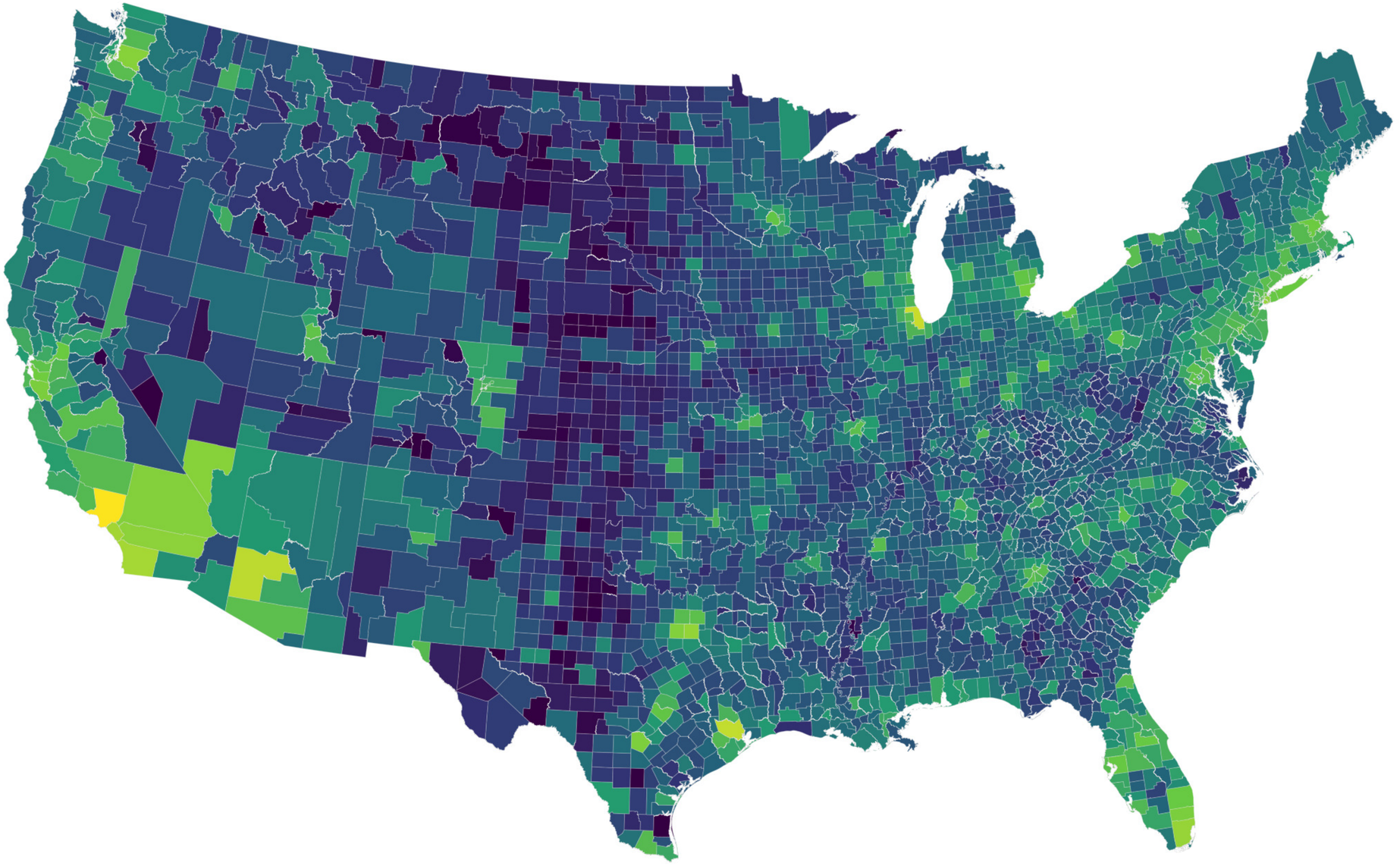
# We’ll use viridis() scaling, log-transformed



- Better (not perfect)

# I want to squish the bottom of the scale a bit

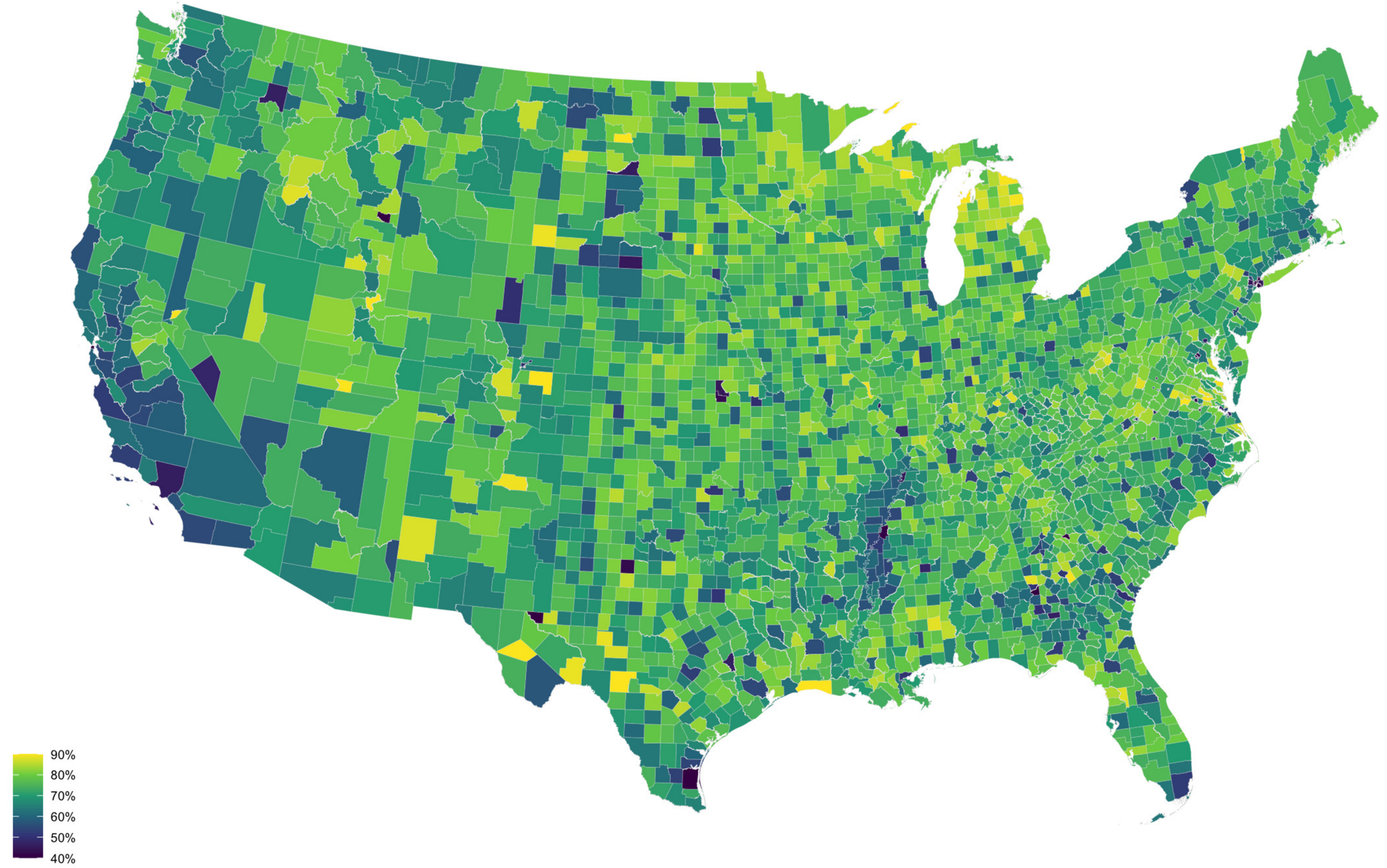
```
vmax <- max(lower48$value, na.rm = TRUE)

p  <- ggplot(lower48, aes(fill = value)); p

p1 <- p + geom_sf(color = "gray90", linewidth = 0.05) + coord_sf(datum =
NA); p1

p2 <- p1 +
  scale_fill_viridis_c(
    trans  = "log10",
    limits = c(1e3, vmax),           # start color mapping at 1,000
    oob    = squish,                 # values <1,000 get the lowest color
    breaks = c(1e3, 1e4, 1e5, 1e6, 1e7),
    labels = label_number(scale_cut = cut_short_scale())
  ) +
  labs(title = "Population by County, 2020", fill = NULL); p2
p2 + theme_map() + guides(fill = guide_colorbar(barheight = unit(60, "pt")))
```



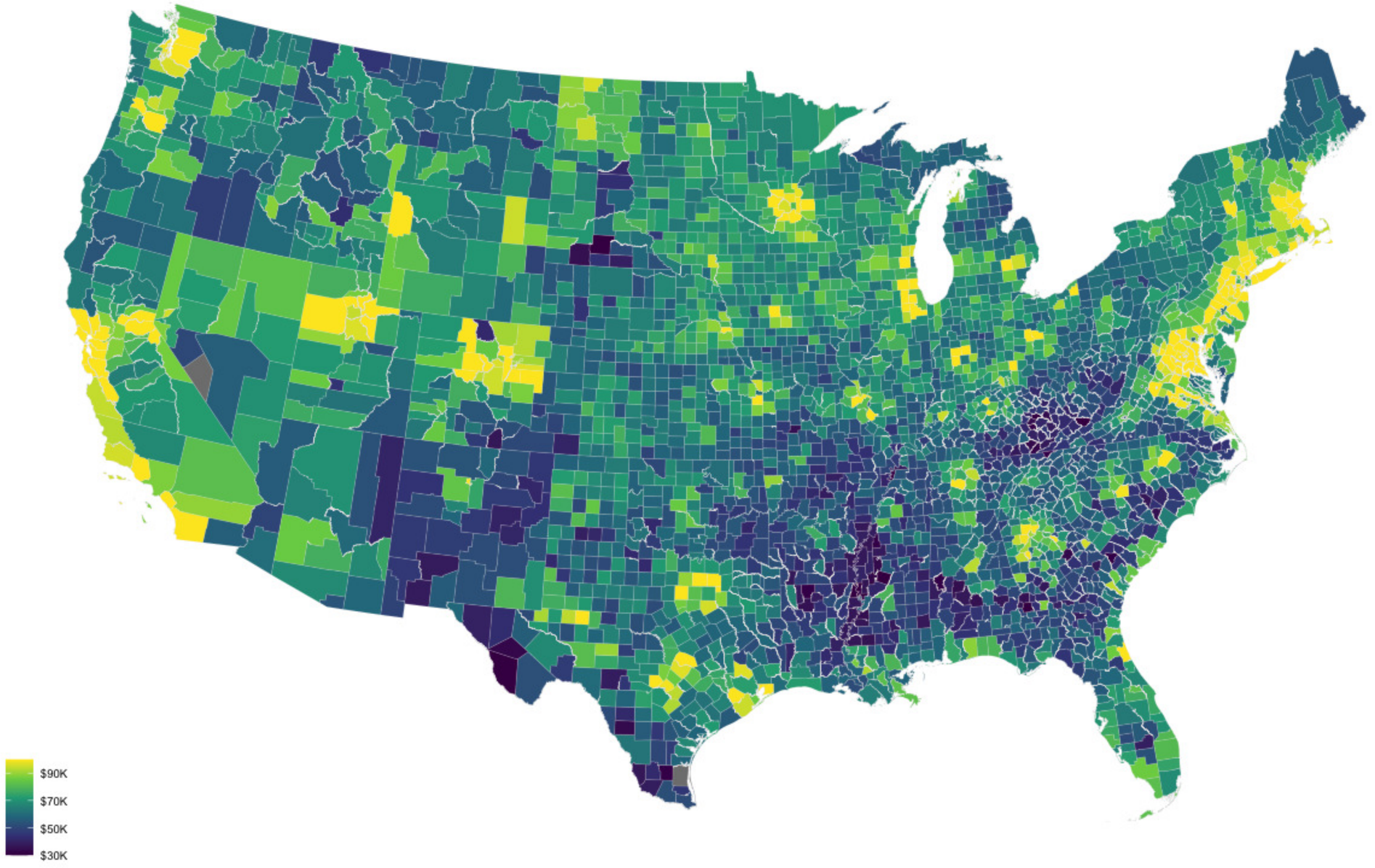
# What about other variables?

- Homeownership
- Median household income
- Educational attainment

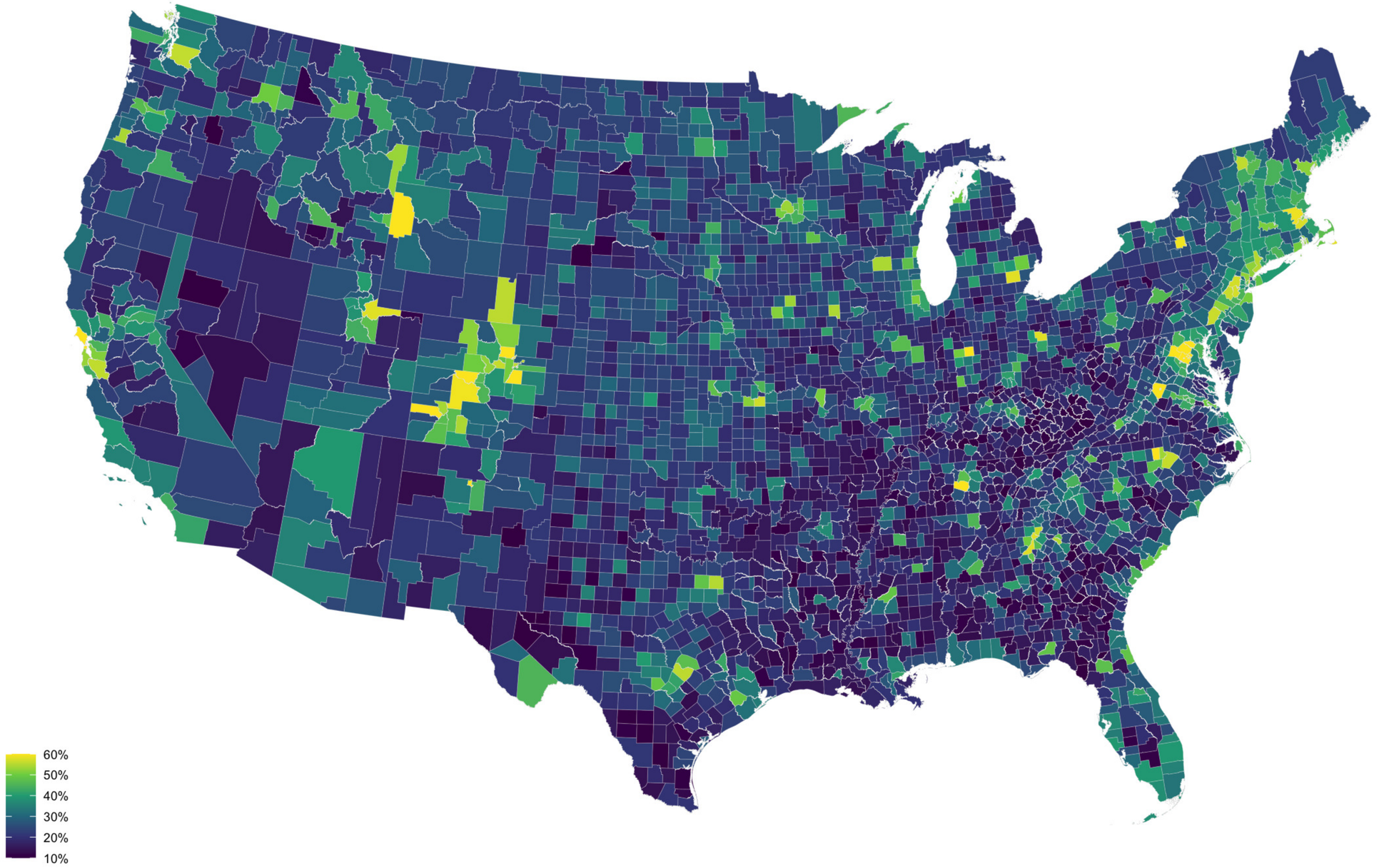
Homeownership Rate by County, 2023



Median Household Income by County, 2023



Adults 25+ with Bachelor's Degree or Higher, 2023



# In R: Additional Data Examples

- Street Maps with Open Street Map

# Terrain (Raster) Data

- The second major kind of data
- Rather than polygons, we focus on continuous data across an area

# There is much more to GIS!

- GIS is not just visualization, cartography, and pretty maps
  - Point Pattern Analysis
  - Spatial Autocorrelation
  - Spatial Regression
  - Spatial Interpolation
- And more