

# 13 - Spatial Data

*ST 597 | Spring 2017  
University of Alabama*

*12-spatial.pdf*

## Contents

<b>1 Maps in R</b>	<b>2</b>
1.1 maps package data . . . . .	2
1.2 Map Boundaries . . . . .	3
1.3 Map Projections . . . . .	5
1.4 Adding layers . . . . .	6
<b>2 Choropleth Maps</b>	<b>7</b>
2.1 Filling in the polygons . . . . .	7
<b>3 Visualizing Gun Violence</b>	<b>8</b>
3.1 Get the Gun Violence Data . . . . .	9
3.2 Create another column called PrGun that records the proportion of Murders that are Gun related (Gun Murders). . . . .	9
3.3 Join the PrGun data with the US State Map Data . . . . .	10
3.4 Create choropleth Maps . . . . .	11
<b>4 ggmap package</b>	<b>14</b>
<b>5 Other Spatial Mapping Tools</b>	<b>16</b>
5.1 Spatial Analysis . . . . .	16

---

## Required Packages and Data

```
library(tidyverse)
library(maps)          # ensure that maps is loaded after tidyverse (fixed in new version)
library(ggmap)
library(stringr)
library(rvest)
library(readr)
```

# 1 Maps in R

## 1.1 maps package data

There are several ways to create a map in R. Here, We will make use of the data available in the `maps` package.

```
library(maps)

help(package="maps")      # Documentation for `maps` package

data(package="maps")      # data sets in `maps` package
```

To make a map with `ggplot2`, we need to reformat the `maps` data using the function `map_data()` (from `ggplot2` package).

```
library(tidyverse)          # plotting functions (including map_data)
library(maps)               # for map data

world_map = map_data(map="world")           # polygons for world
state_map = map_data(map="state")            # polygons for US states
ky_state_map = map_data(map="state", region="kentucky") # polygon for Kentucky
county_map = map_data(map="county")          # polygons for US counties
ky_county_map = map_data("county", region="kentucky") # polygon for counties in Kentucky
```

The `map_data()` function calls `map()` (type `?map` to see details) and then does some formatting to convert it into a *data frame*. So the arguments to `map_data()` are the same as those for `map()`. Notice the first argument is the `database` = which specifies `usa`, `state`, `county`, `world`, and others. The `regions` = argument can limit the map to certain regions (this differs depending on the database).

We can examine one of these data frames

```
head(state_map)
#>   long   lat group order  region subregion
#> 1 -87.46 30.39     1     1 alabama      <NA>
#> 2 -87.48 30.37     1     2 alabama      <NA>
#> 3 -87.53 30.37     1     3 alabama      <NA>
#> 4 -87.53 30.33     1     4 alabama      <NA>
#> 5 -87.57 30.33     1     5 alabama      <NA>
#> 6 -87.59 30.33     1     6 alabama      <NA>
```

which shows a data frame with 6 columns and with rows corresponding to geographical points (in longitude and latitude). This information corresponds to a set of polygons that define the map boundaries. The `group` column is important if there are multiple polygons (for example, one polygon per state) and the `order` column specifies the sequence for drawing the lines of the borders.

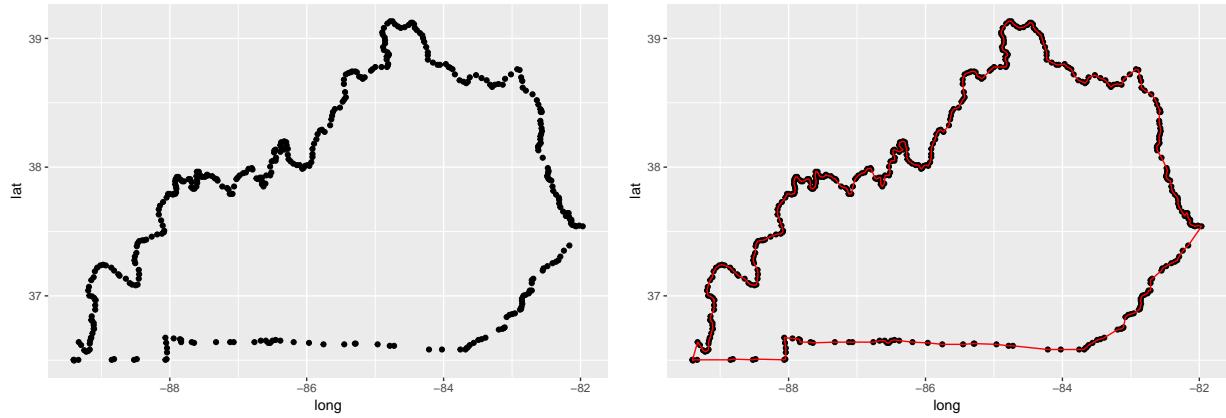
## 1.2 Map Boundaries

Here is an example of the polygon for the state of Kentucky.

```
#- first create a ggplot object with the data
ky = ggplot(ky_state_map, aes(x=long, y=lat))

#- plot the points
ky + geom_point()

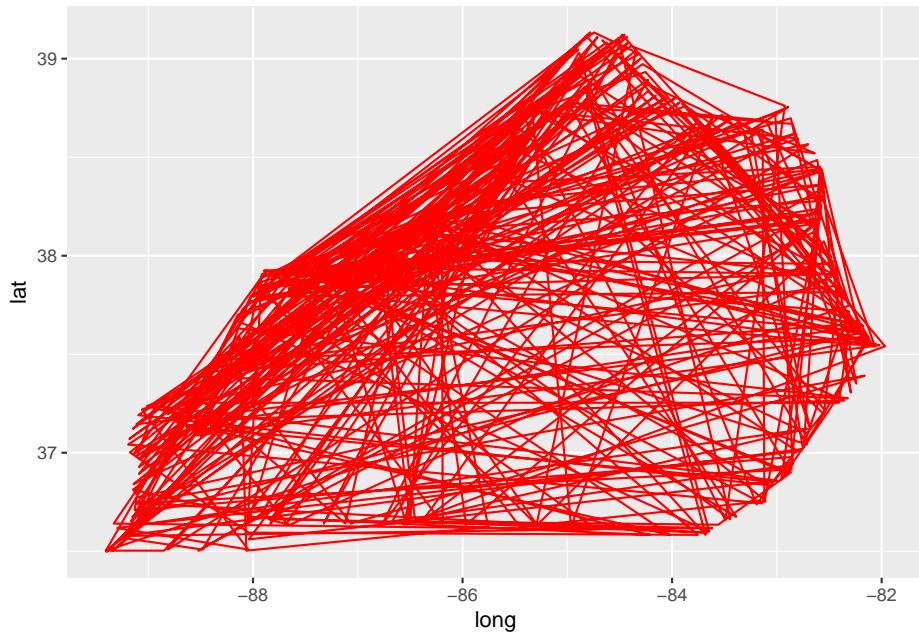
#- connect the lines
ky + geom_point() + geom_path(color="red")
```



Notice that the data in `ky_state_map` is ordered by the `order` column. This is no accident - it controls the how the lines are connected. If we change the order of the data, we will get a very strange result:

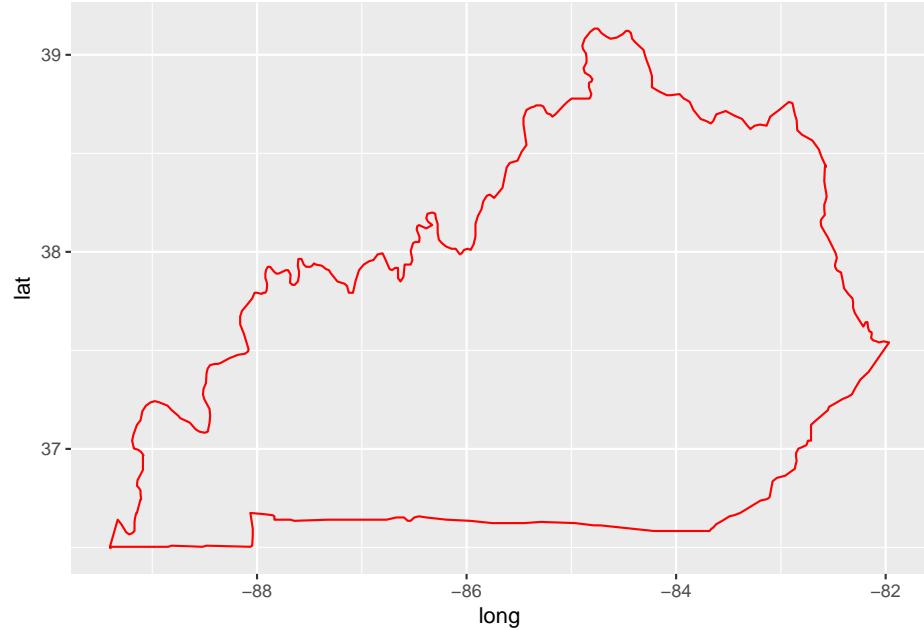
```
unordered = sample_frac(ky_state_map)           # randomly reorder the rows

ggplot(unordered, aes(x=long, y=lat)) + geom_path(color="red")
```



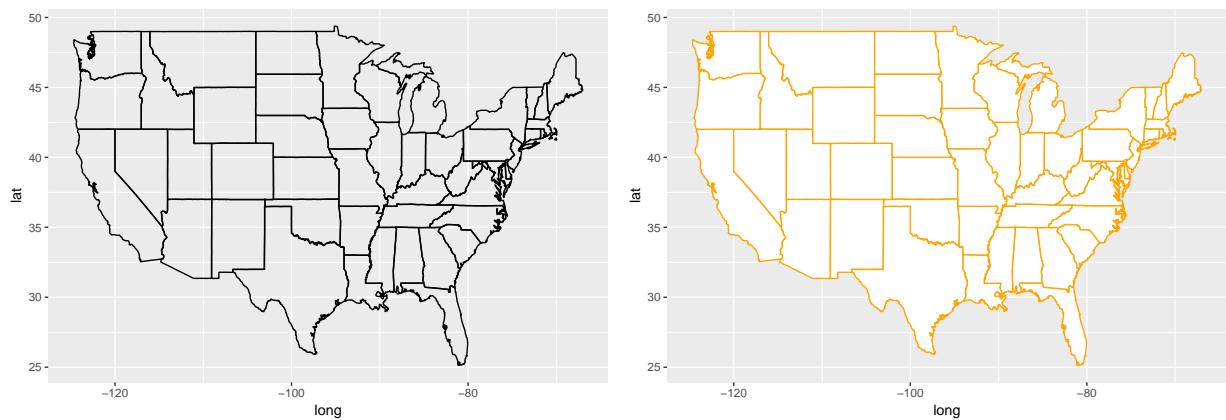
If you find that the data is not ordered properly (usually result of merging datasets) - just reorder (arrange()) by group and order

```
ggplot(arrange(unordered, group, order), aes(x=long, y=lat)) +  
  geom_path(color="red")
```



If we have map data with multiple regions/groups, then use the group aesthetic

```
#- US states data with group=group aesthetic  
us = ggplot(state_map, aes(x=long, y=lat, group=group))  
  
#- add geom_path  
us + geom_path()  
  
#- or use geom_polygon, which is just like geom_path, but allows a fill color  
us + geom_polygon(fill="white", color="orange")
```



## 1.3 Map Projections

Maps refer to locations on a non-flat earth. As such, euclidean distances and areas on a 2D map can be deceptive. Transformations (or *spatial projections*) can help show a map with better proportions. In ggplot2, use the `coord_map()` and `coord_quickmap` functions

```
usmap = us + geom_polygon(fill="white", color="black")           # initial map object

#- approximate (for small regions)
usmap + coord_quickmap() + ggtitle("quickmap")

#- default coord_map(): mercator projection
usmap + coord_map() + ggtitle("Mercator")

#- polyconic projection
usmap + coord_map("polyconic") + ggtitle("Polyconic")

#- albers projection
usmap + coord_map("albers", lat0=29.5, lat1=45.4) + ggtitle("Albers")
```



Some Map Projection Info: <http://egsc.usgs.gov/isb//pubs/MapProjections/projections.html>

## 1.4 Adding layers

We can add additional info, like state capital cities. The `us.cities` in the `maps` package contains lat, long, and population (2006 estimate) for 1005 of the largest cities in the US. Let's add this to our US map

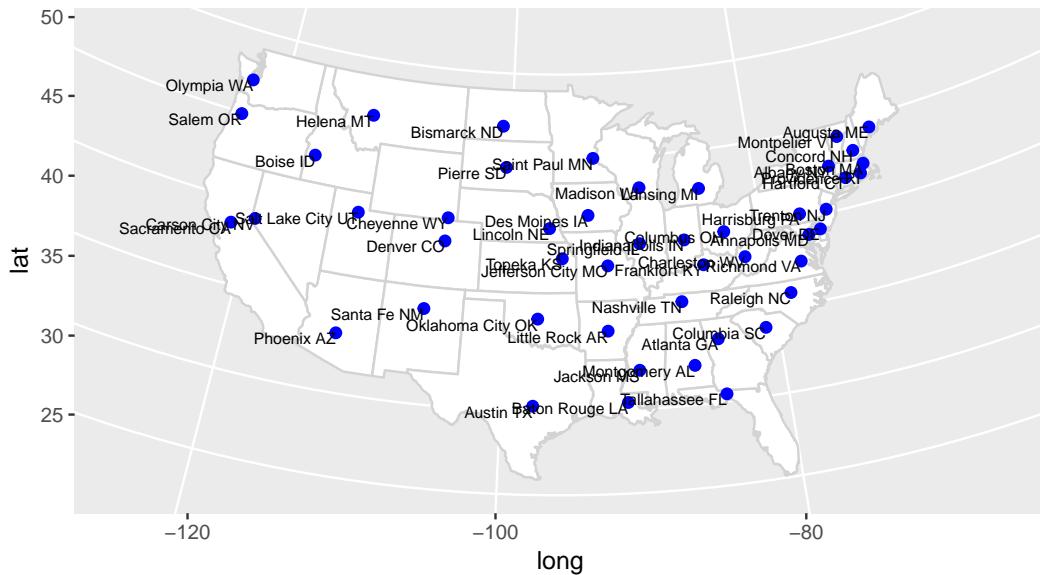
```
#- Create base map
base_map = ggplot(state_map, aes(x=long, y=lat)) +
  geom_polygon(aes(group=group), fill="white", color="lightgrey") +
  coord_map("albers", lat0=29.5, lat1=45.4)

## Notice that the group aesthetic is moved to geom_polygon(). If this was
## left in ggplot(), then every other geom must use the group aesthetic.
## Since geom_point() and geom_text() use the us.capitals data,
## there is no group column.

#- get us city info
library(maps)
data(us.cities)      # notice the `capital == 2` indicates state capital

#- I only want capital cities for continental US (remove Alaska and Hawaii)
us.capitals = us.cities %>% filter(capital==2, !country.etc %in% c("AK", "HI"))

#- Make the map
base_map +
  geom_point(data=us.capitals, color="blue", size=2) +
  geom_text(data=us.capitals, aes(label=name),
            size=2.5, vjust=1, hjust=1)
```



Notice that `aes(x=long, y=lat)` is inherited from `base_map` for `geom_point()` and `geom_text()`.

## 2 Choropleth Maps

### 2.1 Filling in the polygons

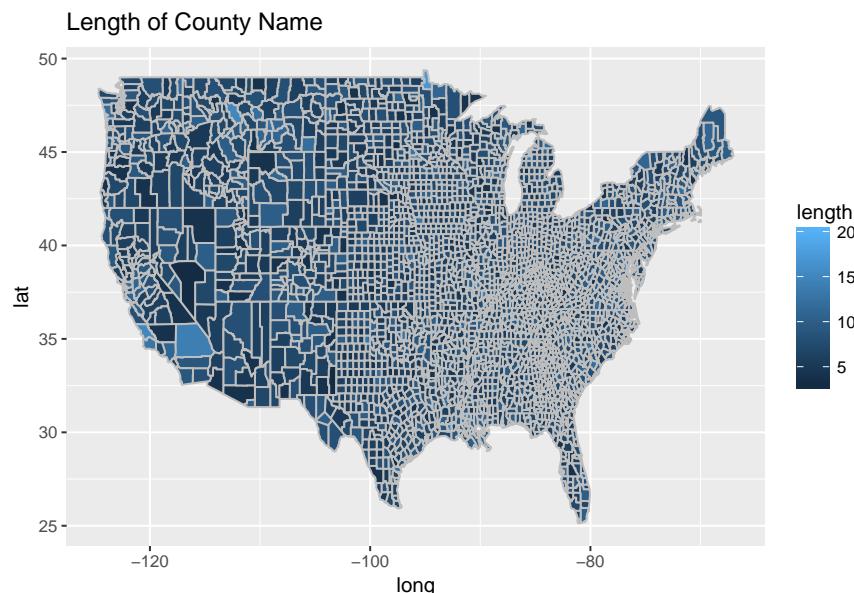
Choropleth Maps are thematic maps in which areas are shaded or patterned in proportion to the measurement of the statistical variable being displayed on the map, such as population density or per-capita income (quoted from [wikipedia](#)). This means that we need to have additional data to map a color to.

One way to do this is by adding a column to the map data with the value we want to represent with color. For this example, we will create a column `length` for the length of the county name and map this to a fill color

```
library(stringr)      # for the str_length() function
county_data_map = county_map %>% mutate(length=str_length(subregion))
head(county_data_map)
#>   long  lat group order  region subregion length
#> 1 -86.51 32.35     1     1 alabama autauga     7
#> 2 -86.53 32.35     1     2 alabama autauga     7
#> 3 -86.55 32.37     1     3 alabama autauga     7
#> 4 -86.56 32.38     1     4 alabama autauga     7
#> 5 -86.58 32.38     1     5 alabama autauga     7
#> 6 -86.59 32.38     1     6 alabama autauga     7
```

Now for the map

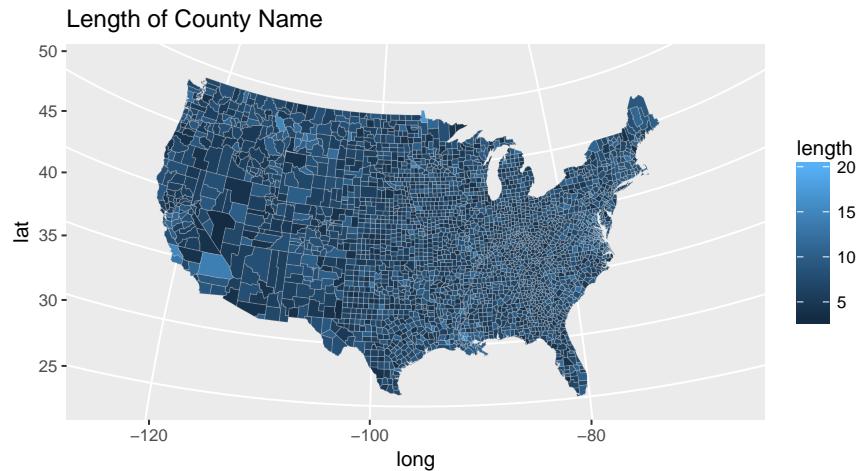
```
ggplot(county_data_map, aes(x=long, y=lat, group=group)) +      # use new data_map
       geom_polygon(aes(fill = length), color='grey') +          # add fill color and grey border
       ggtitle("Length of County Name")
```



But we can dress this up a bit. The borders are overwhelming, so we will modify the size and transparency.

However, we cannot do this from `geom_polygon()` because `alpha` will control the fill color and not line color. Also, we will add a nice `polyconic` projection.

```
ggplot(county_data_map, aes(x=long, y=lat, group=group)) +      # use new data_map
  geom_polygon(aes(fill = length), color=NA) +      # add fill color (no border color)
  geom_path(color="grey", size=.1, alpha=.2) +      # add county outline
  coord_map("polyconic") +      # add projection
  ggtitle("Length of County Name")
```



Note that you can adjust the color mapping with `scale_fill_brewer()`, `scale_fill_continuous()`, etc. [See RGraphics 12 for details.]

### 3 Visualizing Gun Violence

Suppose we would like to visualize the gun violence data from [http://en.wikipedia.org/wiki/Gun\\_violence\\_in\\_the\\_United\\_States\\_by\\_state](http://en.wikipedia.org/wiki/Gun_violence_in_the_United_States_by_state). Specifically, we are going to plot the **proportion of Murders that are Gun related** by US state.

Here are the steps:

1. Get the Gun Violence Data
2. Create another column called `PrGun` that records the proportion of Murders that are Gun related (Gun Murders).
3. Join the `PrGun` data with the US counties Map Data
4. Create choropleth Maps

### 3.1 Get the Gun Violence Data

```
#- get wiki table
library(rvest)
url = 'https://en.wikipedia.org/wiki/Gun_violence_in_the_United_States_by_state'
x = read_html(url) %>% html_node("table.wikititable") %>% html_table()

#- add column names and clean
colnames(x) = c('State', 'Pop', 'PopDensity', 'Murders', 'GunMurders',
               'GunOwnership', 'MurderRate', 'GunMurderRate')
guns = type_convert(x) # removes commas and %

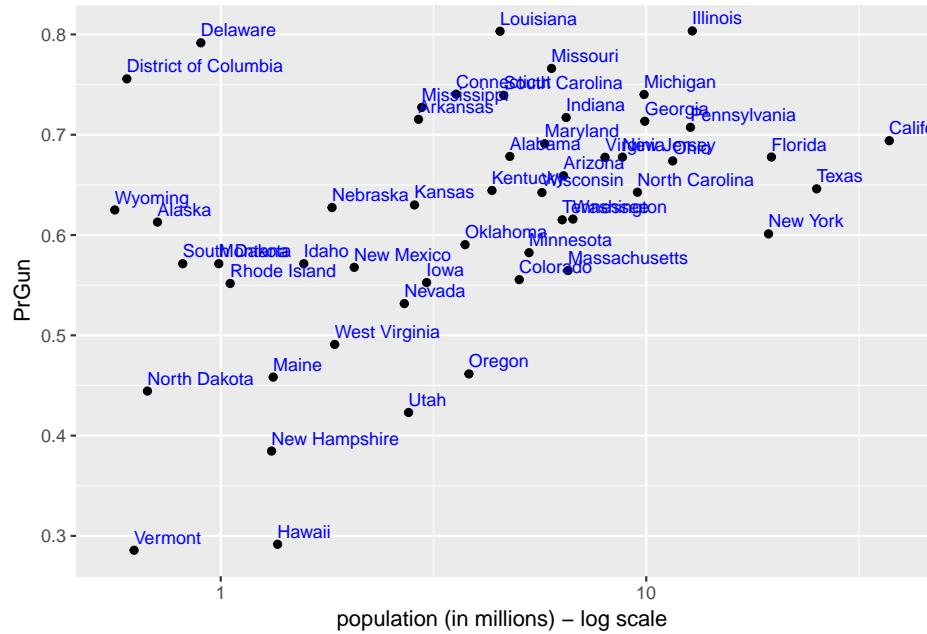
#- how does it look?
glimpse(guns)
#> Observations: 51
#> Variables: 8
#> $ State      <chr> "Alabama", "Alaska", "Arizona", "Arkansas", "California", "Colo...
#> $ Pop        <dbl> 4779736, 710231, 6392017, 2915918, 37253956, 5029196, 3574097, ...
#> $ PopDensity <dbl> 94.650, 1.264, 57.050, 56.430, 244.200, 49.330, 741.400, 470.70...
#> $ Murders    <dbl> 199, 31, 352, 130, 1811, 117, 131, 48, 131, 987, 527, 24, 21, 4...
#> $ GunMurders <dbl> 135, 19, 232, 93, 1257, 65, 97, 38, 99, 669, 376, 7, 12, 364, 1...
#> $ GunOwnership <chr> "51.7%", "57.8%", "31.1%", "55.3%", "21.3%", "34.7%", "16.7%", ...
#> $ MurderRate  <dbl> 4.2, 4.4, 5.5, 4.5, 4.9, 2.3, 3.7, 5.3, 21.8, 5.0, 5.3, 1.8, 1...
#> $ GunMurderRate <dbl> 2.8, 2.7, 3.6, 3.2, 3.4, 1.3, 2.7, 4.2, 16.5, 3.4, 3.8, 0.5, 0...
```

### 3.2 Create another column called PrGun that records the proportion of Murders that are Gun related (Gun Murders).

```
guns = mutate(guns, PrGun = GunMurders/Murders)
```

Quick visualization

```
ggplot(guns, aes(x=Pop/1E6, y=PrGun)) +
  geom_point() +
  geom_text(aes(label=State), size=3, vjust=-.5, hjust=0, color="blue") +
  scale_x_log10("population (in millions) - log scale")
```



### 3.3 Join the PrGun data with the US State Map Data

First, let's examine the map data

```
state_map = map_data(map="state")           # polygons for US states
glimpse(state_map)
#> Observations: 15,537
#> Variables: 6
#> $ long      <dbl> -87.46, -87.48, -87.53, -87.53, -87.57, -87.59, -87.59, -87...
#> $ lat       <dbl> 30.39, 30.37, 30.37, 30.33, 30.33, 30.33, 30.31, 30.29, 30.28, 30.2...
#> $ group     <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
#> $ order      <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, ...
#> $ region    <chr> "alabama", "alabama", "alabama", "alabama", "alabama", "al...
#> $ subregion <chr> NA, ...
```

This has a `region` column with the state name in all lowercase.

Joining is straightforward in this example (sometimes we need an intermediary data set to help with the links). We only need to change the `State` column of `guns` to lowercase.

```
library(stringr)
guns = mutate(guns, State = str_to_lower(State))    # guns$State = str_to_lower(guns$State)
```

We can do a quick check to see what values we have in each data set using the `setdiff()` function.

```
#- values in state_map$region, but not in guns$State
anti_join(state_map, guns, by=c("region" = "State"))
#> [1] long      lat       group     order      region      subregion
#> <0 rows> (or 0-length row.names)

#- values in guns$State, but not state_map$region
anti_join(guns, state_map, by=c("State" = "region"))
```

```
#>      State      Pop PopDensity Murders GunMurders GunOwnership MurderRate GunMurderRate
#> 1 alaska 710231     1.264      31       19      57.8%      4.4      2.7
#> 2 hawaii 1360301    216.800      24        7      6.7%      1.8      0.5
#> PrGun
#> 1 0.6129
#> 2 0.2917
```

It looks like alaska and hawaii are not in the state\_map (because these are mainland US map). We will leave out poor hawaii and alaska for this example, but with a little `extra work` we can bring them back in.

Now for the join. I will use `left_join()` to preserve all the map\_data info. (An `inner_join()` would give the same result in this case because there `guns$State` has all the value in `map_data$region`).

```
state_data_map = left_join(state_map, guns, by = c("region" = "State"))
```

### 3.4 Create choropleth Maps

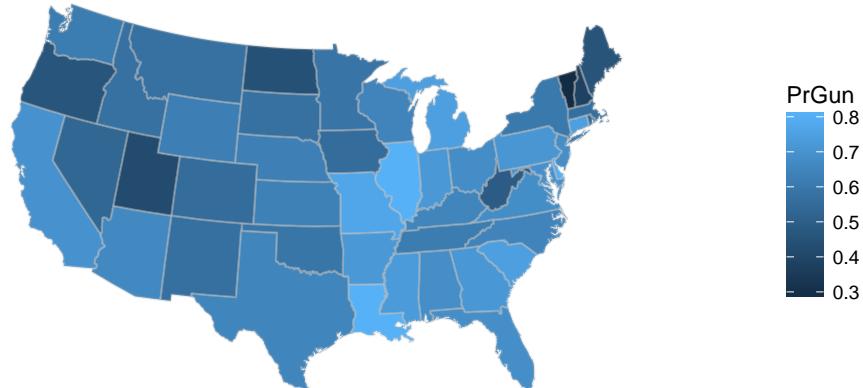
We are going to use the `PrGun` column for the fill color. To get a clean background, I created a new theme object called `map_theme`.

```
#- Make a map_theme
map_theme = theme_bw() + theme(line = element_blank(),
                           axis.text=element_blank(),
                           axis.title.x=element_blank(),
                           axis.title.y=element_blank(),
                           panel.border=element_blank())

#- Put it all together
gmap =
ggplot(state_data_map, aes(x=long, y=lat, group=group)) +
  geom_polygon(aes(fill=PrGun), color=NA) + # add filled polygons
  geom_path(color="grey", alpha=.3) + # add state borders (with alpha)
  coord_map("polyconic") + # set map projection
  ggtitle("Proportion of Murders that are Gun related") +
  map_theme

gmap
```

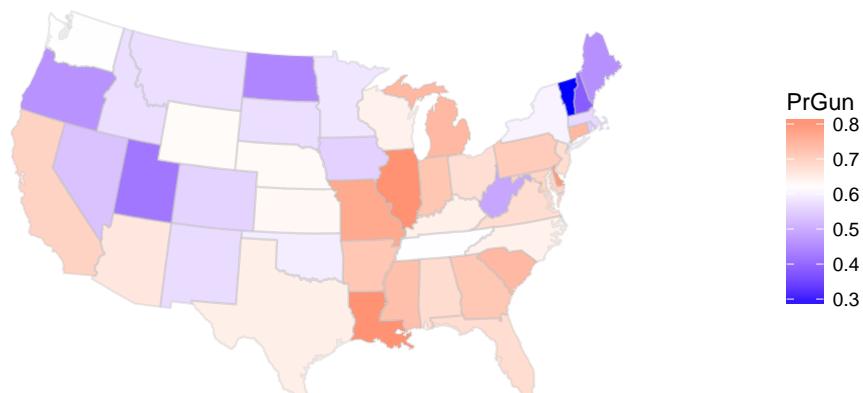
Proportion of Murders that are Gun related



We can change the fill color scale with `scale_fill_` family of functions. Here I will use `scale_fill_gradient2()` to make a diverging color scale with white set to the mean.

```
gmap +  
  scale_fill_gradient2(low="blue", mid='white', high="red",  
    midpoint=mean(guns$PrGun) )  # change fill colors
```

Proportion of Murders that are Gun related



### 3.4.1 Binning

It is common to bin choropleth data to help distinguish the general spatial patterns. There are several ways to create the bins. Here are a few:

- Equal intervals between min and max (`ggplot2::cut_interval()`)
- Equal percentile intervals (so an approximately equal number of observations in each bin) (`ggplot2::cut_number()`)
- Create a midpoint and move equal distance to each side (perhaps standard deviations), etc.

We will use percentile intervals to create decile bins, so approximately 10% of the data will fall into each bin. This minimizes the impact of outliers and skewness.

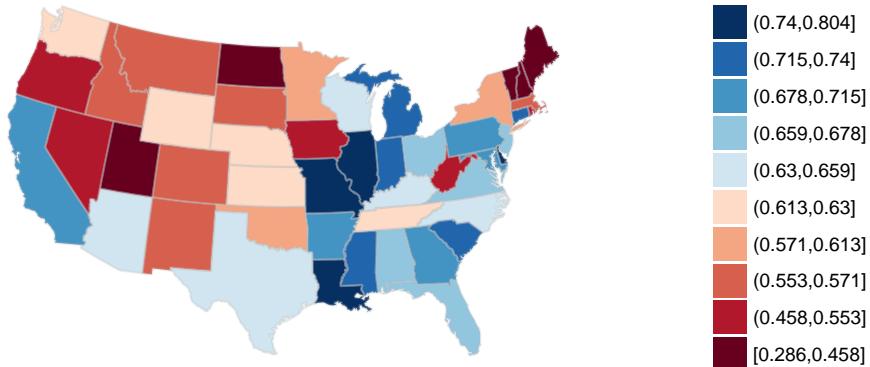
```
guns = mutate(guns, PrGun.binned = cut_number(PrGun, n=10))
```

We need to get this new binned data into the map data. We will do another join and then rebuild the map

```
#- re-join updated data to map
state_data_map = left_join(state_map, guns, by = c("region" = "State"))

#- plot it
ggplot(state_data_map, aes(x=long, y=lat, group=group)) +
  geom_polygon(aes(fill=PrGun.binned), color=NA) +          # add filled polygons
  geom_path(color="grey", alpha=.3) +                         # add state borders
  scale_fill_brewer(type="div", palette="RdBu", name="") +   # change fill colors
  guides(fill=guide_legend(reverse = TRUE)) +                 # reverse legend so
                                                               # largest values at top
  coord_map("polyconic") +                                    # set map projection
  ggtitle("Proportion of Murders that are Gun related") +
  map_theme
```

Proportion of Murders that are Gun related



We used the `scale_fill_brewer()` function to get diverging Red/Blue palette.

## 4 ggmap package

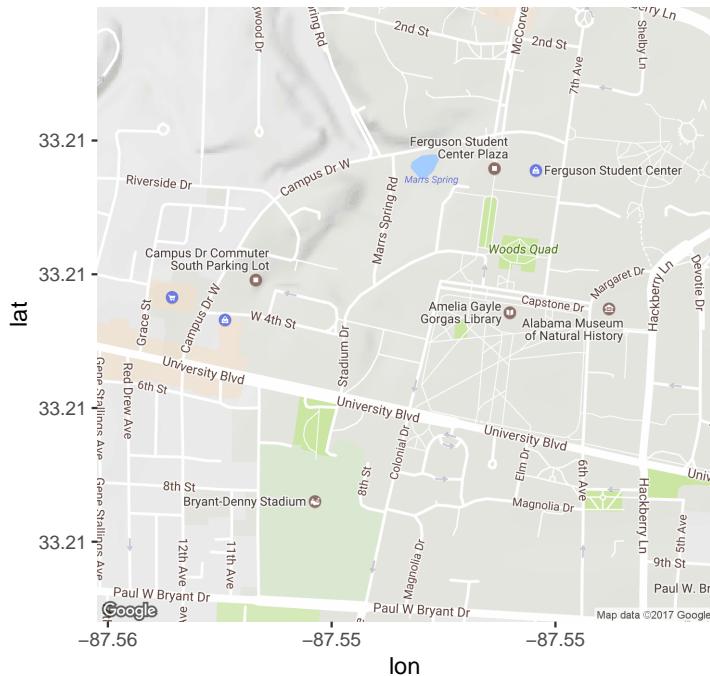
The ggmap packages allows `ggplot2` to work on top of a map from: google maps, open street maps, statemaps, or cloudmade maps. I found a cheatsheet <http://www.nceas.ucsb.edu/~frazier/RSpatialGuides/ggmap/ggmapCheatsheet.pdf> which is helpful for getting started.

First, make sure the package `ggmap` is installed and loaded. Then use the `get_map()` function to retrieve the map data. Finally `ggmap()` function replaces the `ggplot()` function for displaying the basic map information.

```
library(ggmap)    # install.packages("ggmap")

## Get University of Alabama Map (from google maps)
# UA = get_map("University of Alabama", zoom = 15)
UA = get_map("Bidgood Hall, Tuscaloosa, AL", zoom = 16)
#> Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=Bidgood+Hall,+Tuscaloosa,AL&z=16
#> Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Bidgood+Hall,+Tuscaloosa,AL&key=AIzaSyC1QzXWVJLcOOGHdDwvIjyPm7GKUuBZMk

## Plot it
ggmap(UA)
```



Use the help function `?get_map` to see the details of the arguments. If the first argument `location`= is a name or address, it will try to geocode it (geocode means convert to spatial coordinates). For example:

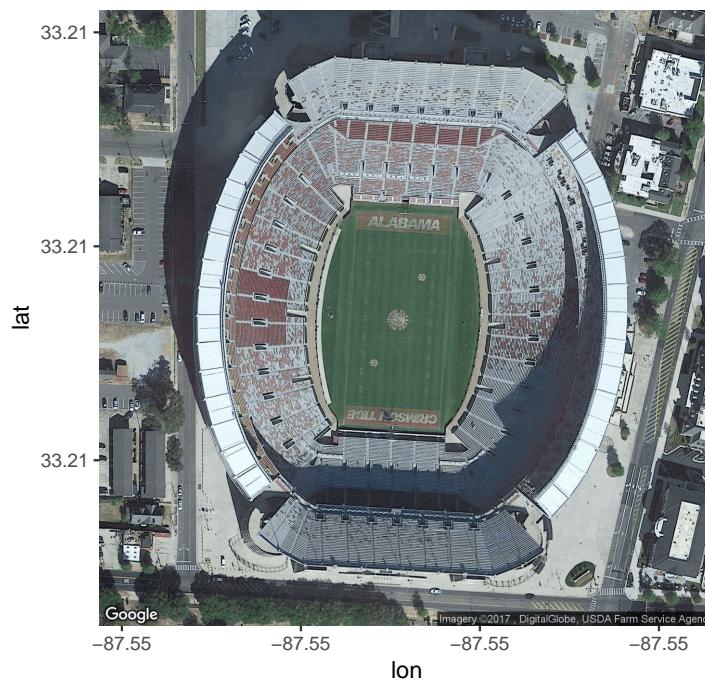
```
library(ggmap)
geocode("University of Alabama")
#> Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=University+of+Alabama&key=AIzaSyC1QzXWVJLcOOGHdDwvIjyPm7GKUuBZMk
#>      lon   lat
#> 1 -87.54 33.21
geocode("Tuscaloosa, AL")
#> Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Tuscaloosa,AL&key=AIzaSyC1QzXWVJLcOOGHdDwvIjyPm7GKUuBZMk
```

```

#>      lon  lat
#> 1 -87.57 33.21
geocode("920 Paul W Bryant Dr, Tuscaloosa, AL 35401")
#> Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=920%20P
#>      lon  lat
#> 1 -87.55 33.21

-- Do you know this address?
stadium.address = "920 Paul W Bryant Dr, Tuscaloosa, AL 35401"
stadium = get_map(stadium.address, zoom=18, maptype="satellite")
#> Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=920+Paul+W+Bryant+D
#> Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=920%20P
ggmap(stadium)

```



Here, we will duplicate the earlier US map with capital cities - but also add and shade the state of Alabama.

```

-- Get the required data
data(us.cities)           # us cities
us.capitals = us.cities %>% filter(capital==2, !country.etc %in% c("AK", "HI"))
AL = map_data(map="state", region="alabama") # polygon for Alabama

-- Get US map
USA = get_map("united states", zoom=4)
#> Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=united+states&zoom=
#> Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=united%20s

-- Now for plots. Notice we need to add the data to the geoms
ggmap(USA) +
  geom_polygon(data=AL, aes(x=long, y=lat),
               color="black", fill="#990000", alpha=.6) +
  geom_point(data=us.capitals, aes(x=long, y=lat),

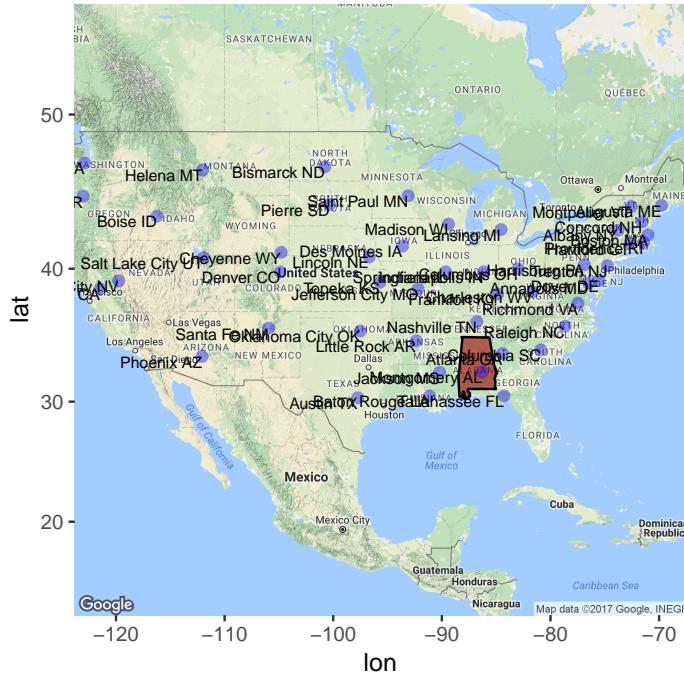
```

```

        color="blue", size=2, alpha=.4) +
geom_text(data=us.capitals, aes(x=long, y=lat, label=name),
           size=2.5, vjust=1, hjust=1) +
ggtitle("State Capitals – with Alabama Highlighted")

```

State Capitals – with Alabama Highlighted



## 5 Other Spatial Mapping Tools

If you start searching around for spatial analysis, you will quickly discover the terms GIS for Geographic Information Systems. The Microsoft Office of GIS systems is ESRI's ArcGIS. However, there are several [open source](#) GIS systems. Most of the geographical analysts I worked with used [QGIS](#). I have used this and found it very easy - especially mixing data with multiple projections. And of course, R could probably be considered a GIS system too.

Spatial data can be represented as raster data (think of pixels) and vector data (think of lines). The primary filetype for vector data is something called a [shapefile](#). R has good capabilities to read (and write) shapefiles. Here are a few R packages that can help: <http://www.nceas.ucsb.edu/scicomp/usecases/ReadWriteESRIShapeFiles>.

### 5.1 Spatial Analysis

Unfortunately, we don't have time to get into the analysis of spatial data. Here are some essentials. Spatial data can be of several forms: areal units (like we did for choropleth), point data (like crime locations), grids/lattice (like measurements from weather stations), and other things (like routes from GPS).

Keep in mind that the primary difference with spatial data (this is common with temporal data too) is described by Tobler's First Law of Geography

"Everything is related to everything else, but near things are more related than distant things."

Tobler W., (1970) "A computer movie simulating urban growth in the Detroit region". Economic Geography, 46(2): 234-240.

So, if you try to model spatial data using regular regression methods do not be surprised to find spatially correlated errors. There is an entire field of spatial statistics and geostatistics to better incorporate this phenomenon.