13 - Spatial Data

Data and Information Engineering

SYS 2202 | Fall 2019

13-spatial.pdf

Contents

Required Packages and Data

```r
library(maps)
library(mapproj)
library(ggmap)
library(rvest)
library(tidyverse)
```

You may need to *install* some of these packages.

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

VA_state_map = map_data(map="state", region="virginia") # polygon for Virginia

county_map = map_data(map="county")                     # polygons for US counties

VA_county_map = map_data("county", region="virginia")   # polygon for counties in Virginia
```

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 -87.5 30.4     1     1 alabama      <NA>
#> 2 -87.5 30.4     1     2 alabama      <NA>
#> 3 -87.5 30.4     1     3 alabama      <NA>
#> 4 -87.5 30.3     1     4 alabama      <NA>
#> 5 -87.6 30.3     1     5 alabama      <NA>
#> 6 -87.6 30.3     1     6 alabama      <NA>
```

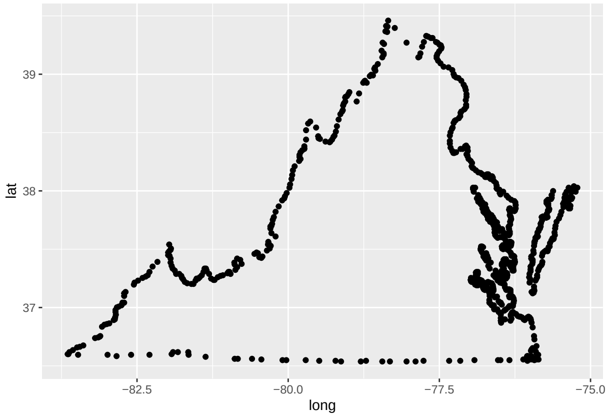
which shows a data frame with 6 columns and with rows corresponding to geographical points (in longitude and latitude). This information corresponds to a set of polygons that define the map boundaries. The group column is important if there are multiple polygons (for example, one polygon per state) and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

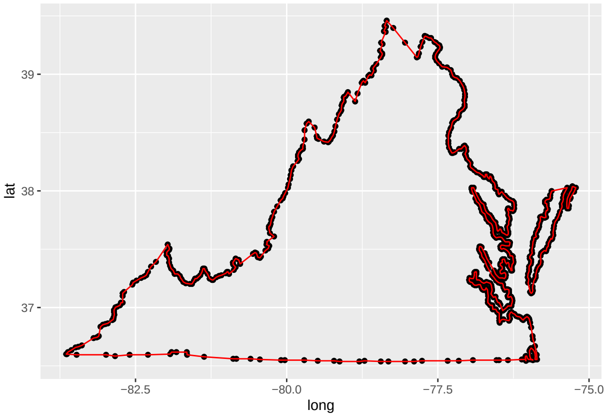
1.2 Map Boundaries

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

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

#- plot the points
VA + geom_point()
```

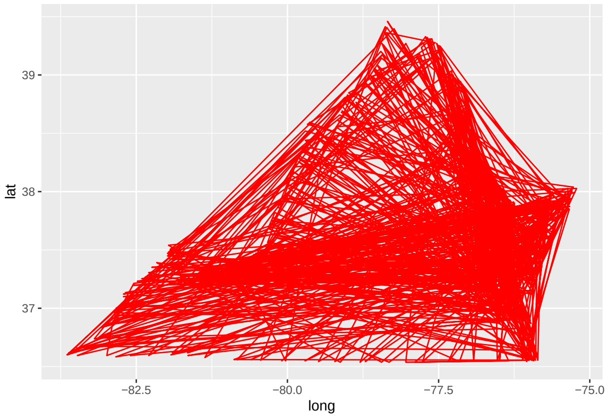
```
#- connect the lines
VA + geom_point() + geom_path(aes(group=group),color="red")
```





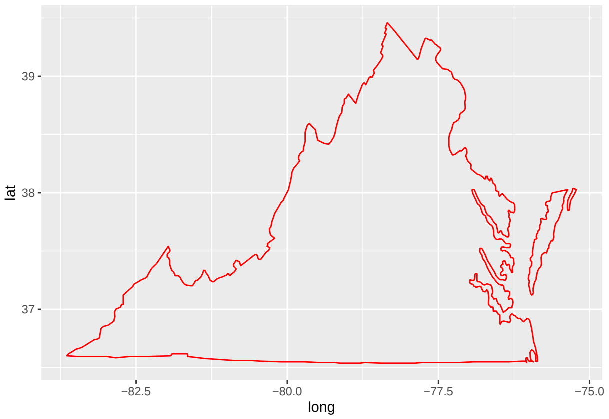
Notice that the data in `VA_state_map` is ordered by the `order` column. This is no accident - it controls the how the lines are connected. If we change the order of the data, we will get a very strange result:

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

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



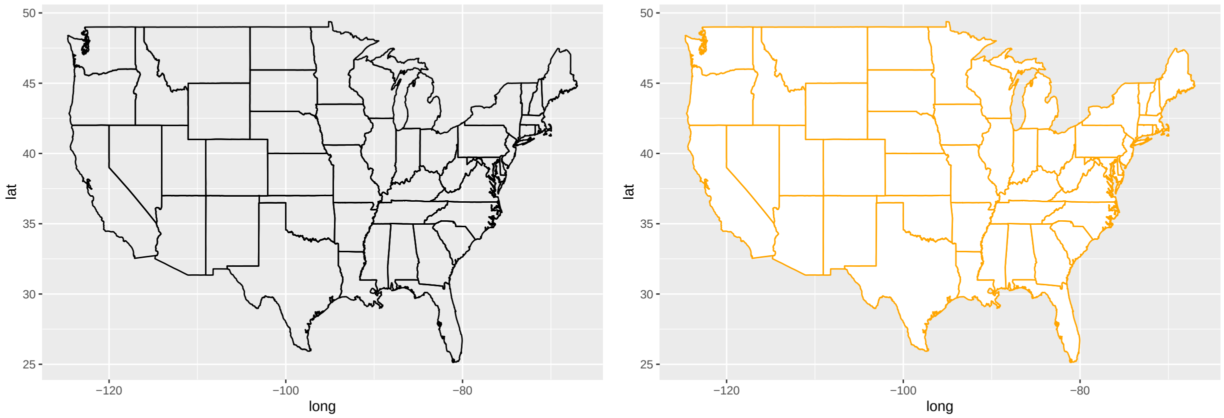
If you find that the data is not ordered properly (usually result of merging datasets) - just reorder (`arrange()`) by `group` and `order`

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

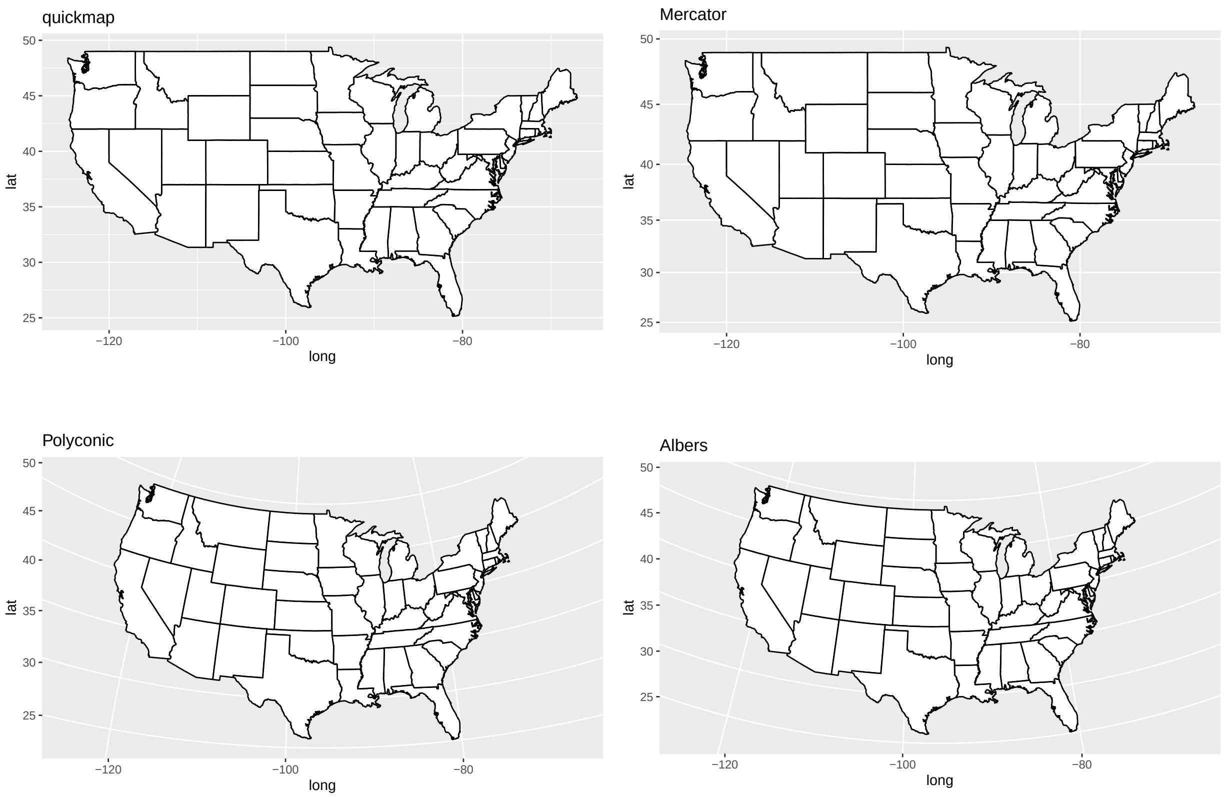


1.3 Map Projections

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")
```



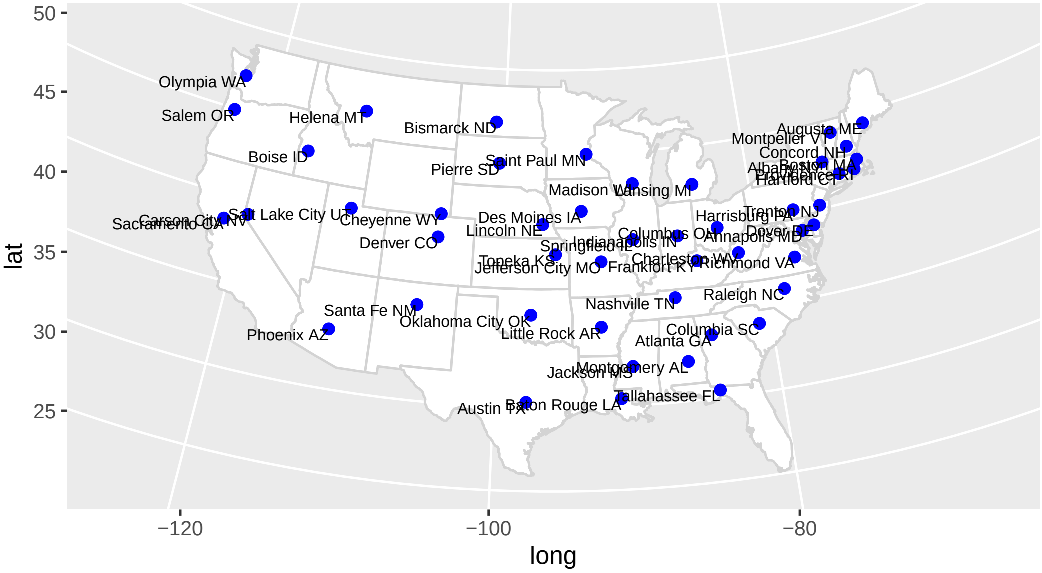
Some Map Projection Info: http://egsc.usgs.gov/isb//pubs/MapProjections/projections.html

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.
##  The us.capitals data, used in geom_point() and geom_text() below, does
##  not have a 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

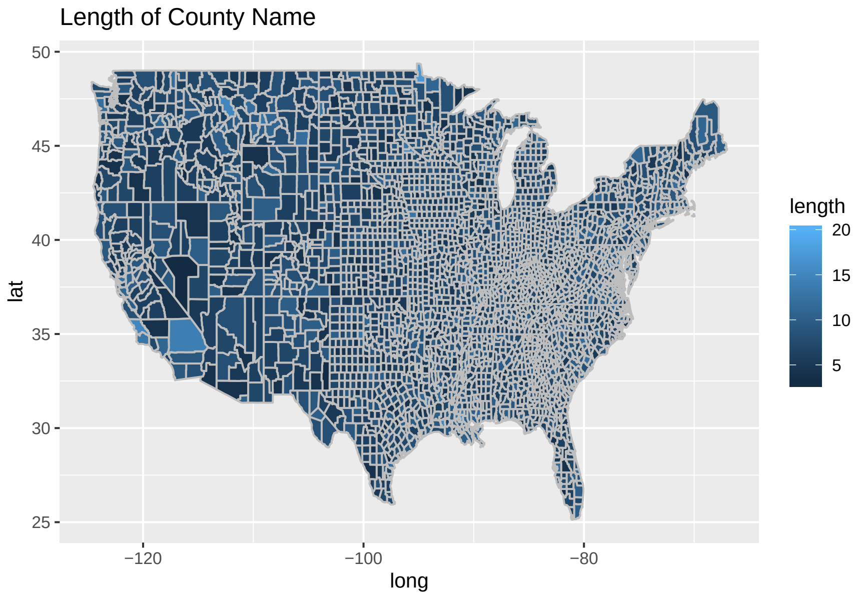
Choropleth Maps are thematic maps in which areas are shaded or patterned in proportion to the measurement of the statistical variable being displayed on the map, such as population density or per-capita income (quoted from wikipedia). This means that we need to have additional data to map a color to.

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 -86.5 32.3     1     1 alabama   autauga      7
#> 2 -86.5 32.4     1     2 alabama   autauga      7
#> 3 -86.5 32.4     1     3 alabama   autauga      7
#> 4 -86.6 32.4     1     4 alabama   autauga      7
#> 5 -86.6 32.4     1     5 alabama   autauga      7
#> 6 -86.6 32.4     1     6 alabama   autauga      7
```

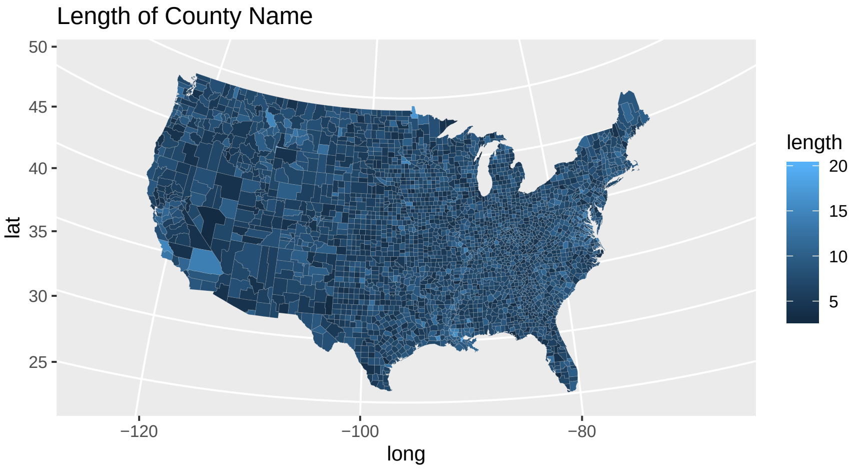
Now for the map

```
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.

3 Visualizing Gun Violence

Suppose we would like to visualize the gun violence data from https://en.wikipedia.org/wiki/Firearm_death_rates_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 (the table under *Murders*)
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 state Map Data
4. Create choropleth Maps

3.1 Get the Gun Violence Data

```
#- get wiki table (Murders)
library(rvest)
url = 'https://en.wikipedia.org/wiki/Firearm_death_rates_in_the_United_States_by_state'
x = read_html(url) %>%
  html_table(fill=TRUE) %>% .[[6]] %>%    # get 6th table
  tbl_df()                                # convert to tibble

#- rename columns (by position) and clean
guns = x %>%
  rename('State'=1, 'Pop'=2, 'PopDensity'=3, 'Murders'=4,
         'GunMurders'=5, 'GunOwnership'=6, 'MurderRate'=7,
         'GunMurderRate'=8) %>%
  type_convert()    # removes commas and %
```

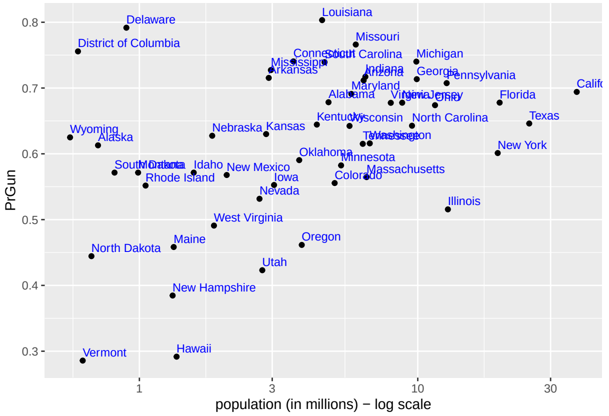
```
#- how does it look?
glimpse(guns)
#> Observations: 51
#> Variables: 8
#> $ State         <chr> "Alabama", "Alaska", "Arizona", "Arkansas", "Cal...
#> $ Pop           <dbl> 4779736, 710231, 6392017, 2915918, 37253956, 502...
#> $ PopDensity    <dbl> 94.65, 1.26, 57.05, 56.43, 244.20, 49.33, 741.40...
#> $ Murders       <dbl> 199, 31, 326, 130, 1811, 117, 131, 48, 131, 987,...
#> $ GunMurders    <dbl> 135, 19, 232, 93, 1257, 65, 97, 38, 99, 669, 376...
#> $ GunOwnership  <chr> "48.9%", "61.7%", "32.3%", "57.9%", "20.1%", "34...
#> $ MurderRate    <dbl> 4.2, 4.4, 4.5, 4.5, 4.9, 2.3, 3.7, 5.3, 21.8, 5....
#> $ GunMurderRate <dbl> 2.8, 2.7, 3.6, 3.2, 3.4, 1.3, 2.7, 4.2, 16.5, 3....
```

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> -87.5, -87.5, -87.5, -87.5, -87.6, -87.6, -87.6, -87...
#> $ lat       <dbl> 30.4, 30.4, 30.4, 30.3, 30.3, 30.3, 30.3, 30.3, 30.3...
#> $ group     <dbl> 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, 1...
#> $ region    <chr> "alabama", "alabama", "alabama", "alabama", "alabama...
#> $ subregion <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 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 `anti_join()` function (also see `setdiff()`).

```
#- 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"))
#> # A tibble: 2 x 9
#>   State     Pop PopDensity Murders GunMurders GunOwnership MurderRate
#>   <chr>   <dbl>      <dbl>   <dbl>      <dbl> <chr>             <dbl>
#> 1 alas~  7.10e5       1.26      31         19 61.7%               4.4
#> 2 hawa~  1.36e6     217.        24          7 45.1%               1.8
#> # ... with 2 more variables: GunMurderRate <dbl>, PrGun <dbl>
```

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.

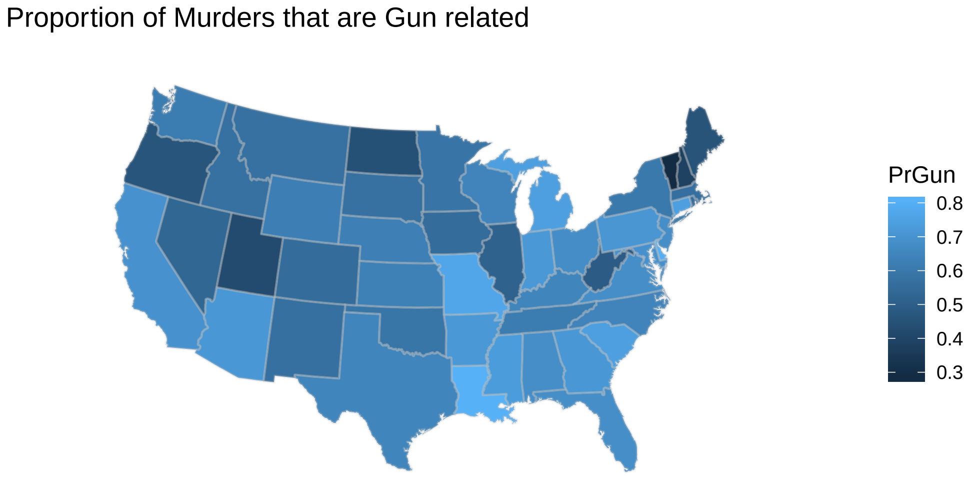
Now for the join. I will use `left_join()` to preserve all the `map_data` info. (An `inner_join()` would give the same result in this case because there `guns$State` has all the value in `map_data$region`).

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

3.4 Create choropleth Maps

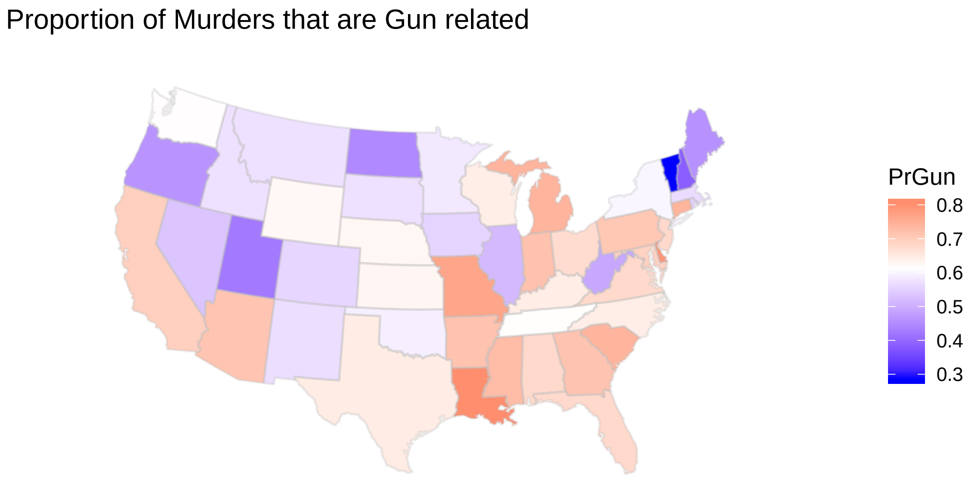
We are going to use the `PrGun` column for the fill color. To get a clean background, I created a new theme object called `map_theme`.

```
#- 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
```



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
```



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:

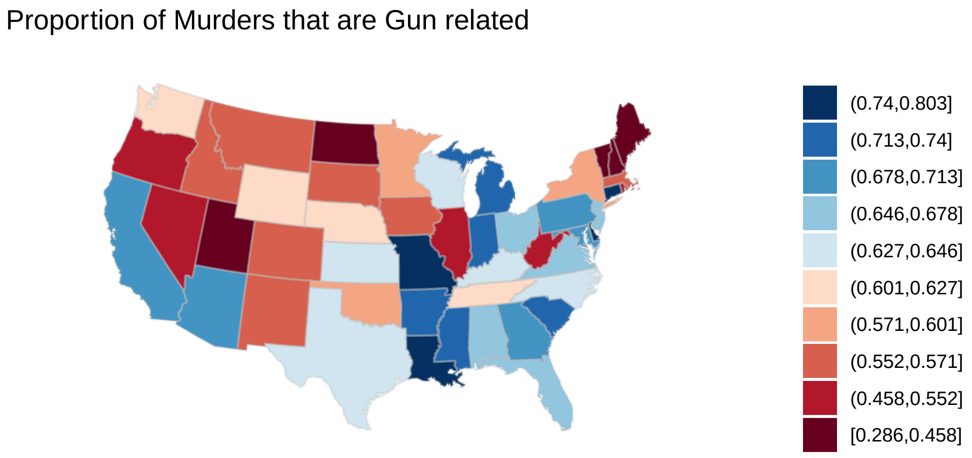
- Equal intervals between min and max (`ggplot2::cut_interval()`)
- Equal percentile intervals (so an approximately equal number of observations in each bin) (`ggplot2::cut_number()`)
- Create a midpoint and move equal distance to each side (perhaps standard deviations), etc.

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
```



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

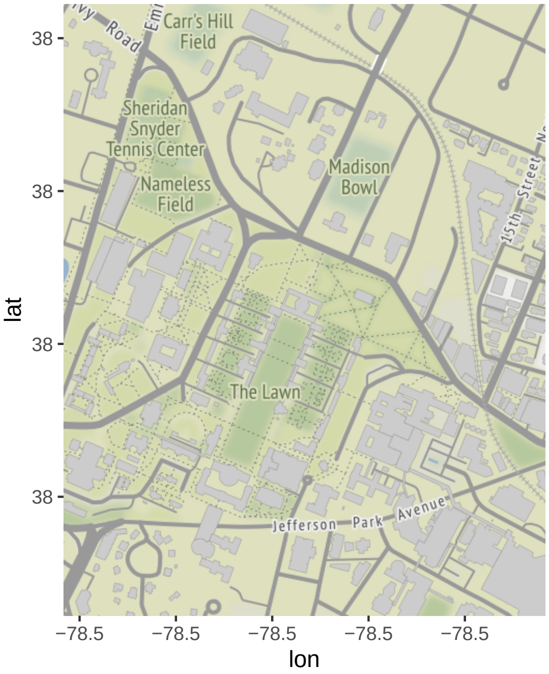
4 ggmap package

Google has new terms that require registration before using any geocoding or google maps. This is why we need to define the bounding box and use `source='stamen'`

The `ggmap` packages allows `ggplot2` to work on top of a map from: google maps, open street maps, statem maps, 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.

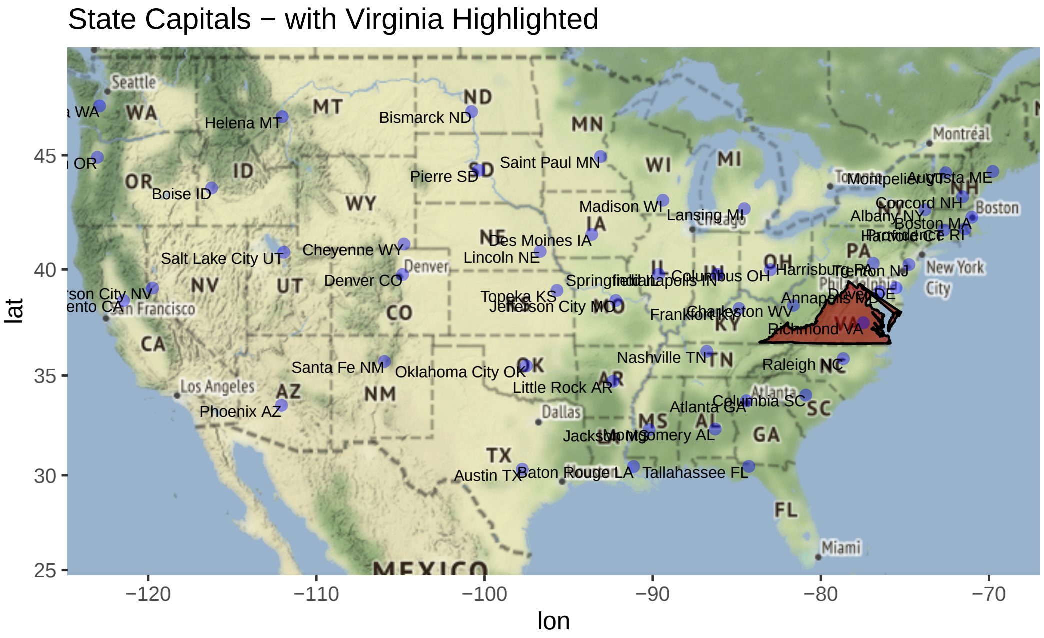
```
library(ggmap)   # install.packages("ggmap")

#-- Get University of Virginia Map (from stamen maps)
UVA = get_map(c(left = -78.503333-.005, bottom=38.035556-.005,
                right =-78.503333+.005,
                top=38.035556+.005),
             zoom = 16,
             source='stamen', maptype='terrain')

#-- Plot it
ggmap(UVA)
```



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

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

#-- Get USA map
```

```
bbox_USA = c(left = -124.7844079,
             bottom =  24.7433195,
             right = -66.9513812,
             top = 49.3457868)
USA = get_map(bbox_USA, source='stamen', maptype='terrain-background')
#USA = get_map("united states", zoom=4)


#-- Now for plots. Notice we need to add the data to the geoms
ggmap(USA) +
  geom_polygon(data=VA, 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 Virginia 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

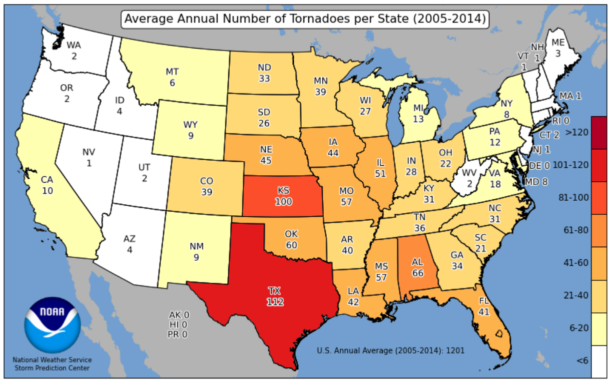
> "Everything is related to everything else, but near things are more related than distant things."
>
> Tobler W., (1970) "A computer movie simulating urban growth in the Detroit region". Economic Geography, 46(2): 234-240.

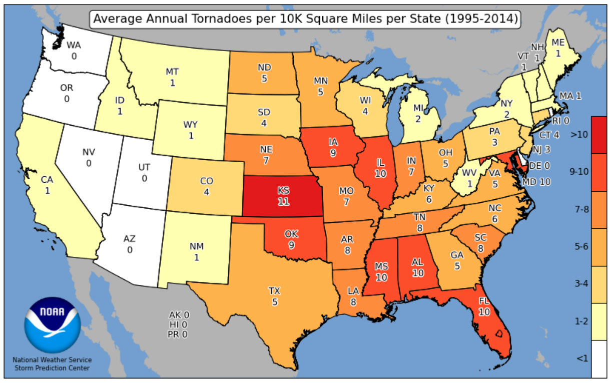
So, if you try to model spatial data using regular regression methods do not be surprised to find spatially correlated errors. There is an entire field of spatial statistics and geostatistics to better incorporate this phenomenon.

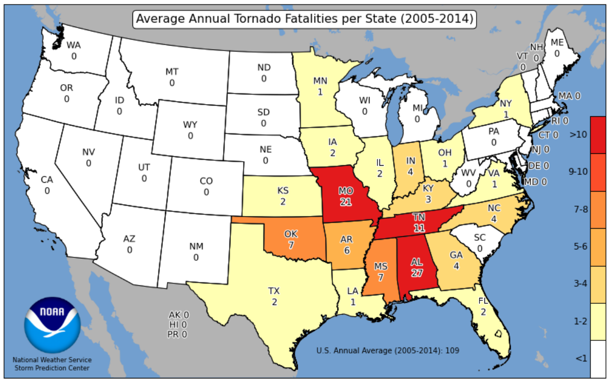
6 Your Turn: Tornado Analytics

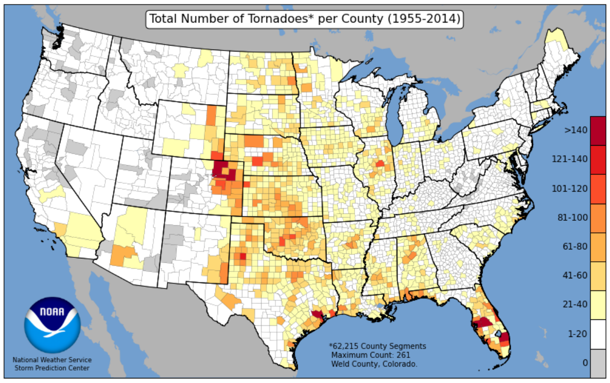
Your Turn #1 : Info

What information is needed to make the following 4 maps?









6.1 Tornado Data

The US Storm Prediction Center make severe weather data available from the website http://www.spc.noaa.gov/wcm/#data. This data is used by insurance companies to help with their claims evaluation and forecasting. A description of the data can be found http://www.spc.noaa.gov/wcm/data/SPC_severe_database_description.pdf.

Import the tornado data from http://www.spc.noaa.gov/wcm/data/Actual_tornadoes.csv. This is not completely accurate as it only uses the first state in a multi-state tornado, but is suitable for our purposes.

Use the column names :

```
cnames = c("om","yr","mo","dy","date","time","tz","st","stf","stn","f","inj",
  "fat","loss","closs","slat","slon","elat","elon","len","wid",
  "ns","sn","sg", "f1","f2","f3","f4", "fc")
```

Your Turn #2 : Import the tornado data

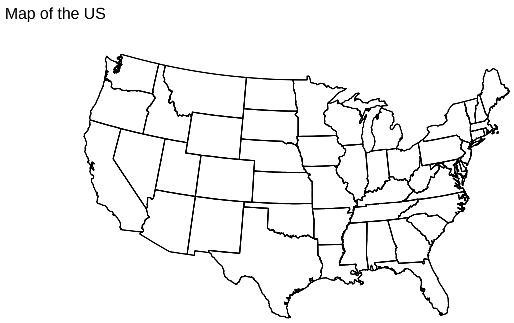
Import the tornado data

```
torn.url = "http://www.spc.noaa.gov/wcm/data/Actual_tornadoes.csv"
```

6.2 Map Data

6.2.1 Make the Base Map

```
#-- Get state boundaries
library(ggplot2)
library(maps)
state_map = map_data("state")                    # get state boundaries

#-- Plot US state map
map_proj = coord_map(projection = "albers", lat0=29.5, lat1=45.4)
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())

USmap = ggplot(state_map,aes(x=long,y=lat,group=group)) + geom_path() +
        map_theme + map_proj

USmap + ggtitle("Map of the US")
```



6.2.2 Get the State Info

```r
library(datasets)
# R has a built-in state data. See ?state.abb for description

state_info =
  tibble(abb=state.abb,
              name=state.name,
              region=state.region,
              x=state.center$x,
              y=state.center$y,
              area=state.x77[,"Area"])  # note: state.x77 is a matrix
```

```r
# we need to be able to join the state_info with state_map. What is key?
# state_map$region (all lowercase) joined with state_info$name (uppercase)
#  so we can convert the names with the stringr str_to_lower() function
library(stringr)
state_info = mutate(state_info, name = str_to_lower(name))
```

6.3 Map 1: average number of tornadoes 2005-2014

Your Turn #3 : Make Map 1: Avg Annual Number of Tornadoes per State (2005-2014)

1. Get all the required data
2. Combine it
3. Make the map
4. Tweak the map

6.4 Map 2: avg annual count per 10K sq miles 1995-2014

Your Turn #4 : Make Map 2: Avg Annual Tornadoes per 10K miles per State (2005-2014)

1. Get all the required data
2. Combine it
3. Make the map
4. Tweak the map

6.5 Map 3: average fatalaties 2005-2014

Your Turn #5 : Make Map 3: Avg Annual Tornado Fatalities per State (2005-2014)

1. Get all the required data
2. Combine it
3. Make the map
4. Tweak the map

6.6 Map 4: Total number of tornadoes per county 1955-2014

Your Turn #6 : Make Map 4: Total Number of Tornadoes per county (2005-2014)

1. Get all the required data
2. Combine it
3. Make the map
4. Tweak the map