Mapping In R

Jiwon Kim

December 3, 2017

Introduction

There are many different ways to create maps in R. In this post, we will explore the different ways to create maps that create visual arguments. The different ways to map include ggmap, shapefiles and the leaflet library. Data mapped with a map can make for a strong argument that ultimately affects the way we look at the world. The structure of this will be as follows. We will explore using the library known as leaflet, followed by some exploration of using ggplot and ggmap to create visual arguments through the use of example.

Leaflet

The first way we will explore data in maps in R is the library known as leaflet, according to this documentation Leaflet in R. This section will be structured as follows: 1. Basic Leaflet set-up 2. Placing Markers on Maps 3. Putting Shapes on Maps

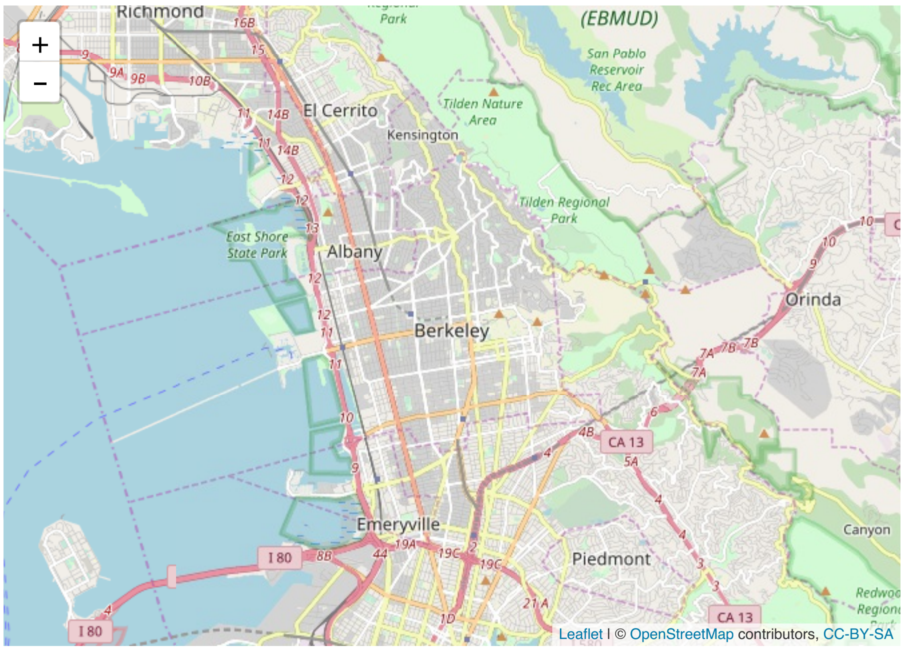
Set Up

For example, we first run `install.packages("leaflet")` to get the packages that is required to run all of our examples. The first way we can use leaflet is by creating a view that shows us some sort of "google-maps" sort of view of the world. By simplying providing the latitude and longitude in the view of leaflet argument, we can show different portions of the global map. Unfortunately leaflet does not have an easy way to add titles to a map, so this will purposely be ommitted (we will come back to this when we use ggplot instead).

```r
library(devtools)
devtools::install_github('rstudio/leaflet')
```

```
## Skipping install of 'leaflet' from a github remote, the SHA1 (d489e2cd) has not changed since last install.
##   Use `force = TRUE` to force installation
```

```r
library(leaflet)
# set latitude and longitude of the view
map <- leaflet() %>% setView(lng = -122.2727, lat = 37.8716 , zoom = 12)
map %>% addTiles()
```

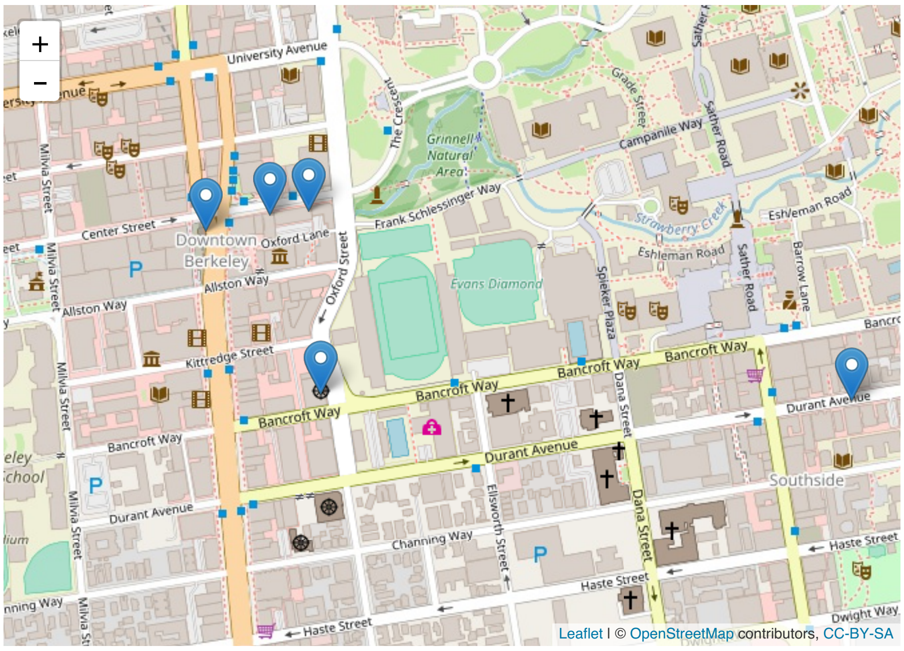


This allows the user to interact with a map, just as they do with a regular Google interactive map. The map above is a simple map of Berkeley; however the power of leaflet is the ability to add different points of interest in the map. This brings us to the second section: Markers.

Markers

Unlike the previous map, the following map shows different places in Berkeley of importance, including the Downtown Berkeley Station, the Two Top Dog locations, Sliver, and Ucha. This allows for another level of interaction with the user. To reiterate leaflet does not have an easy way to add titles to a map, so this will purposely be ommitted .

```
# Set the View and add each Marker independently
restaurants = leaflet() %>% setView(lat = 37.8689112, lng = -122.2641155 , zoom = 16)
restaurants = restaurants %>% addTiles()
restaurants = restaurants %>% addMarkers(lat = 37.8701, lng = - 122.2681, popup = "Downtown Berkeley")
restaurants = restaurants %>% addMarkers(lat = 37.8703507, lng = -122.2664279, popup = "Top Dog #1")
restaurants = restaurants %>% addMarkers(lat = 37.8679396, lng = -122.2576998, popup = "Top Dog #2")
restaurants = restaurants %>% addMarkers(lat = 37.8702951, lng =  -122.2670526, popup = "Sliver Pizzeria")
restaurants = restaurants %>% addMarkers(lat = 37.8680316, lng = -122.2662387, popup = "Ucha")
restaurants
```



The problem with creating graphs in this manner is that you have to manually input the latitude, longitude and the popup for every point of interest, which is simply impossible to do with bigger datasets. So instead, we download a dataset which include latitudes and longitudes in some sort of csv file, iterate through the resulting dataframe and add each restaurant onto the map. This allows us to create a network. The file that I downloaded can be downloaded here. This link should make the following result reproducible.

To accomplish this task, we will need some libraries. To get these libraries run the command `install.packages(c("readr", "dplyr"))`. Once these are installed and the csv is downloaded we are ready to analyze the data. We will create a function just in case we need to analyze from different datasets

```
options(warn = -1)
library(readr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
csvToMap = function(filename, latind, longind, n = 10) {
  # read csv file
  restaurants = read_csv(filename)
  map = leaflet()
  map = addTiles(map)
  # add each marker
  for(row in 1: n) {
    map = addMarkers(map, lat = as.numeric(restaurants[row,latind]),
                     lng = as.numeric(restaurants[row, longind]))
  }
  return(map)
}
```

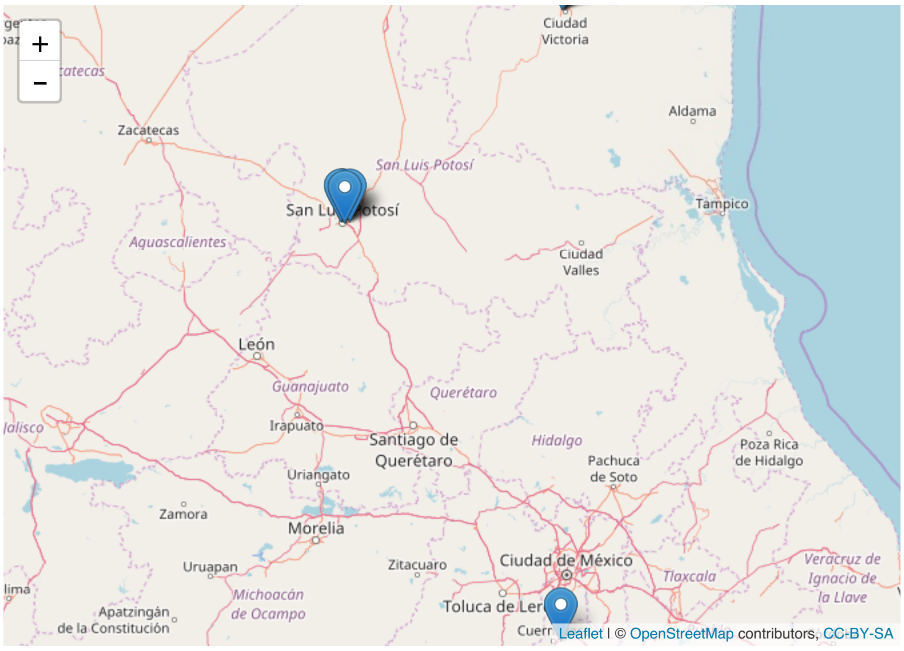
Restaurants

To reiterate leaflet does not have an easy way to add titles to a map, so this will purposely be ommitted .

```
csvToMap("geoplaces2.csv", 2, 3)
```

```
## Parsed with column specification:
## cols(
##   .default = col_character(),
##   placeID = col_integer(),
##   latitude = col_double(),
##   longitude = col_double()
## )
```

```
## See spec(...) for full column specifications.
```



Similarly we can analyze the latitude and longitudes of various earthquakes from the past 50 years. To show this on a map, we use the dataset from kaggle.

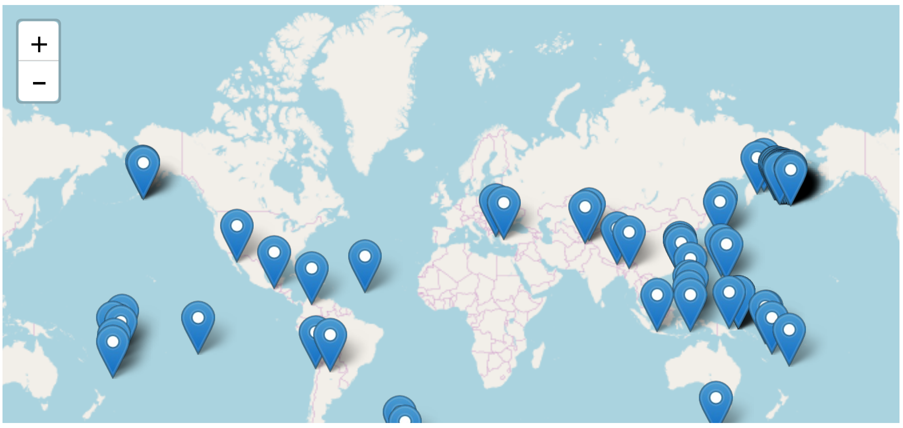
Earthquakes

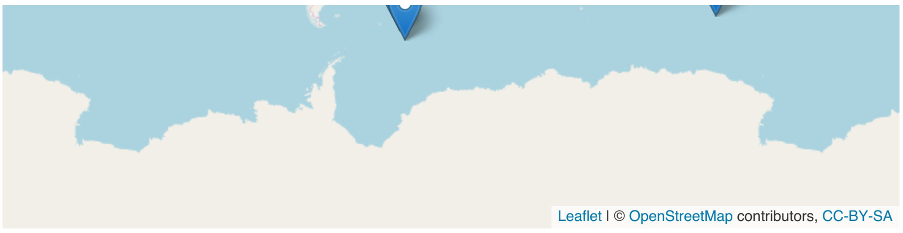
To reiterate leaflet does not have an easy way to add titles to a map, so this will purposely be ommitted .

```
options(warn = -1)
csvToMap("database.csv", 3, 4, 90)
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Date = col_character(),
##   Time = col_time(format = ""),
##   Type = col_character(),
##   `Depth Seismic Stations` = col_integer(),
##   `Magnitude Type` = col_character(),
##   `Magnitude Seismic Stations` = col_integer(),
##   `Azimuthal Gap` = col_integer(),
##   ID = col_character(),
##   Source = col_character(),
##   `Location Source` = col_character(),
##   `Magnitude Source` = col_character(),
##   Status = col_character()
## )
```

```
## See spec(...) for full column specifications.
```





Shapes

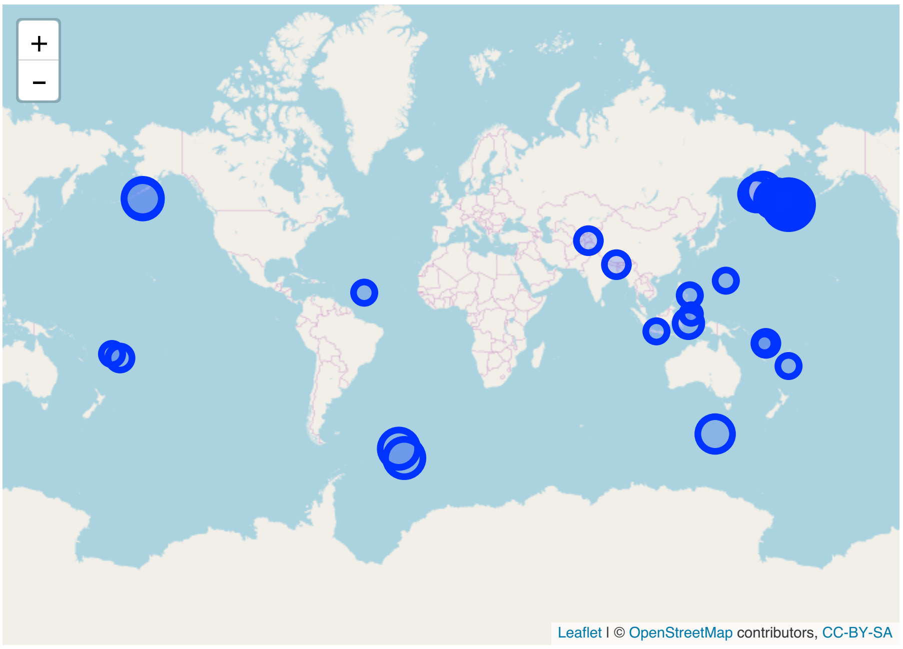
Leaflet also allows us to draw shapes inside the map. For example, we can add circles around the epicenter of earthquakes and make the radius proportional to the magnitude of the earthquake to get a better idea of the destruction caused by the earthquake. To reiterate leaflet does not have an easy way to add titles to a map, so this will purposely be ommitted .

```
earthquake = read_csv("database.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Date = col_character(),
##   Time = col_time(format = ""),
##   Type = col_character(),
##   `Depth Seismic Stations` = col_integer(),
##   `Magnitude Type` = col_character(),
##   `Magnitude Seismic Stations` = col_integer(),
##   `Azimuthal Gap` = col_integer(),
##   ID = col_character(),
##   Source = col_character(),
##   `Location Source` = col_character(),
##   `Magnitude Source` = col_character(),
##   Status = col_character()
## )
```

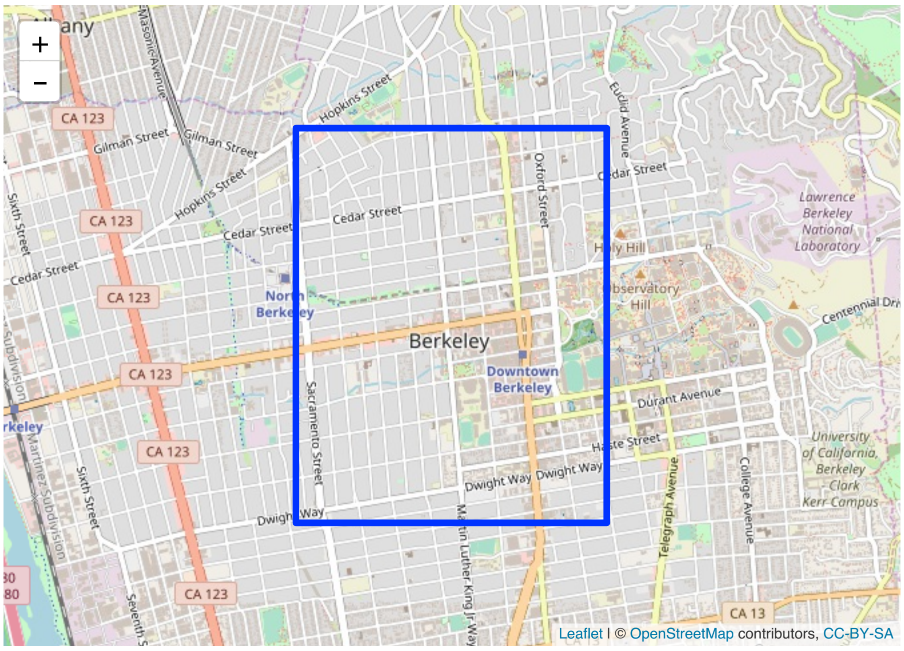
```
## See spec(...) for full column specifications.
```

```
map_earthquake = leaflet()
map_earthquake = addTiles(map_earthquake)
for (ind in 1:50) {
  # add 50 circles where size of each circle is dependent on its magnitude
  map_earthquake = addCircles(map_earthquake, lat = as.numeric(earthquake[ind, 3]),
               lng = as.numeric(earthquake[ind, 4]),
               radius = 100000 * as.numeric(earthquake[ind, 9]))
}
map_earthquake
```



Furthermore, we are not limited to only drawing circles. Leaflet allows a wide variety of different shapes including rectangles and even polygons. We will do a rectangle example next.

```
map <- leaflet() %>% setView(lng = -122.2727, lat = 37.8716 , zoom = 14)
map %>% addTiles() %>% addRectangles(
    lng1 =-122.2727 - 0.01 ,lat1 = 37.8716 - 0.01,
    lng2 = -122.2727 + 0.01,lat2 = 37.8716 + 0.01,
    fillColor = "transparent"
  )
```



GGMap

Similarly we can add data to maps using the ggmap instead of the leaflet library. This section will be structured as follows: 1. Basic Set-up 2. Simple US maps 3. Visual Arguments

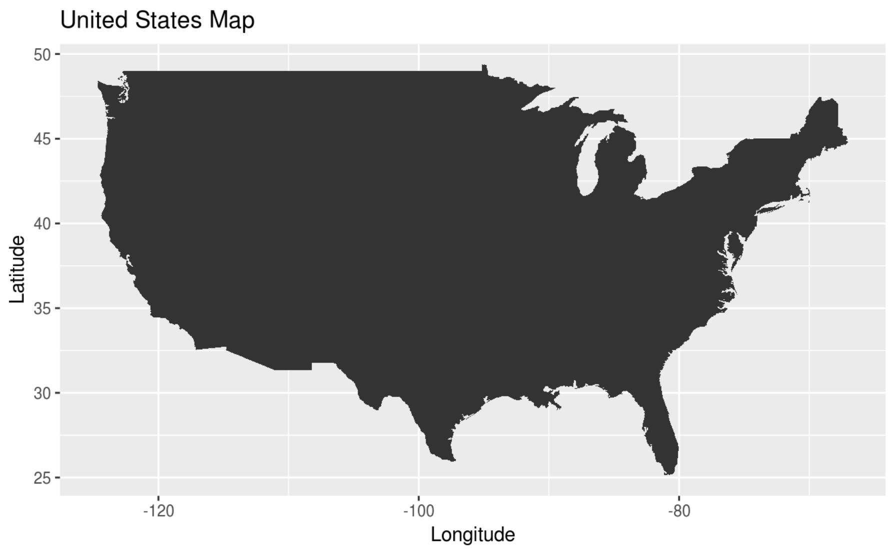
Basic Set-up

To use ggmap, we will firstly need auxilary libraries. Run the command `install.packages(c("ggplot2", "devtools", "dplyr", "stringr", "maps", "mapdata"))`. For more information, check out the following github link. We will also follow this Making Maps with R document Firstly, we will look at the usa map. Notice that we will use the `coord_fixed` command, which tells ggplot to maintain a 1.3 aspect ratio.

```
library(stringr)
library(maps)
library(mapdata)
library(ggplot2)
usa = map_data("usa")
dim(usa)
```

```
## [1] 7243    6
```

```
# create first map ggplot
ggplot() + geom_polygon(data = usa, aes(x = long, y = lat, group = group)) +
  coord_fixed(1.3) + labs(title = "United States Map", x = "Longitude", y = "Latitude")
```

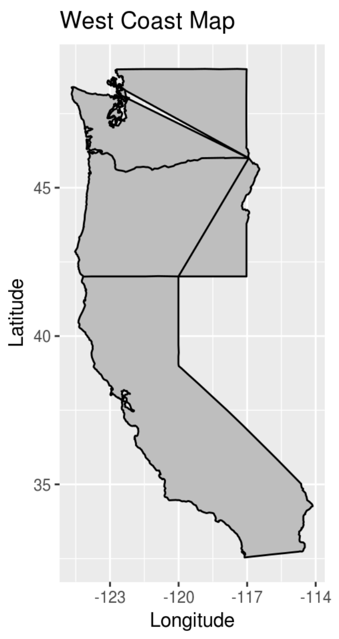


Basic US Maps

Using these libraries we are even able to focus in on specific subsets of the nation like the west coast.

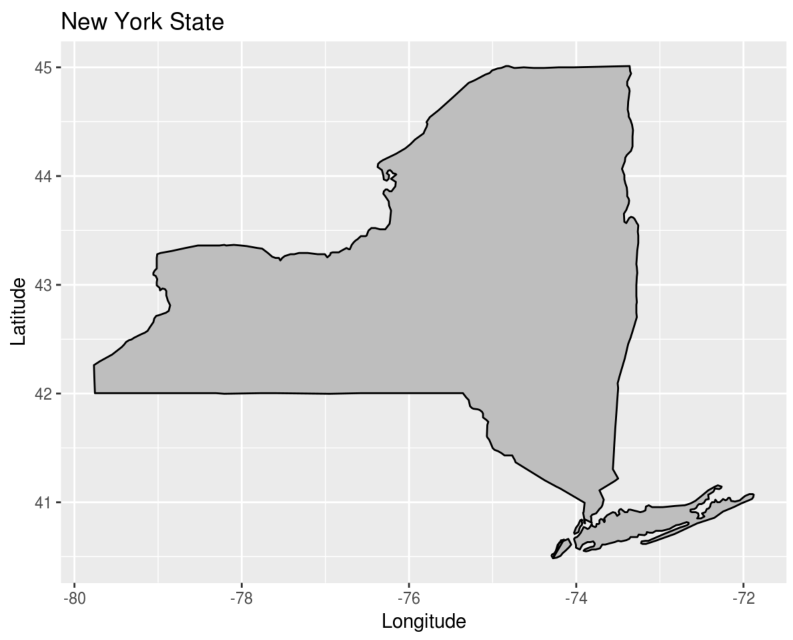
```r
# download state map data
states = map_data("state")
# get west coast
west_coast = subset(states, region %in% c("california", "oregon", "washington"))

# create plot
temp = ggplot(data = west_coast) +
  geom_polygon(aes(x = long, y = lat), color = "black", fill = "gray") + coord_fixed(1.3)
# add labels
temp = temp + labs(title = "West Coast Map", x = "Longitude", y = "Latitude")
temp
```



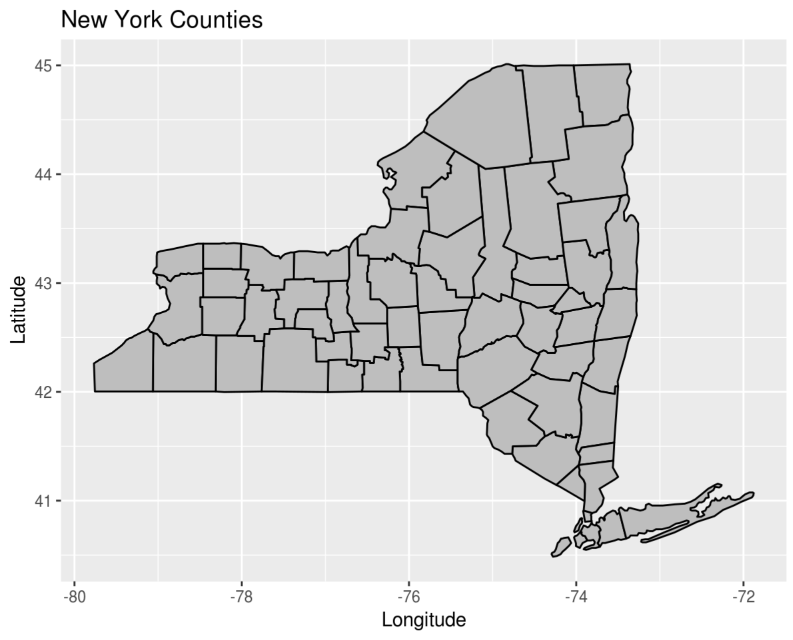
We can also specifically look at one state; for example, we can look at the state of New York.

```r
# select new york
ny_df = subset(states, region == "new york")
# create plot
ny = ggplot(data = ny_df, mapping = aes(x = long, y = lat, group = group)) +
  coord_fixed(1.3) + geom_polygon(color = "black", fill = "gray") +
  labs(title = "New York State", x = "Longitude", y = "Latitude")
ny
```



Furthermore, we can even get a map of the counties in the great state of new york by getting the county map data.

```r
# get county map information
counties = map_data("county")
# select new york
ny_county = subset(counties, region == "new york")
# create ggplot graph
ny_county_graph = ggplot(data = ny_county, mapping = aes(x = long, y = lat, group = group)) +
  coord_fixed(1.3) + geom_polygon(color = "black", fill = "gray")
ny_county_graph + labs(title = "New York Counties", x = "Longitude", y = "Latitude")
```



```r
head(ny_county)
```

```
##            long      lat group order   region subregion
## 52932 -73.78550 42.46763  1795 52932 new york    albany
## 52933 -74.25533 42.41034  1795 52933 new york    albany
## 52934 -74.27252 42.41607  1795 52934 new york    albany
## 52935 -74.24960 42.46763  1795 52935 new york    albany
## 52936 -74.22668 42.50774  1795 52936 new york    albany
## 52937 -74.23241 42.56504  1795 52937 new york    albany
```

Visual Arguments

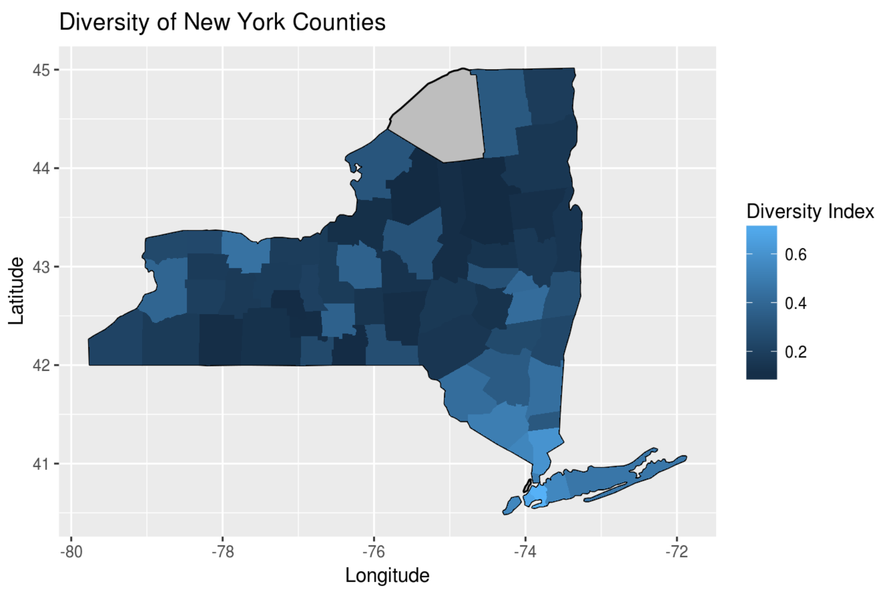
However the power of maps is not to simply show the geographic locations but to use them to show some sort of data, especially to make a point. For example, in the context of an election, it may be of interest to many politicians the ethnic breakdown of New York by county. To analyze something like this, we first need a dataset, which we will get from kaggle, which has the ethnic percentage of various ethnic groups as well as a diversity index. However in order to work with this data effectively, we have to do some pre-processing.

```r
# read diversity csv with appopriate types
div_df = read_csv("diversityindex.csv", col_types = list(
  diversity_index = col_character(),
  black_african_american = col_double(),
  indian_alaska_native = col_double(),
  asian = col_double(),
  haiiwan = col_double(),
  two_plus_races = col_double(),
  hispanic = col_double(),
  white_alone = col_double()
))
# only select those with NY counties
ny_div = filter(div_df, grepl("^.*NY$",Location))
# creates a new column with only the county name
ny_div$subregion = sapply(str_split_fixed(ny_div$Location,", ", 2)[,1], tolower)
# this removes the word county from the column
temp = str_split_fixed(ny_div$subregion, " ", 2)[,1]
# resets the column
ny_div$subregion = temp
ny_div
```

```
## # A tibble: 62 x 10
##                  Location `Diversity-Index`
##                     <chr>             <dbl>
##  1       Queens County, NY          0.742224
##  2        Kings County, NY          0.692349
##  3     New York County, NY          0.658205
##  4 Westchester County, NY          0.605911
##  5       Nassau County, NY          0.551435
##  6     Richmond County, NY          0.547813
##  7     Rockland County, NY          0.542071
##  8       Orange County, NY          0.504601
##  9        Bronx County, NY          0.499601
## 10      Suffolk County, NY          0.468776
## # ... with 52 more rows, and 8 more variables: `Black or African American
## #   alone, percent, 2013` <dbl>, `American Indian and Alaska Native alone,
## #   percent, 2013` <dbl>, `Asian alone, percent, 2013` <dbl>, `Native
## #   Hawaiian and Other Pacific Islander alone, percent,` <dbl>, `Two or
## #   More Races, percent, 2013` <dbl>, `Hispanic or Latino, percent,
## #   2013` <dbl>, `White alone, not Hispanic or Latino, percent,
## #   2013` <dbl>, subregion <chr>
```

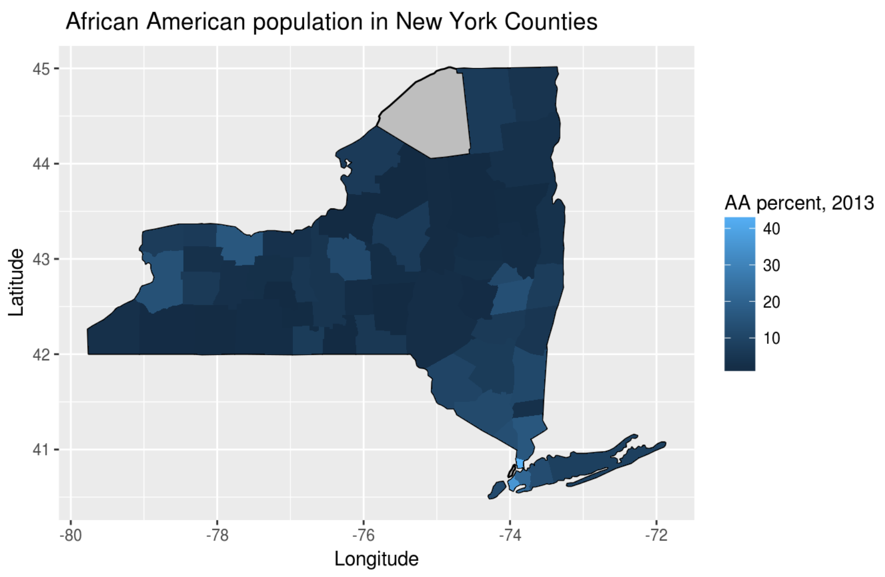
In the previous piece of code, we read in the csv file and then mutated and created a different column such that we can join this data frame and the other dataframe which represents the map, to create one cohesive image. Here, we map what the creater of the data called a diversity index which measures the diversity within a county.

```r
# This joins the two data tables
ny_div_map = inner_join(ny_county, ny_div, by = "subregion")
# creates the combined graph
combined = ny_county_graph +
  geom_polygon(data = ny_div_map, aes(fill = `Diversity-Index`)) +
  labs(fill = "Diversity Index", title = "Diversity of New York Counties",
       x = "Longitude", y = "Latitude")
combined
```



We can also map the proportion of African American populations in each county.

```
# create the african population graph with appropriate fill
african_prop = ny_county_graph +
  geom_polygon(data = ny_div_map, aes(fill =
                                        `Black or African American alone, percent, 2013`)) +
  labs(fill = "AA percent, 2013")
# add appropriate labels
african_prop + labs(title = " African American population in New York Counties",
                    x = "Longitude", y = "Latitude")
```



# Message At the end of the day, we

have learned two separate ways to show information on graphs. Although leaflet is a little harder to learn and the functions aren't that great, its main benefit is the user's ability to play around with the map. Although ggplot's ability to make visual arguments is much better than that of leaflet, the user cannot interact with it as well as they can with leaflet. Both have its pros and cons

References

1. Leaflet in R
2. Making Maps with R
3. Restaurants in Mexico
4. Kaggle Earthquakes
5. Ggmap Github
6. Kaggle Diversity in New York
7. Leaflet Documentation