Chapter 6 Maps and Spatial Data

In this section, we will look at a spatial data. We will learn how to create maps and combine geographic information with other types of data.

6.1 Using Simple Features in R

To get started, we will work with a package called `sf`, which stands for simple (not spatial) features. So *what is a feature*? A feature is an real-world object, such as a building, city, tree, or forest. Features have spatial and non-spatial attributes. The simple features website notes the following:

> "Features have a geometry describing where on Earth the feature is located, and they have attributes, which describe other properties. The geometry of a tree can be the delineation of its crown, of its stem, or the point indicating its centre. Other properties may include its height, color, diameter at breast height at a particular date, and so on."

In other words, most of the data that we have been working with in this class, such as text information, contains *attributes* about specific news articles, social media posts, or words from these data sources. Our dataframes have, for the most part, not included *geographic* information about where each article or post came from (although some of our data included general categories, such as "US" and "World News").

To get started with `sf`, we will install it and library the package. We can then load the shapefile for North Carolina, which is pre-loaded into the `sf` package.

```
library(sf)

# load a simple features object
nc <- st_read(system.file("shape/nc.shp", package="sf"))
```

```
## Reading layer `nc' from data source
##   `C:\Users\tyler\AppData\Local\R\win-library\4.3\sf\shape\nc.shp'
##   using driver `ESRI Shapefile'
## Simple feature collection with 100 features and 14 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -84.32385 ymin: 33.88199 xmax: -75.45698 ymax: 36.58965
## Geodetic CRS:  NAD27
```

You'll note that we get a lot more information than when we load in .csv files to our R environment. R notes that in this simple features collection, there are 100 features and 14 fields. This really just means that there are 100 observations in our data and 14 variables (or 15 including the geometry). We see that these type of data are *MULTIPOLYGON*, meaning that each observation is a different polygon. We also have a *bounding box*, which notes the most extreme longitude and latitude points. All the data is contained geographically in this bounding box. The xmin and xmax values represent latitudes, whereas the ymin and ymax values represent longitudes. Positive values represent East and North, negative values represent West and South. If we look at the `class` of the object, we can see it is a "simple features" object as well as a "data frame."

```
# look at class of object
class(nc)
```

```
## [1] "sf"         "data.frame"
```

This means that while nc is an `sf` object, we can still use `dplyr` -type commands on hte attributes as if it were another dataframe. Let's take a look at what attributes, and then what the *geometry* features are like.

```r
library(dplyr)

# look at attributes (normal variables)
nc %>%
  head()
```

```
## Simple feature collection with 6 features and 14 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -81.74107 ymin: 36.07282 xmax: -75.77316 ymax: 36.58965
## Geodetic CRS:  NAD27
##    AREA PERIMETER CNTY_ CNTY_ID        NAME  FIPS FIPSNO CRESS_ID BIR74 SID74
## 1 0.114     1.442  1825    1825        Ashe 37009  37009        5  1091     1
## 2 0.061     1.231  1827    1827   Alleghany 37005  37005        3   487     0
## 3 0.143     1.630  1828    1828       Surry 37171  37171       86  3188     5
## 4 0.070     2.968  1831    1831   Currituck 37053  37053       27   508     1
## 5 0.153     2.206  1832    1832 Northampton 37131  37131       66  1421     9
## 6 0.097     1.670  1833    1833    Hertford 37091  37091       46  1452     7
##   NWBIR74 BIR79 SID79 NWBIR79                       geometry
## 1      10  1364     0      19 MULTIPOLYGON (((-81.47276 3...
## 2      10   542     3      12 MULTIPOLYGON (((-81.23989 3...
## 3     208  3616     6     260 MULTIPOLYGON (((-80.45634 3...
## 4     123   830     2     145 MULTIPOLYGON (((-76.00897 3...
## 5    1066  1606     3    1197 MULTIPOLYGON (((-77.21767 3...
## 6     954  1838     5    1237 MULTIPOLYGON (((-76.74506 3...
```

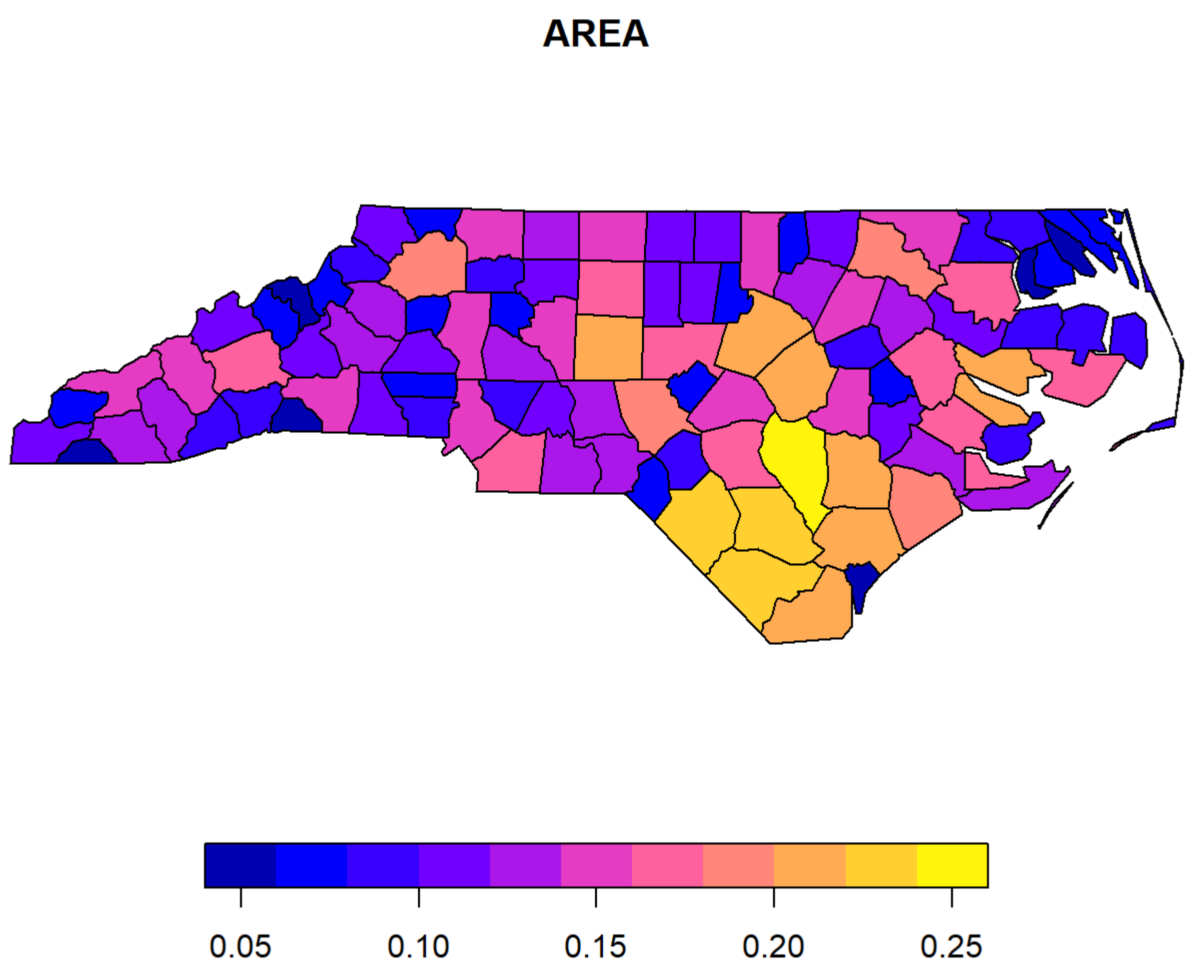
Let’s take a closer look at the geometries!

geometry
list(list(c(-81.4727554321289, -81.5408401489258, -81.5619812011719, -81.6330642700195, -81.7410736083984, -81.6982803344727, -81.7027969360352, -81.6699981689453, -81.3452987670898, -81.347541809082, -81.3247756958008, -81.3133239746094, -81.2662353515625, -81.2628402709961, -81.2406921386719, -81.2398910522461, -81.2642440795898, -81.3289947509766, -81.3613739013672, -81.3656921386719, -81.354133605957, -81.3674545288086, -81.4063873291016, -81.4123306274414, -81.431037902832, -81.4528884887695, -81.4727554321289, 36.2343559265137, 36.2725067138672, 36.2735939025879, 36.3406867980957, 36.3917846679688, 36.4717788696289, 36.5193405151367, 36.5896492004395, 36.5728645324707, 36.537914276123, 36.5136795043945, 36.4806976318359, 36.4372062683105, 36.4050407409668, 36.3794174194336, 36.365364074707, 36.3524131774902, 36.3635025024414, 36.3531608581543, 36.3390502929688, 36.2997169494629, 36.2786979675293, 36.2850532531738, 36.2672920227051, 36.2607192993164, 36.2395858764648, 36.2343559265137)))
list(list(c(-81.2398910522461, -81.2406921386719, -81.2628402709961, -81.2662353515625, -81.3133239746094, -81.3247756958008, -81.347541809082, -81.3452987670898, -80.9034423828125, -80.9335479736328, -80.9657745361328, -80.9496688842773, -80.9563903808594, -80.9779510498047, -80.9828414916992, -81.0027770996094, -81.0246429443359, -81.0428009033203, -81.0842514038086, -81.0985641479492, -81.1133117675781, -81.1293792724609, -81.1383972167969, -81.1533660888672, -81.1766738891602, -81.2398910522461, 36.365364074707, 36.3794174194336, 36.4050407409668, 36.4372062683105, 36.4806976318359, 36.5136795043945, 36.537914276123, 36.5728645324707, 36.5652122497559, 36.4983139038086, 36.4672203063965, 36.4147338867188, 36.4037971496582, 36.3913764953613, 36.3718338012695, 36.3666801452637, 36.3778343200684, 36.4103355407715, 36.4299201965332, 36.43115234375, 36.4228515625, 36.4263305664062, 36.4176254272461, 36.4247398376465, 36.4154434204102, 36.365364074707)))

We notice that each MULTIPOLYGON observation consists of a list within a list. The first list contains polygons, whereas the second list contains the bounding points of these polygons.

We can try plotting these data! Notice that when we specify `nc[1]`, this selects the AREA attribute as well as the geometry.

```
# plot with base R
plot(nc[1], reset = FALSE) # reset = FALSE: we want to add to a plot
```



We can identify the observations (counties, in this case) with multiple polygons by examining which observations have lengths > 1. We'll first do this using base R:

```
# define index for lengths greater than 1
w <- which(sapply(nc$geometry, length) > 1)

# take a look
nc[w,]
```

```
## Simple feature collection with 6 features and 14 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -77.4741 ymin: 34.58783 xmax: -75.45698 ymax: 36.55716
## Geodetic CRS:  NAD27
##     AREA PERIMETER CNTY_ CNTY_ID      NAME  FIPS FIPSNO CRESS_ID BIR74 SID74
## 4  0.070     2.968  1831    1831 Currituck 37053  37053       27   508     1
## 56 0.094     3.640  2000    2000      Dare 37055  37055       28   521     0
## 57 0.203     3.197  2004    2004  Beaufort 37013  37013        7  2692     7
## 87 0.167     2.709  2099    2099      Hyde 37095  37095       48   338     0
## 91 0.177     2.916  2119    2119    Craven 37049  37049       25  5868    13
## 95 0.125     2.868  2156    2156  Carteret 37031  37031       16  2414     5
##    NWBIR74 BIR79 SID79 NWBIR79                       geometry
## 4      123   830     2     145 MULTIPOLYGON (((-76.00897 3...
## 56      43  1059     1      73 MULTIPOLYGON (((-75.78317 3...
## 57    1131  2909     4    1163 MULTIPOLYGON (((-77.10377 3...
## 87     134   427     0     169 MULTIPOLYGON (((-76.51894 3...
## 91    1744  7595    18    2342 MULTIPOLYGON (((-76.89761 3...
## 95     341  3339     4     487 MULTIPOLYGON (((-77.14896 3...
```

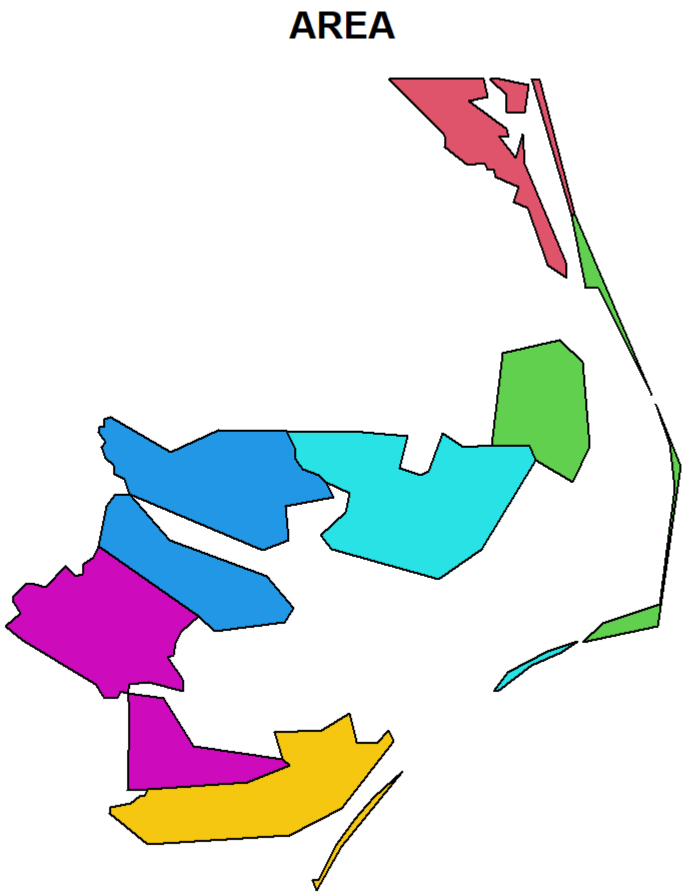
And now we'll try the same thing using `dplyr` methods. We should get the same result, and we do:

```
# get all observations with lengths > 1
nc %>%
  filter(sapply(geometry, length) > 1)
```

```
## Simple feature collection with 6 features and 14 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -77.4741 ymin: 34.58783 xmax: -75.45698 ymax: 36.55716
## Geodetic CRS:  NAD27
##    AREA PERIMETER CNTY_ CNTY_ID      NAME  FIPS FIPSNO CRESS_ID BIR74 SID74
## 1 0.070     2.968  1831    1831 Currituck 37053  37053       27   508     1
## 2 0.094     3.640  2000    2000      Dare 37055  37055       28   521     0
## 3 0.203     3.197  2004    2004  Beaufort 37013  37013        7  2692     7
## 4 0.167     2.709  2099    2099      Hyde 37095  37095       48   338     0
## 5 0.177     2.916  2119    2119    Craven 37049  37049       25  5868    13
## 6 0.125     2.868  2156    2156  Carteret 37031  37031       16  2414     5
##   NWBIR74 BIR79 SID79 NWBIR79                       geometry
## 1     123   830     2     145 MULTIPOLYGON (((-76.00897 3...
## 2      43  1059     1      73 MULTIPOLYGON (((-75.78317 3...
## 3    1131  2909     4    1163 MULTIPOLYGON (((-77.10377 3...
## 4     134   427     0     169 MULTIPOLYGON (((-76.51894 3...
## 5    1744  7595    18    2342 MULTIPOLYGON (((-76.89761 3...
## 6     341  3339     4     487 MULTIPOLYGON (((-77.14896 3...
```

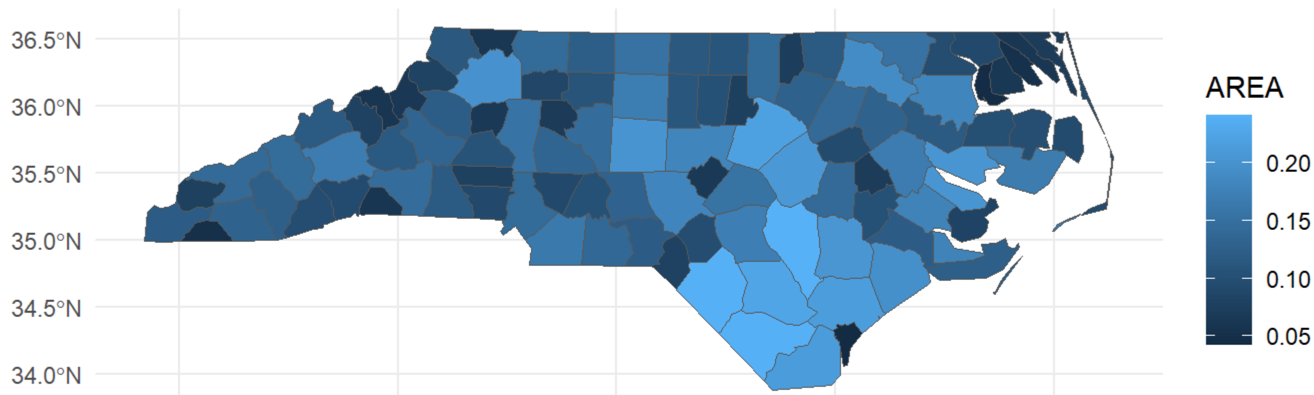
Great! Now that we have identified observations with multiple polygons, let’s plot these.

```
# now plot the results
plot(nc[w,1], col = 2:7)
```



You might have noticed that up until now, we've only plotted these data with the base R `plot()` function. However, like with the other data we've seen, as we become more advanced in our data visualization techniques, we can use `ggplot()` to customize our graphics. Let's try plotting each county by its area again, but with `ggplot`. To do this, we'll use the `geom_sf()` command to add the spatial layer.

```
library(ggplot2)
# plot with ggplot
ggplot(nc)+
  geom_sf(aes(fill = AREA))+
  theme_minimal()
```



6.2 Mapping Points and Polygons

Across the globe, academics, civil servants, and industry leaders are working to understand how cities can be more resilient and more equitable in terms of their environmental impacts and risks. In Measuring Urban Climate Equity, global metrics for climate hazards and actions are quantified by a company called CDP, which surveys governments and other organizations about climate-related topics.

I've the 2020 CDP survey to a few questions of interest for our purposes (but there is a lot more information in these surveys, if you want to look into them yourself!). We can download some basic information about urban hazards and actions with the following code, after downloading the "cdp_hazards.csv" and "cdp_actions.csv" files from Canvas.

```r
library(readr)

# read climate hazards
cdp_hazards <- read_csv("Data/cdp_hazards.csv")

# read climate actions
cdp_actions <- read_csv("Data/cdp_actions.csv")
```

Let’s take a look at our data! Below are the climate hazards that the 566 cities reported. Note that the variable “hazard” shows the type of hazard relevant to each city.

Questionnaire	Year Reported to CDP	Account Number	Organization	Country	CDP Region

And now, let's look at the climate actions that these cities report. The variable "action" here shows the category of climate actions taken.

Questionnaire	Year Reported to CDP	Account Number	Organization	Country	CDP Region
Cities 2020	2020	54538	Bath and North East Somerset	United Kingdom of Great Britain and Northern Ireland	Europe

Great! We'll want to plot the different hazards and actions that cities are taking around the globe. In order to do this, we will need geographic data on the locations of the reporting cities. CDP includes these in a file called "CDP-Cities-geographical-coordinates.csv," which I've added to Canvas.

```
library(magrittr)

# load location data
cdp_geo <- read_csv("Data/CDP-Cities-geographical-coordinates.csv")

# only need account number and lat/long
cdp_geo %<>%
   rename(`Account Number` = Account.Number) %>%
  select(`Account Number`, lat, long)
```

We'll *join* the geometries with our previous datasets on cities' hazards and actions using the `left_join()` function. Since each data has all the same cities, we could also do a `full_join()` and achieve the same result.

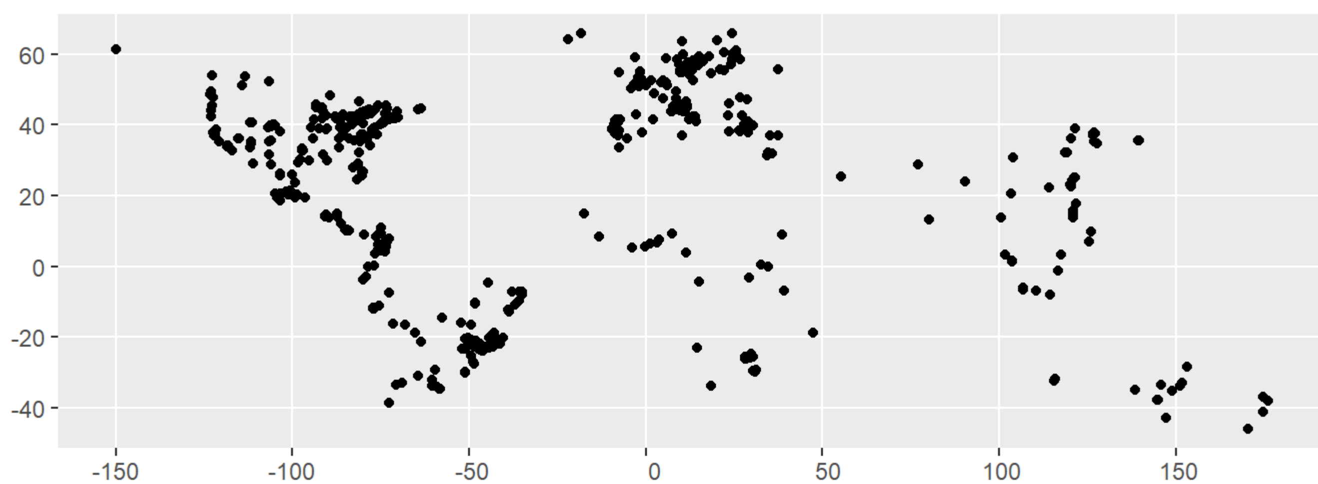
```
# join hazards with geom
cdp_hazards %<>%
  left_join(cdp_geo)

# join actions with geom
cdp_actions %<>%
  left_join(cdp_geo)
```

Great! Let's try plotting some of the hazards. For example, we might want to look at heat waves. First we'll convert our data to "simple features" format:

```
# convert hazards to simple features
cdp_hazards %<>%
  st_as_sf(coords = c("long", "lat"))
```

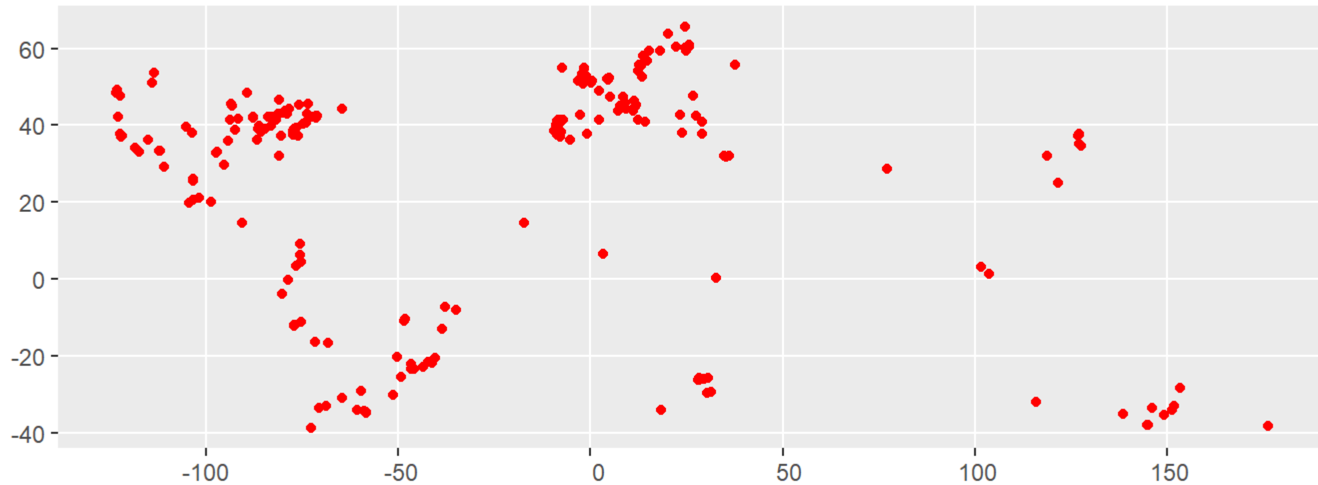
Now let's try plotting our cities!

```
ggplot(cdp_hazards)+
  geom_sf()
```



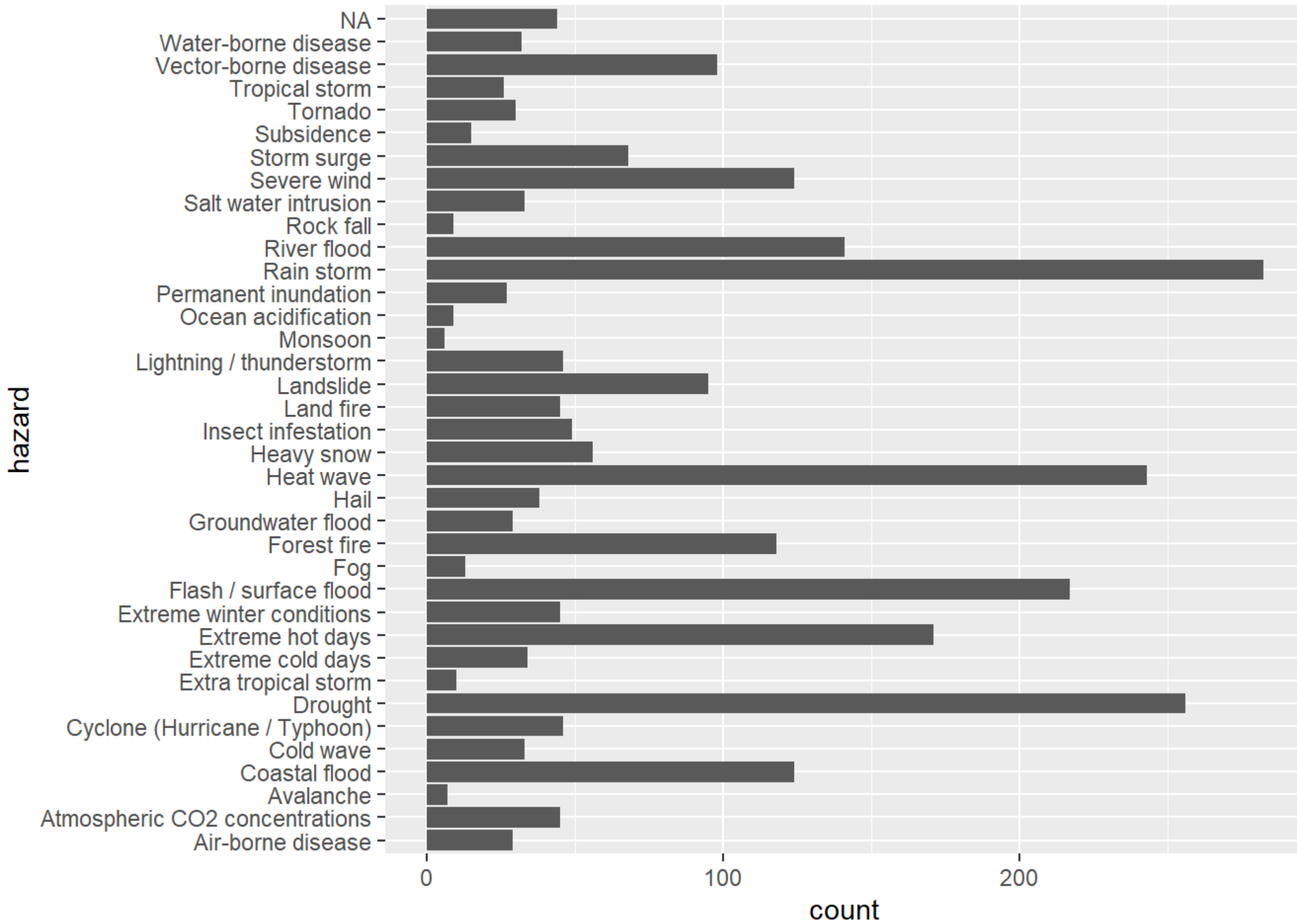
And now, let's just plot the cities with heat waves.

```
# plot heat wave cities
ggplot(cdp_hazards %>% filter(hazard == "Heat wave"))+
  geom_sf(col = "red")
```



Hmmm, what if we wanted to show different types of hazards in different colors? Let's first take a look at all the different types of hazards in our data.

```
# plot hazard types
ggplot(cdp_hazards, aes(hazard))+
  geom_bar()+
  coord_flip()
```

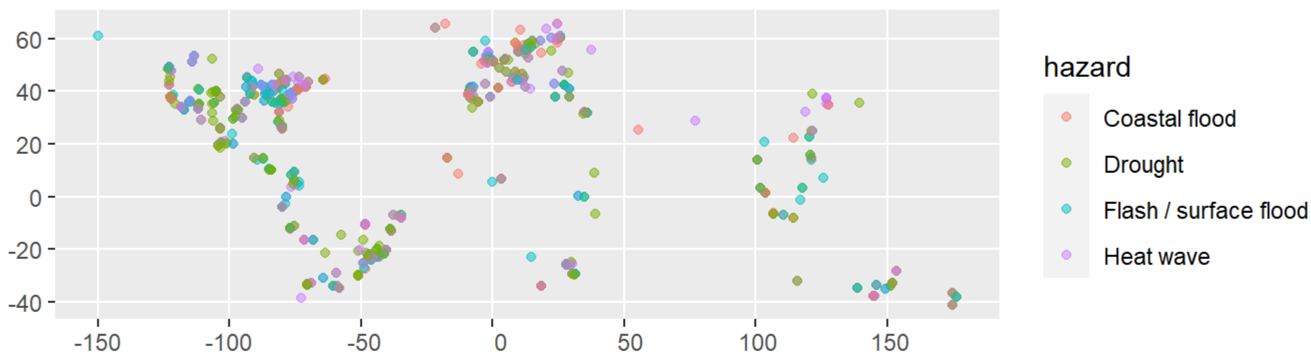


For now, let's just consider the most common types of hazards. We can limit our data with the `%in%` function.

```
# restrict data to only certain types of hazards
cdp_hazards %<>%
  filter(hazard %in% c("Heat wave",
                       "Flash / surface flood",
                       "Drought",
                       "Coastal flood"))
```

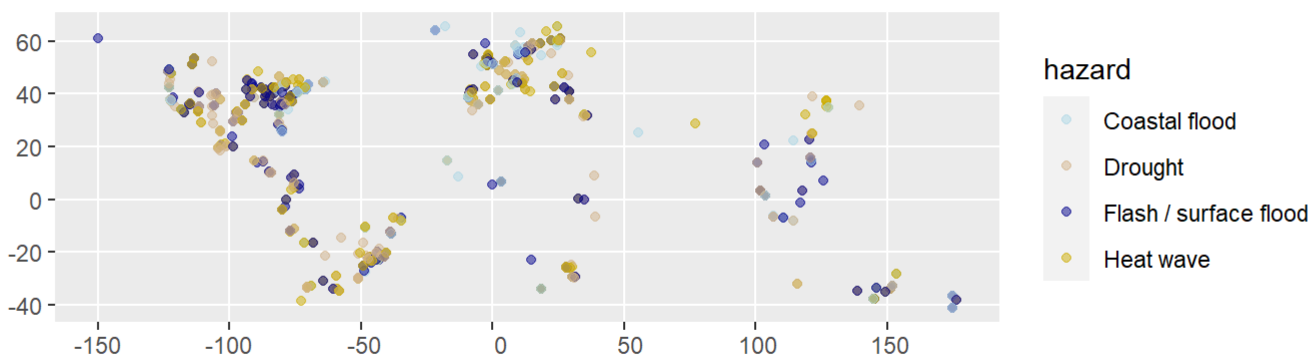
We can do this using the `aes()` function within `geom_sf()`. We'll also set `alpha` to 0.5, outside of the `aes()` function since we want this to apply to all our data points equally, in order to make the points a bit more transparent (`alpha` ranges from 0 to 1).

```
# plot different hazards
ggplot(cdp_hazards)+
  geom_sf(aes(col = hazard), alpha = 0.5)
```



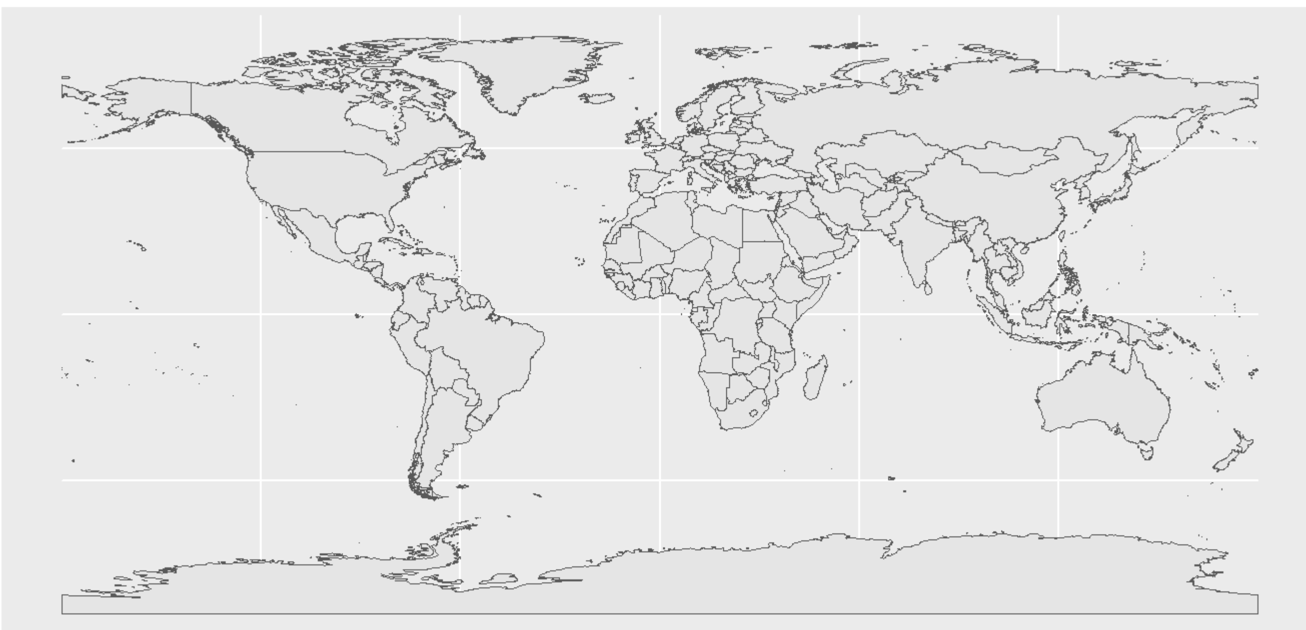
Nice! We have a map that shows the prevalence of the different hazards in major cities. We might want to modify the colors so that they are more intuitive. We can do this with the `scale_color_manual()` argument. Within this argument, we can specify the colors that we want, in the order of the variable's values.

```
# plot different hazards
ggplot(cdp_hazards)+
  geom_sf(aes(col = hazard), alpha = 0.5)+
  scale_color_manual(values = c("lightblue",
                                "tan",
                                "darkblue",
                                "gold3"))
```



There is an obvious omission from our map up to this point - there are no boundaries! It's a bit hard to tell where these cities actually are.

```
library(rnaturalearth)
library(rnaturalearthdata)

world <- ne_countries(scale = "medium", returnclass = "sf")

## let's try plotting the world
ggplot(world)+
  geom_sf()
```



We can also take a look at the world data. Specifically, let's take a look at the *Coordinate Reference System*, or CRS.

```
# take a look at world data
st_crs(world)
```

```
## Coordinate Reference System:
##   User input: +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
##   wkt:
## BOUNDCRS[
##     SOURCECRS[
##         GEOGCRS["unknown",
##             DATUM["World Geodetic System 1984",
##                 ELLIPSOID["WGS 84",6378137,298.257223563,
##                     LENGTHUNIT["metre",1]],
##                 ID["EPSG",6326]],
##             PRIMEM["Greenwich",0,
##                 ANGLEUNIT["degree",0.0174532925199433],
##                 ID["EPSG",8901]],
##             CS[ellipsoidal,2],
##                 AXIS["longitude",east,
##                     ORDER[1],
##                     ANGLEUNIT["degree",0.0174532925199433,
##                         ID["EPSG",9122]]],
##                 AXIS["latitude",north,
##                     ORDER[2],
##                     ANGLEUNIT["degree",0.0174532925199433,
##                         ID["EPSG",9122]]]]],
##     TARGETCRS[
##         GEOGCRS["WGS 84",
##             DATUM["World Geodetic System 1984",
##                 ELLIPSOID["WGS 84",6378137,298.257223563,
##                     LENGTHUNIT["metre",1]]],
##             PRIMEM["Greenwich",0,
##                 ANGLEUNIT["degree",0.0174532925199433]],
##             CS[ellipsoidal,2],
##                 AXIS["latitude",north,
##                     ORDER[1],
##                     ANGLEUNIT["degree",0.0174532925199433]],
##                 AXIS["longitude",east,
##                     ORDER[2],
```

```
##                     ANGLEUNIT["degree",0.0174532925199433]],
##             ID["EPSG",4326]]],
##     ABRIDGEDTRANSFORMATION["Transformation from unknown to WGS84",
##         METHOD["Geocentric translations (geog2D domain)",
##             ID["EPSG",9603]],
##         PARAMETER["X-axis translation",0,
##             ID["EPSG",8605]],
##         PARAMETER["Y-axis translation",0,
##             ID["EPSG",8606]],
##         PARAMETER["Z-axis translation",0,
##             ID["EPSG",8607]]]]
```

When we look at our city data, we notice that the CRS is missing:

```
# check crs of cdp data
st_crs(cdp_hazards)
```

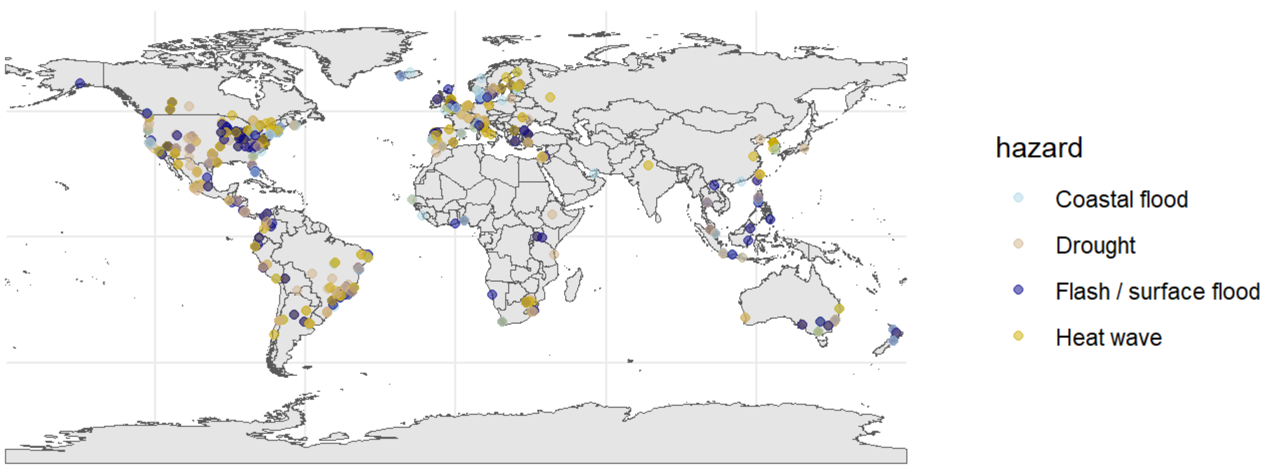
```
## Coordinate Reference System: NA
```

In order to combine these data in the same graph, we'll need them to have matching CRSs. We can change the CRS of `cdp_hazards` using the following:

```
# transform cdp_hazards to have the same reference system as world
cdp_hazards %<>%
  st_set_crs(st_crs(world))
```

Great! Now let's add our hazards on top of the world map. We'll add `theme_minimal()` in order to differentiate the land from the background.

```
## let's try plotting the world
ggplot(cdp_hazards)+
  geom_sf(data = world)+
  geom_sf(aes(col = hazard), alpha = 0.5)+
  scale_color_manual(values = c("lightblue",
                                "tan",
                                "darkblue",
                                "gold3"))+
  theme_minimal()
```

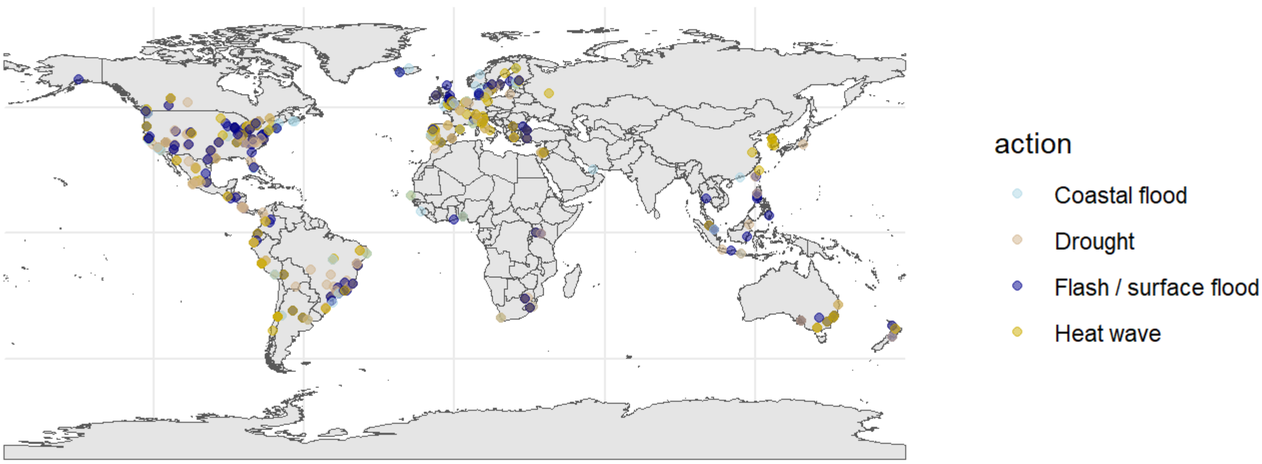


We can similarly look at the types of actions that cities are taking on different climate hazards. We'll clean up our actions data similarly to how we did for the hazards data.

```
# convert actions to simple features
cdp_actions %<>%
  st_as_sf(coords = c("long", "lat"))

# restrict data to only certain types of hazards
cdp_actions %<>%
  filter(action %in% c("Heat wave",
                       "Flash / surface flood",
                       "Drought",
                       "Coastal flood"))

# transform cdp_actions to have the same reference system as world
cdp_actions %<>%
  st_set_crs(st_crs(world))
```

Now we can plot the actions! How does this compare with the hazards plot from above?

```
## let's try plotting the world
ggplot(cdp_actions)+
  geom_sf(data = world)+
  geom_sf(aes(col = action), alpha = 0.5)+
  scale_color_manual(values = c("lightblue",
                                "tan",
                                "darkblue",
                                "gold3"))+
  theme_minimal()
```



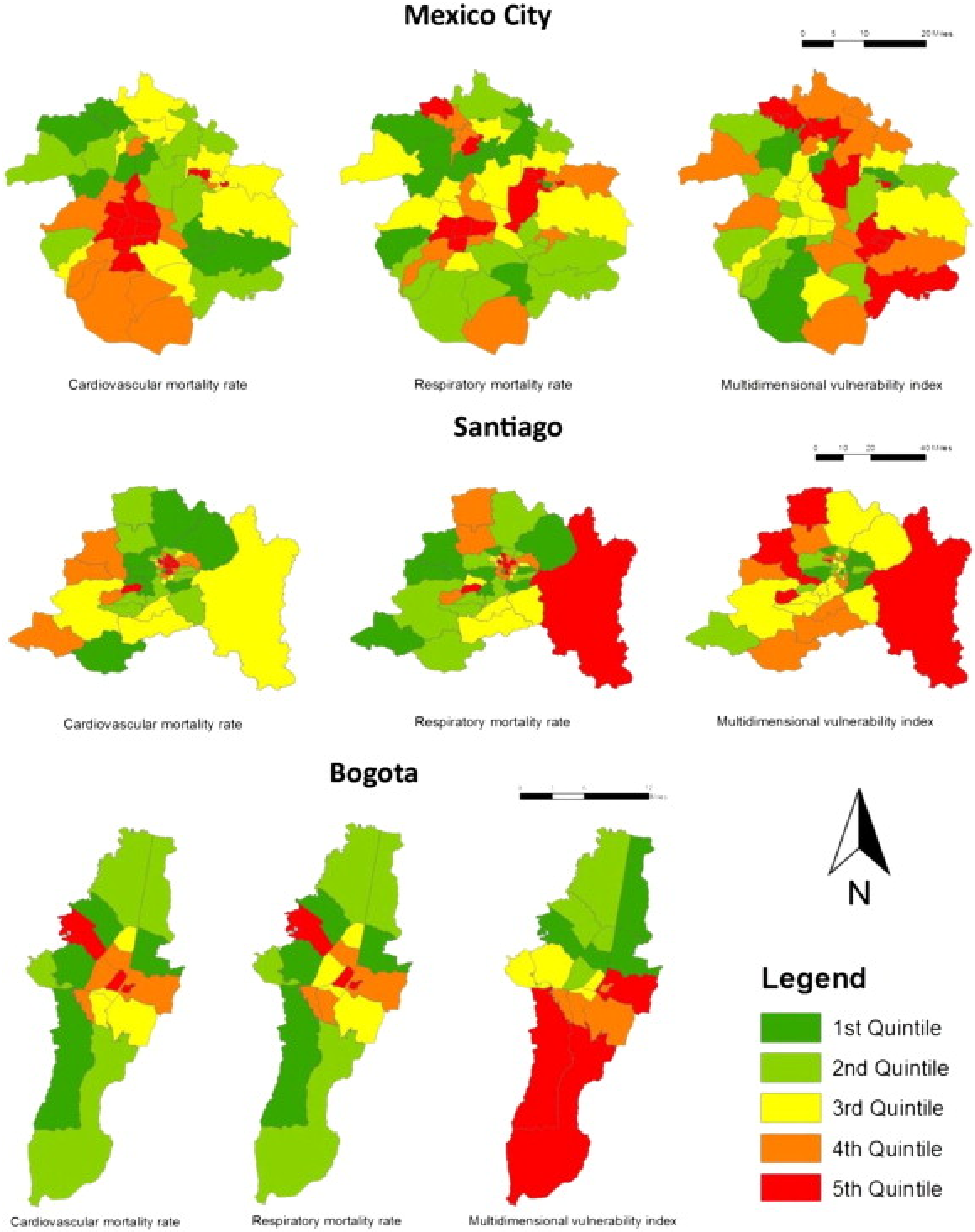
6.3 Air Quality Across Space

In the following sections, we will examine how spatial unevenness in air quality. Pollution is a major contributor to poor health and morbidity: Schwartz notes that in the U.S., "particulates are estimated to account for over 100,000 deaths annually, more than breast cancer, prostate cancer, and AIDS combined." Air pollution from particulates (e.g. particulate matter 2.5 or PM 2.5) reduces life spans by about two years on average. Globally, 97% of the population lives in regions with "unsafe" levels of particulate pollution (according to guidelines from the World Health Organization).

Considering unevenness in air quality across space, we might return to two concepts that we've introduced earlier in this class: risks and environmental justice. These two broad sociological approaches may be useful in connecting air quality to existing social structures and policy actions.

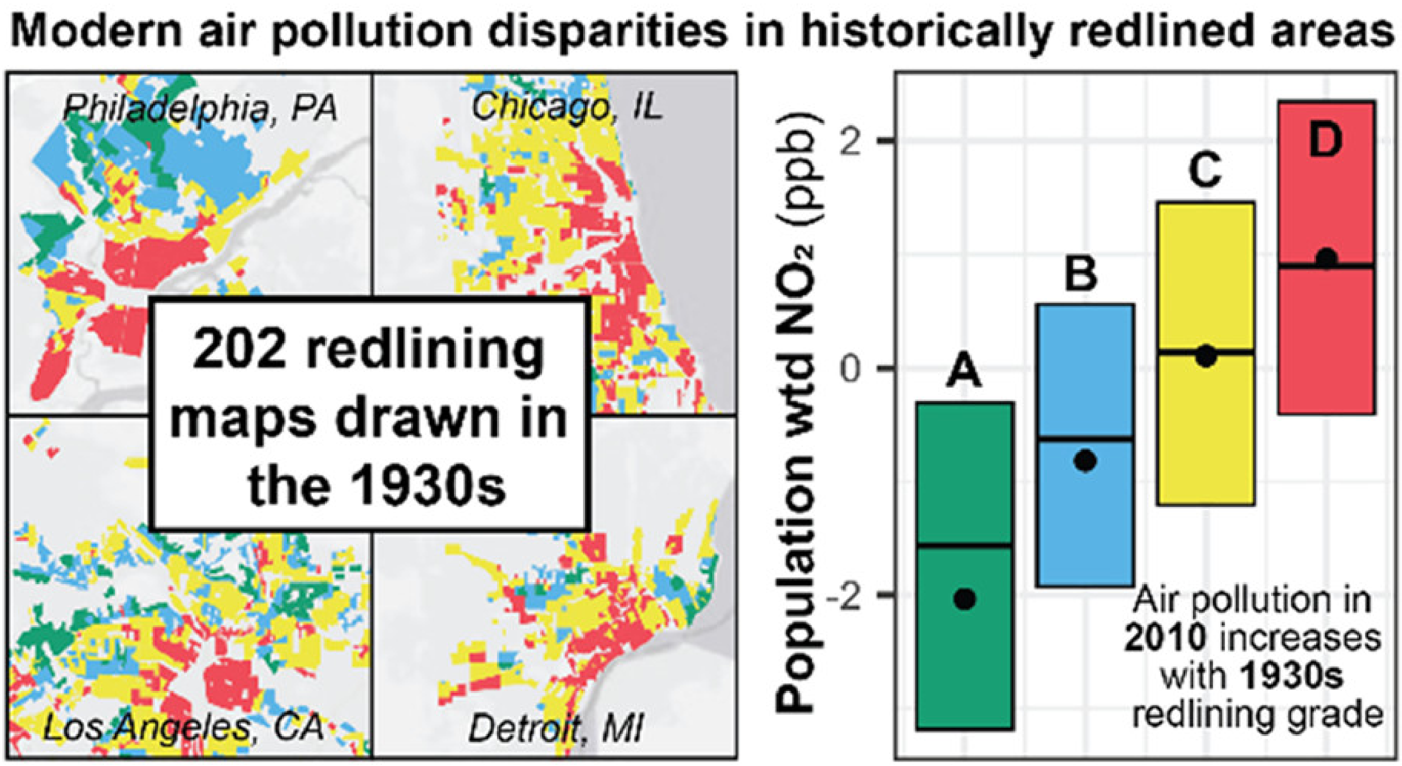
First, in recent decades, sociological theorists have considered whether in modernity, society may come to be organized not around class (as Marx, and Weber, and others described), but around *risks*. As Anthony Giddens puts it, “when we stopped worrying so much about what nature could do to us, and we started worrying more about what we have done to nature, we moved from a society dominated by dangers to a society dominated by risks.” This vision of a *risk society* is fully articulated in Ulrich Beck’s book, Risk Society: Towards a New Modernity. These theories note that capitalism has not just created “goods,” but also “bads,” and these “bads” impose perceptible and imperceptible risks on various populations.

The risk society approach suggests that while risks are unevenly distributed, they ultimately will affect all strata of society. Beck famously notes that “while poverty is hierarchic, smog is democratic,” and defines a *boomerang effect* in which even the most advantaged members of society will eventually be afflicted by the risks that society creates. In one study of Latin American cities, air quality is uncorrelated with social advantage across space, providing evidence for the *risk society* interpretation. Mortality rates and social disadvantage (or “vulnerability”) for these cities are shown here:



Note: These maps show from left to right the geographic distribution of human mortality rates attributable to cardiovascular and respiratory disease, and the distribution of vulnerable groups as measured by the multidimensional vulnerability index (MVI) in the three cities. Data in the maps are divided into five equal groups. The first quintile contains the lowest 20% of values, while the fifth quintile has the highest 20%.

The environmental justice framework offers a different way to think about air quality and other risks. As we noted earlier, EJ sprung from a movement protesting the unequal siting of hazardous waste facilities in North Carolina. EJ scholars might point out that it is difficult to ignore inequalities in air quality across the U.S, particularly those related to proximity to industrial facilities: the overwhelming majority of the most polluting industrial facilities are located in disadvantaged neighborhoods. Factors such as race and unemployment are frequently associated with proximity to these types of hazards. In some cases, such as West Oakland, urban planners purposely designed highways and polluting facilities to encompass Black neighborhoods. Another study finds that 1930's redlining is associated with current air quality levels:



An environmental distributive justice approach would emphasize the needs to redistribute hazards equally among populations. Currently, Black children in the U.S. are twice as likely as White children to have asthma, demonstrating some of the unequal burdens of air pollution. Asthma rates are on the rise, and make up a significant public health issue. An EJ approach to this problem would consider the underlying conditions that cause these unequal burdens of air pollution, such as discriminatory urban planning and environmental racism.

6.4 Using Tigris to Gather Shapefiles

We've seen one method for gathering data at the country level, but we may be interested in smaller units like counties or census tracts. If our scope of analysis is limited to within the United States, there is an R package, `tigris`, that will help us gather these data. We'll first put it in our library.

```
library(tigris)
```

If we type `tigris::` , we should see a list of functions within `tigris` . Most of these are different geographies that we can gather, such as:

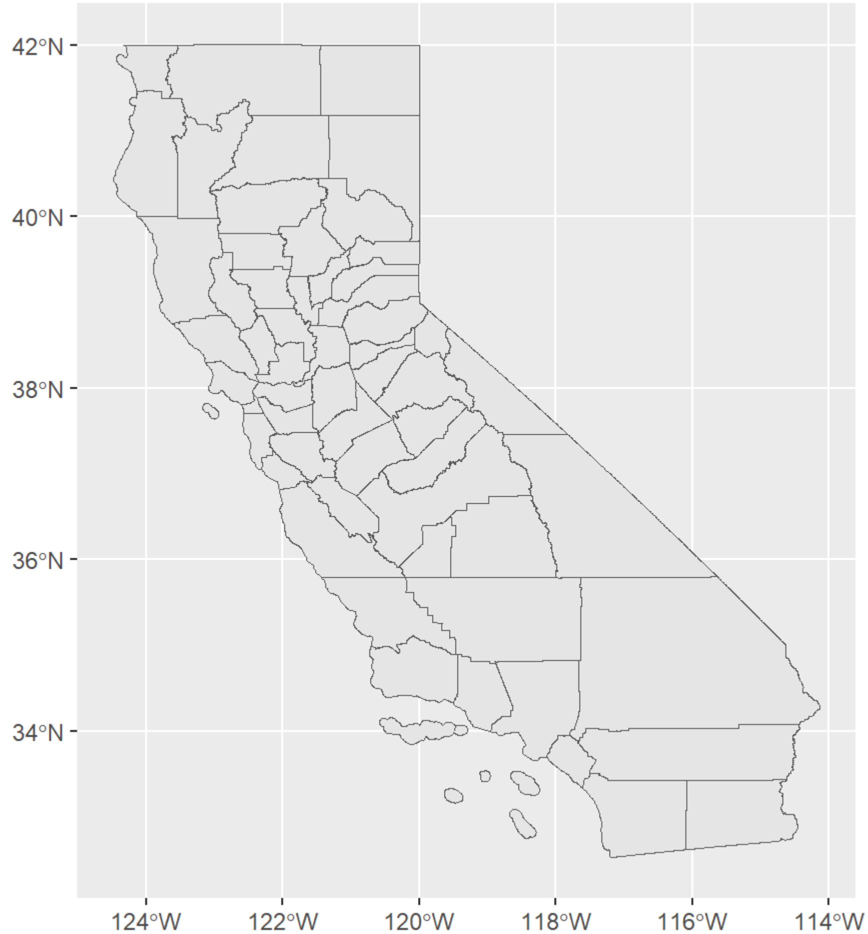
- Census block groups
- Census blocks
- Census tracts
- Counties
- Landmarks
- Native areas
- Roads
- School Districts

Let’s try counties! We can gather a shapefile of US counties with the following code. We can specify a particular state or year within the `counties()` function. For now, we will focus on California. Since we don’t specify the year, this will revert to the default (2021, which we can see in `?counties`).

```
# gather county shapefiles
counties <- counties(state = "CA")
```

Our `counties` data are already in simple features format. We can see this by looking at the head, or by running `class(counties)` . Let’s take a look at the counties! We can try plotting them:

```
# plot counties
ggplot(counties)+
  geom_sf()
```



Great, we have our counties in simple features format and they are plotting correctly.

We can now add other information at the county level to our data. In this section, we'll look at outdoor air quality data from the Environmental Protection Agency. We can download annual air quality index (AQI) data at the county level here. We'll do this for 2022.

```
# read in data
aqi22 <- read_csv("Data/annual_aqi_by_county_2022.csv")

# we can limit them to just California
aqi22 %<>%
  filter(State == "California")
```

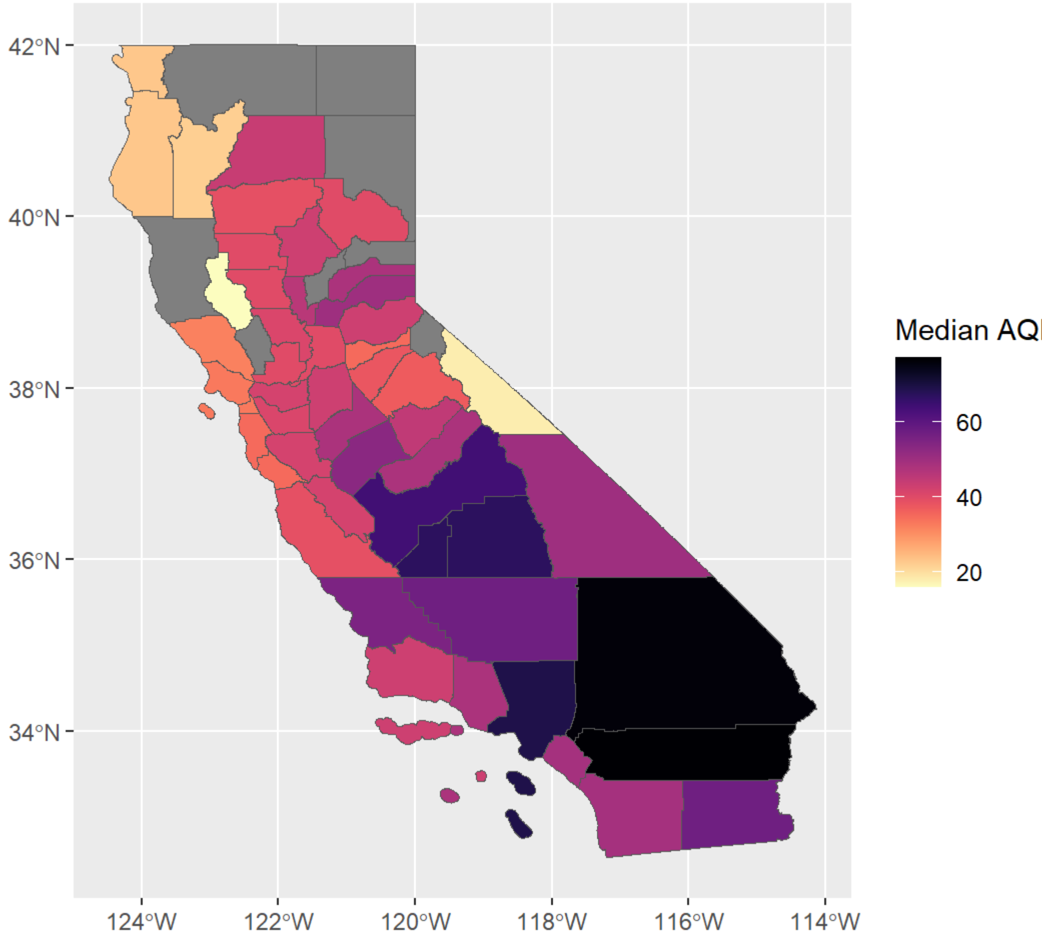
Great, now let's try joining the air quality data onto our county data! When we `left_join()` these data, we'll need to specify that the variable NAME in our left dataset (counties) should be matched with the variable County in our right dataset (aqi22).

```
counties %<>%
  left_join(aqi22, by = c("NAME" = "County"))
```

Any time we join dataframes it's important to make sure that this worked properly. We can do this a few ways. We can visually inspect our data by using the `View(counties)` function and looking at the new columns. We also want to make sure that our data did not change drastically in its number of rows, and we can use `nrow(counties)` to do so. We could run the `table()` function on different variables (e.g. `table(counties$Median AQI)`) to make sure that the new data are roughly what we would expect. In short, the more methods we can use to check that our data were transformed correctly, the better.

Now we can plot air quality for 2022 at the county level!

```
library(viridis)

ggplot(counties)+
  geom_sf(aes(fill = `Median AQI`))+
  scale_fill_viridis(option = "magma", direction = -1)
```



6.5 Spatially Joining Data

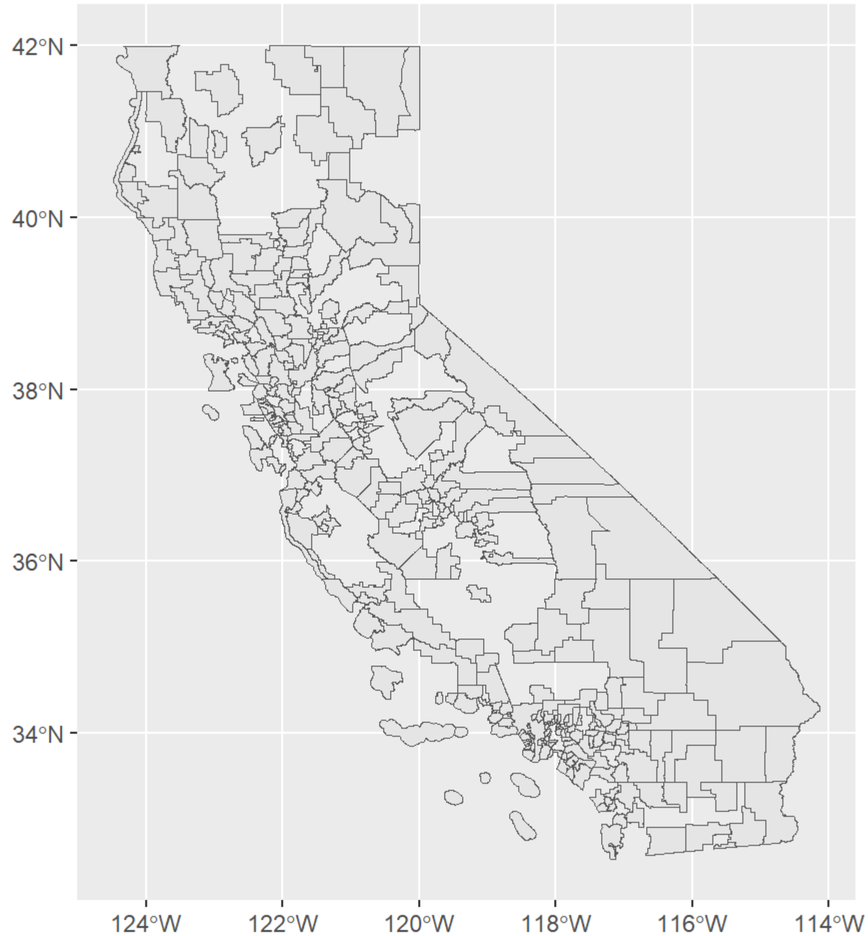
In the previous section, we gathered simple features at the county level and then found county data to join these with. However, what do we do if we want to join two datasets with different spatial units? In this section we'll look at how we might join school districts with AQI data.

First, we'll gather school districts in California using the `tigris` package.

```
# download school districts
school_districts <- school_districts(state = "CA")
```

Great, we can plot these to do an initial visual inspection.

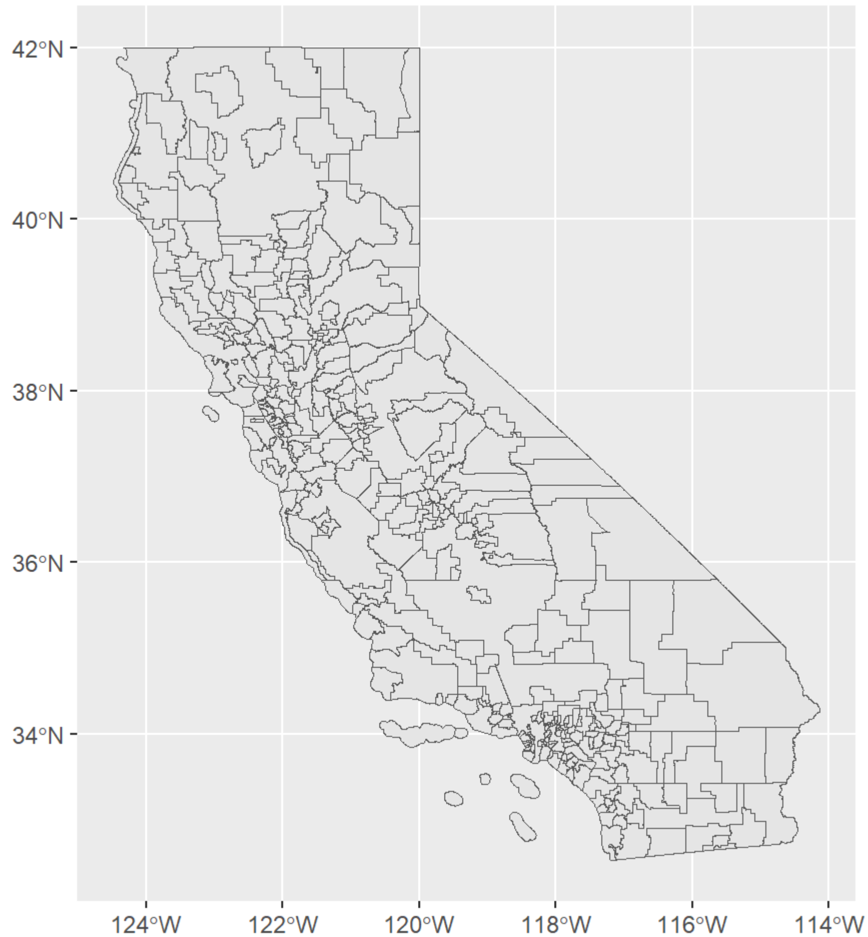
```r
# plot school districts
ggplot(school_districts)+
  geom_sf()
```



Since there are some areas with missing shapes (e.g. national forests or unpopulated regions), we might want the boundary for the entire state in our plot as well. We can do this by dowloading the state file, and adding it to our `ggplot` .

```r
# get state of california shape
ca <- states() %>%
  filter(NAME == "California")
```

```
# try plotting again
ggplot(school_districts)+
  geom_sf(data = ca)+
  geom_sf()
```



Great! Now let's look at AQI. We could look at the county-level data, but school districts tend to be a bit smaller than counties. Instead, we can gather the data at the monitor-level here.

```
# read in monitor-level data
aqi22m <- read_csv("Data/annual_conc_by_monitor_2022.csv")
```

These data contain latitude and longitude coordinates, but are not yet in simple features format. We also notice by visually inspecting these data that there is a column called "Datum," which is equal to "NAD83" for some entries and "WGS84" for others. Both of these are common CRSs. We can set the CRS of these new data to that of our school districts.

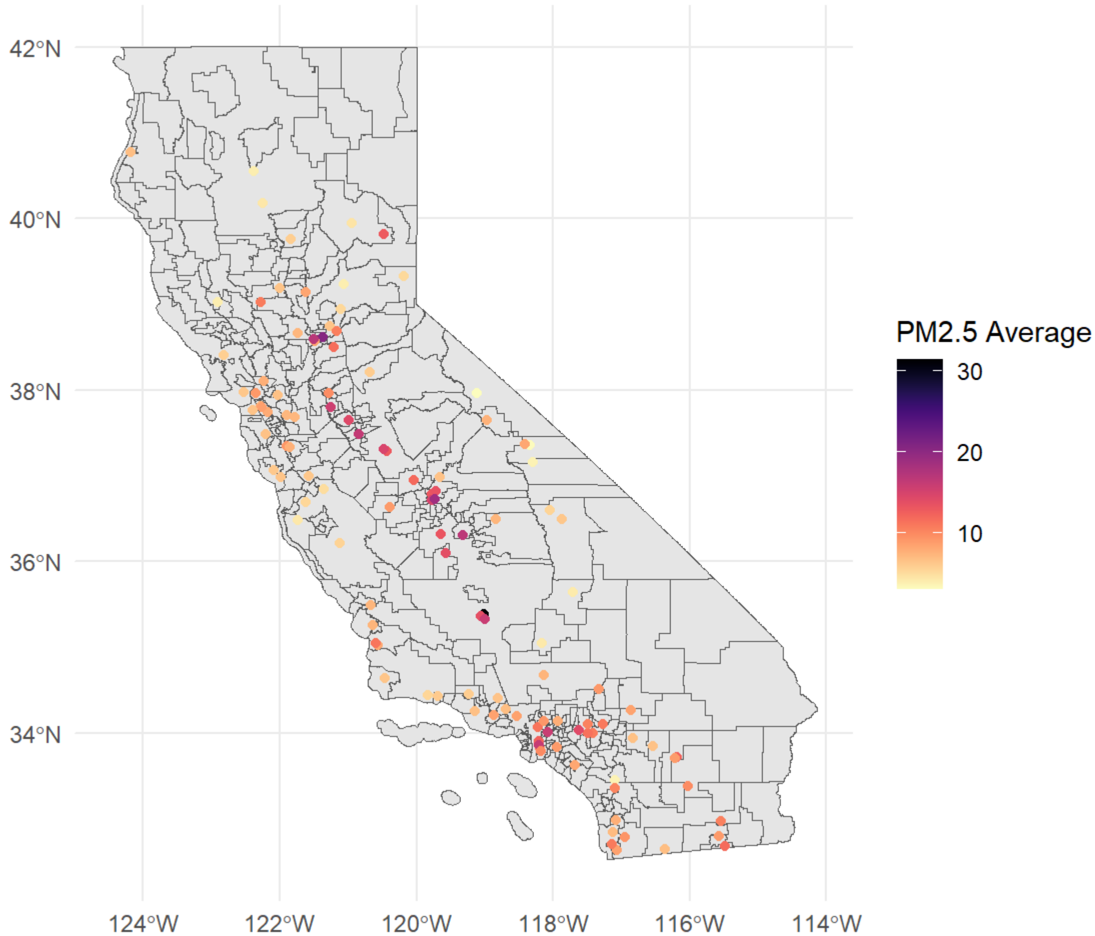
```
# make aqi monitor data sf
aqi22m %<>%
  st_as_sf(coords = c("Longitude", "Latitude"))

# set crs
aqi22m %<>%
  st_set_crs(st_crs(school_districts))
```

We also notice that our AQI data here include many types of pollutants. We'll limit our data to just look a PM2.5 compared to its 24 hour standard set in 2012.

```
# limit pollutant
aqi22m %<>%
  filter(`Pollutant Standard` == "PM25 24-hour 2012" &
           `State Name` == "California")
```

Now let's try plotting these data over the California school districts.

```
# try plotting again
ggplot(aqi22m)+
  geom_sf(data = ca)+
  geom_sf(data = school_districts)+
  geom_sf(aes(col = `Arithmetic Mean`))+
  scale_color_viridis(option = "magma", direction = -1)+
  theme_minimal()+
  labs(col = "PM2.5 Average")
```



Fantastic, we can now see the air quality monitoring sites overlayed on school districts. We notice that many school districts do not have a single air quality monitor within them (at least for PM2.5), while others have multiple monitors located within their boundaries.

We may want to aggregate air quality metrics for each school district with at least one monitor. Right now, our district boundaries and AQI measures are in two separate dataframes, so we'll need to merge them together. However, they share no common variables. So how will we do this?

We will use a *spatial join*. Specifically, we can use `st_join()` from the `sf` package.

```
# join school district boundaries with AQI
school_districts %<>%
  st_join(aqi22m)
```

We will then group the data by GEOID, which uniquely identifies each school district. We can now use `head(school_districts)` to see that our data have 390 observations, 68 variables, and 345 groups.

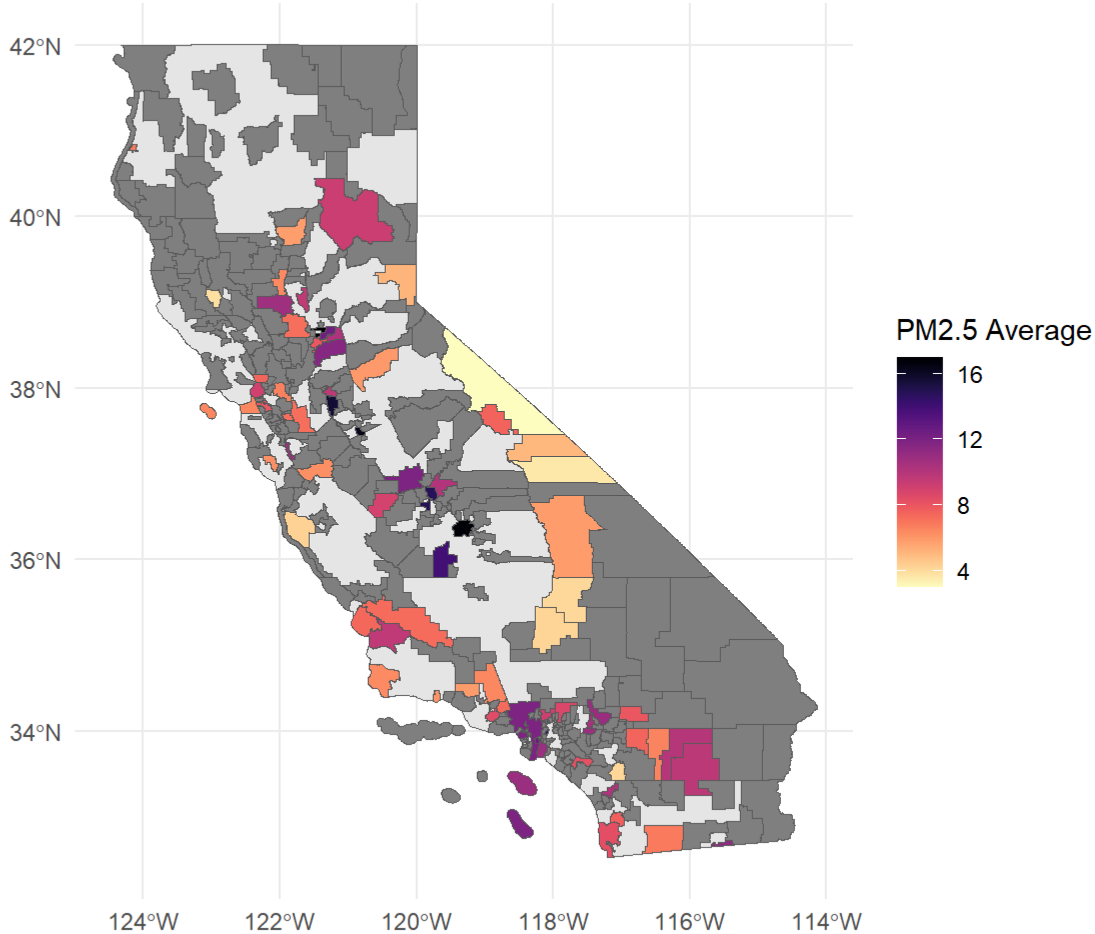
```
# group by school districts
school_districts %<>%
  group_by(GEOID)
```

Let's average the AQI for each school district. Now that we've grouped by GEOID, we can use the `summarise` function to get the average (or any other summary statistic) for each grouping variable (in this case, the district GEOID).

```
# average AQI for each school district with data
school_districts %<>%
  summarise(mn_aqi = mean(`Arithmetic Mean`, na.rm = T))
```

Great! Now we can plot the average AQI for the entire districts, filling them in. Note that we will use `fill` rather than `col` in the `aes()` arguments this time because we want to fill in shapes rather than points.

```
# try plotting again
ggplot(school_districts)+
  geom_sf(data = ca)+
  geom_sf(aes(fill = mn_aqi))+
  scale_fill_viridis(option = "magma", direction = -1)+
  theme_minimal()+
  labs(fill = "PM2.5 Average")
```



We see that many observations are missing, but that in general, school districts in the Central Valley tend to experience greater levels of air pollution, on average, than school districts closer to the coast or to the Nevada border.

We might wonder whether these pollution levels are associated with other district characteristics, such as poverty. We can get information on district demographics from the `educationdata` package. We'll download the "SAIPE" information, which stands for small area income and poverty estimates. These are simply census estimates of the poverty rates of children aged 5-17 in each school district.

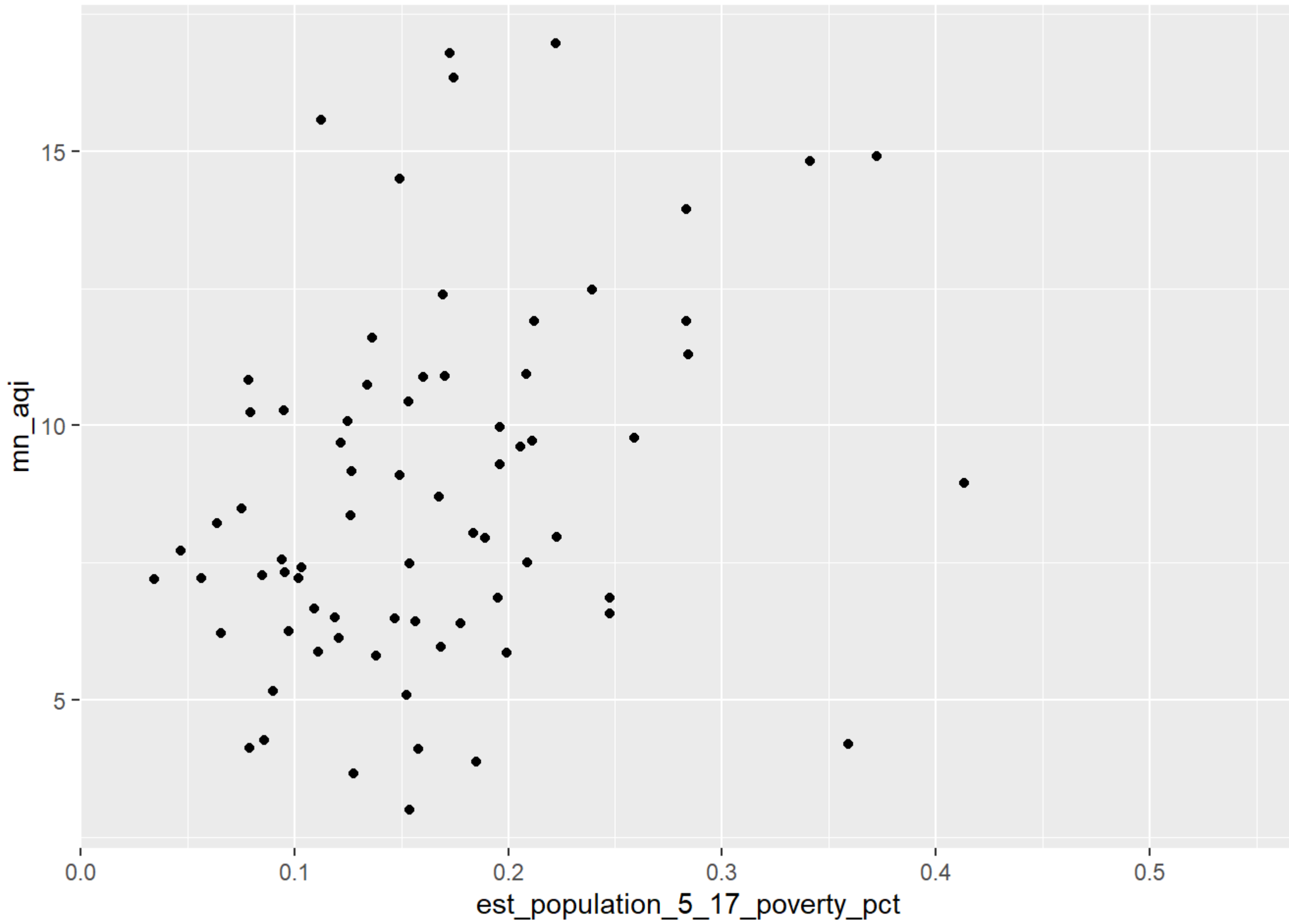
```
library(educationdata)

# download small area income and poverty estimates
districts_saipe <- get_education_data(level = 'school-districts',
                         source = 'saipe',
                         filters = list(year = 2021),
                         add_labels = TRUE)
```

Then we can join these with our district data. We'll specify that we want to join the "GEOID" variable with the "leaid" variable, both of which contain the districts' IDs.

```
# join school districts with poverty data
school_districts %<>%
  left_join(districts_saipe, by = c("GEOID" = "leaid"))
```

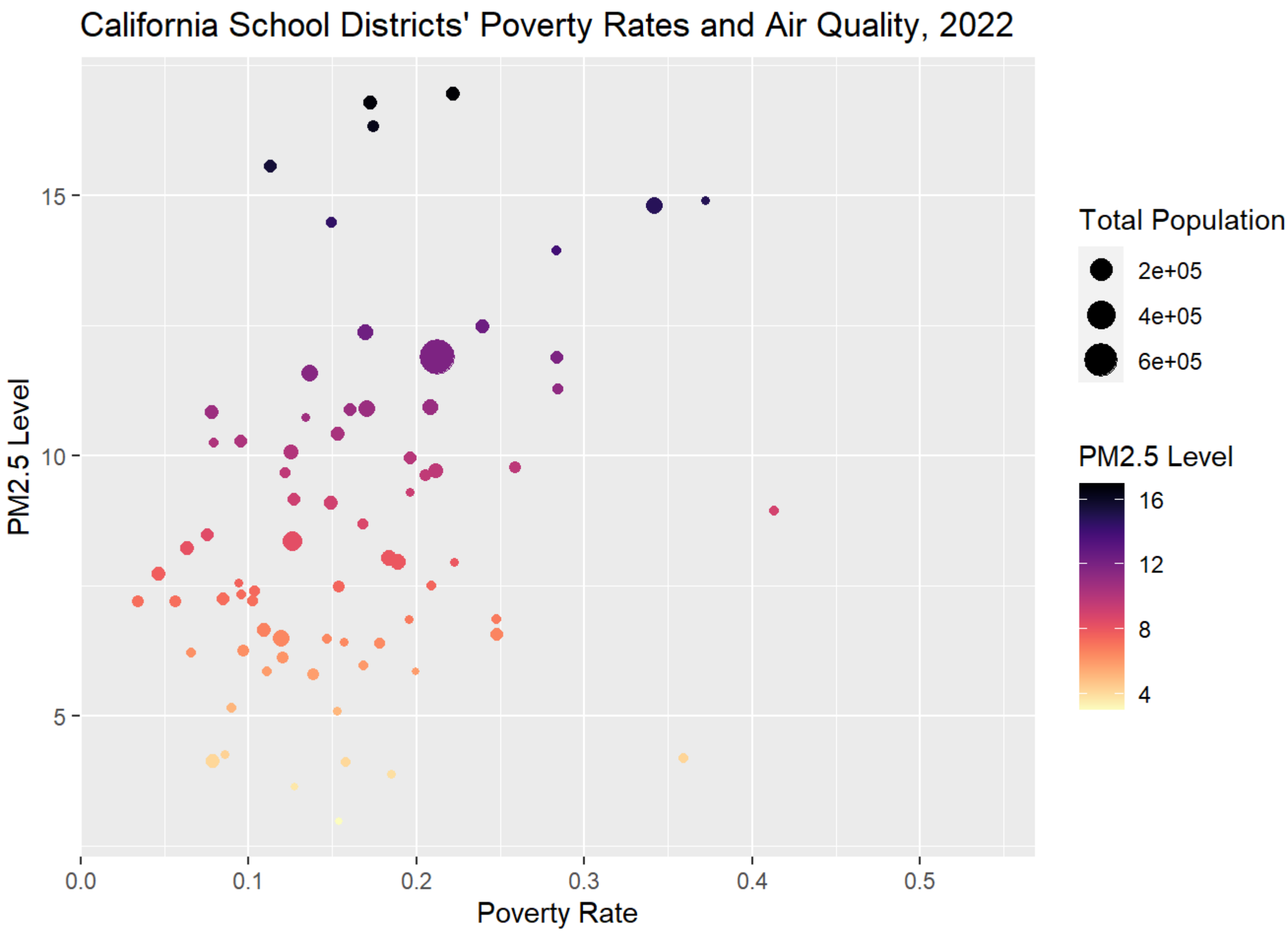
Now, let's look at air quality *and* district poverty rates at the same time.

```
# plot aqi and pov rates
ggplot(school_districts, aes(est_population_5_17_poverty_pct,
                             mn_aqi))+
  geom_point()
```



It appears there is a positive relationship between poverty rates and air pollution. We can improve this graph by making the size of each point proportional to the student population, and by modifying the labels. We'll also color the points according to the air quality levels.

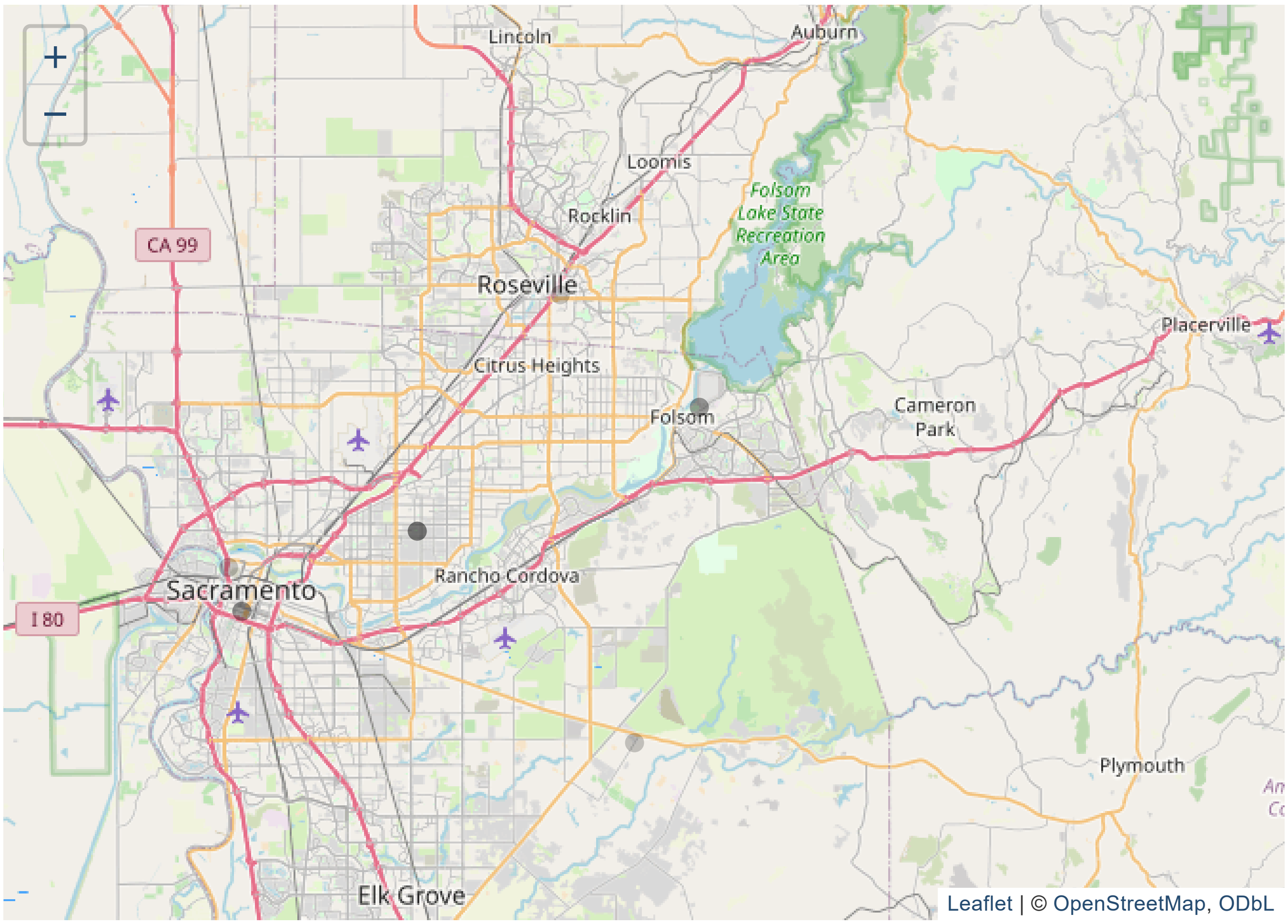
```
# plot aqi and pov rates
ggplot(school_districts, aes(est_population_5_17_poverty_pct,
                             mn_aqi))+
  geom_point(aes(color = mn_aqi, size = est_population_5_17))+
  labs(title = "California School Districts' Poverty Rates and Air Quality, 2022",
       x = "Poverty Rate",
       y = "PM2.5 Level",
       color = "PM2.5 Level",
       size = "Total Population")+
  scale_color_viridis(option = "magma", direction = -1)
```



6.6 Using Leaflet and R Shiny for Interactive Maps

We can use Leaflet to make interactive maps in R. For example, we can plot our air quality monitors in California by specifying that `data = aqi22m` . The `addTiles()` command simply adds the default Leaflet map polygons, and `addMarkers()` adds the AQI monitor points (because our data are point data).

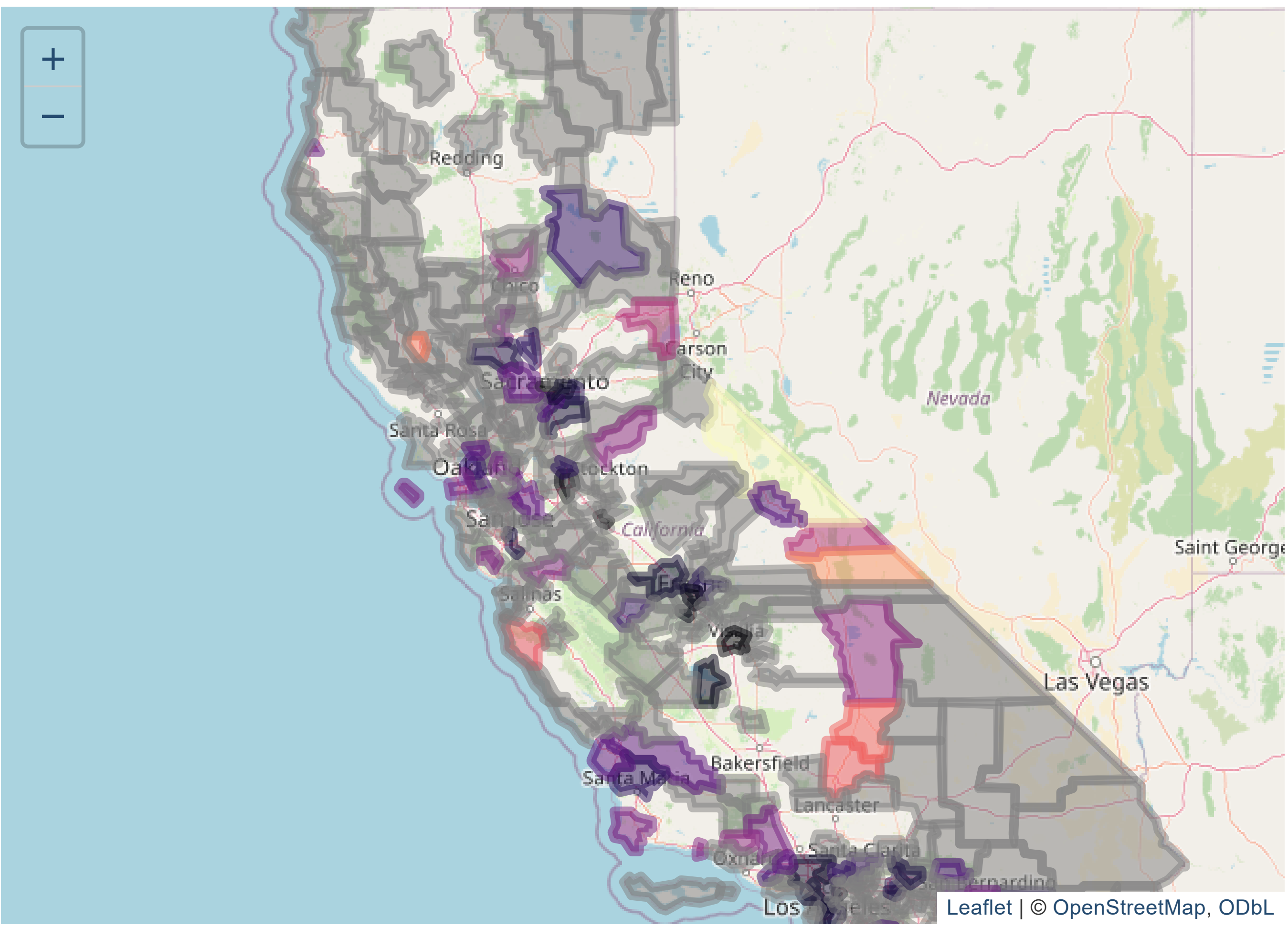
```
library(leaflet)

m <- leaflet(data = aqi22m) %>%
  addTiles() %>%  # Add default OpenStreetMap map tiles
  addCircleMarkers(color = ~`Arithmetic Mean`,
                   radius = 5)
m  # Print the map
```



We can see here, as in our previous maps, that poor air quality tends to be concentrated in the central valley and in Southern California. Unlike previous maps, we can zoom in and see specific monitors and neighborhoods.

We can also use Leaflet to plot polygons. We'll try this with our school district data. We'll first set a color function using `colorNumeric` from the `leaflet` package. This will translate our column data to colors for Leaflet to read. We will take the inverse of our data in order to reverse the color scale here.

```r
library(leaflet)

# function to transform colors
col <- colorNumeric(palette = "magma", 1/school_districts$mn_aqi)

s <- leaflet(data = school_districts) %>%
  addTiles() %>%  # Add default OpenStreetMap map tiles
  addPolygons(fillColor = ~col(1/mn_aqi),
              color = ~col(1/mn_aqi),
              fillOpacity = .5)
s  # Print the map
```



We can see the specific school districts with the highest PM2.5 levels in California. These tend to be clustered around Los Angeles, Fresno, and Sacramento. We could further adjust the color specifications and shape properties using the other arguments in `addPolygons()` .

Lastly, R Shiny is a package that allows the user to develop interactive graphics and user interfaces. Creating these applications goes a bit beyond the scope of this class, but we'll just look at some examples to see what is possible with this type of programming.

This Superzip Map shows several of the capabilities of R Shiny and Leaflet. Try playing around with it! You can zoom in, highlight different places, change the outcome variable, and look at how the descriptive statistics vary in different places. The graphic is a commentary on "Super Zips," or places where per capita income and education levels rank in the top 5% (but you can adjust this as well). If you'd like to learn more about R Shiny and try making your own apps, I recommend exploring the Tutorial pages as well as this user guide.

6.7 Problem Set 6

Recommended Resources:

Simple Features for R

ggplot2: Elegant Graphics for Data Analysis, by Hadley Wickham

1. Read the North Carolina shapefile into R using the process described in section 6.1. Then plot the data using `ggplot2`. Use a variable other than `AREA` to fill the shapes, and try using one of the `scale_fill_` arguments to create a red color scale. See these notes on color scales with ggplot for some theory and ideas here.
2. Read in the CDP hazards data and cities location data from Canvas ("cdp_hazards.csv" and "CDP-Cities-geographical-coordinates.csv"), and plot the cities according to at least one of the hazards. Discuss your graph and what conclusions you might draw about the geographies of risk.
3. Now do the same thing, but for the CDP actions data ("cdp_actions.csv"). Do you notice any differences between the cities facing hazards and those that are taking actions?
4. Use the `tigris` function to download shapefiles of your choice. Plot the data using `ggplot()`.
5. Download one of the clean AQI monitor datasets from Canvas ("aqi22_clean_us.csv" or "aqi22_clean_ca.csv") and plot these inside the boundaries that you have downloaded from `tigris`. Then, calculate the average PM2.5 levels in each of the geographical units, and

plot your geographies again according to these levels (as we did in section 6.5). Which areas had the best (and worst) air quality in 2022? What are the implications?