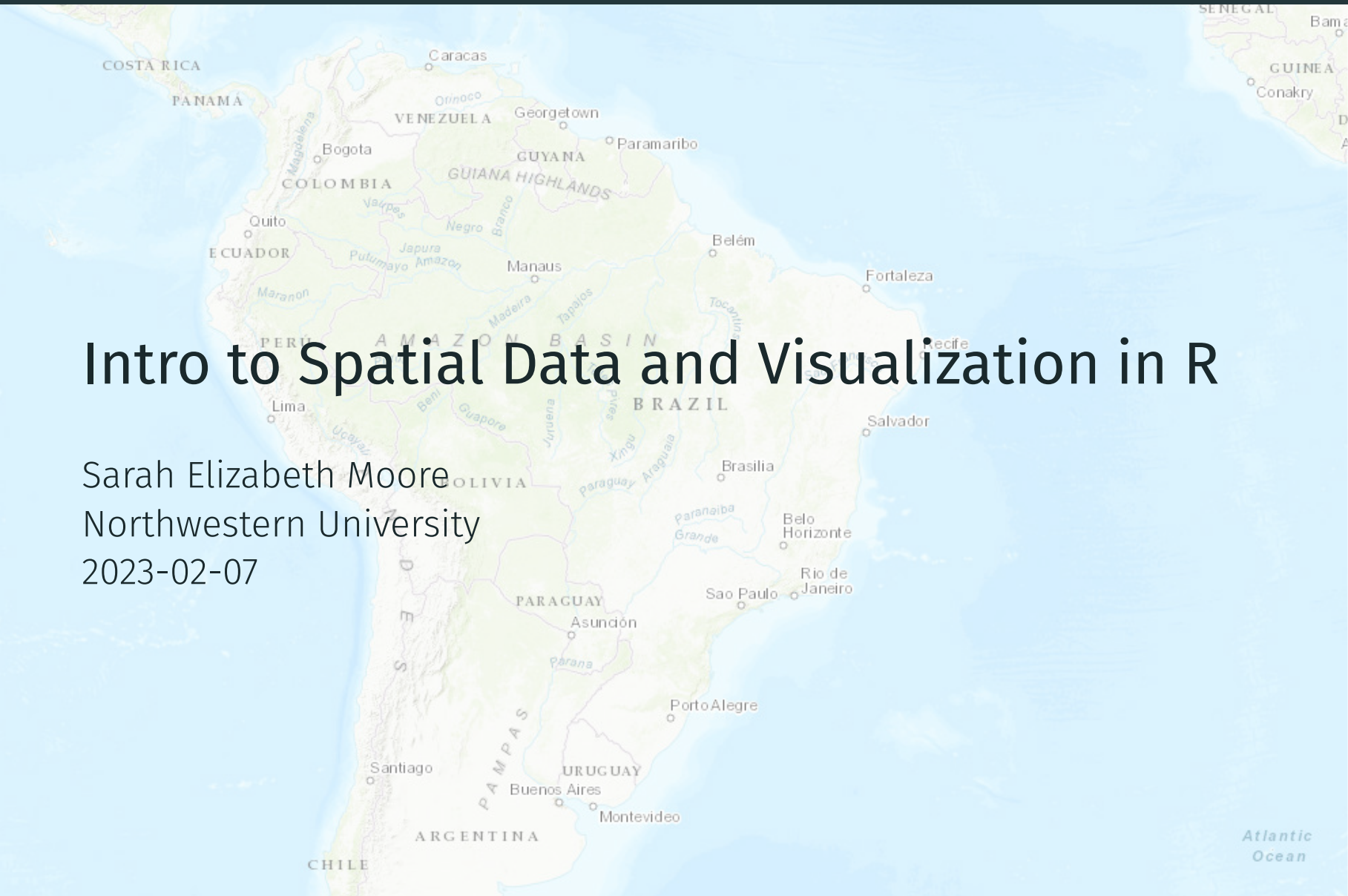


Intro to Spatial Data and Visualization in R

Sarah Elizabeth Moore
Northwestern University
2023-02-07

Workshop Goals:

- Distinguish between different types of spatial data classes and review of basic coordinate system vocabulary.
- Identify spatial data file types and import spatial data into R.
- Familiarize yourself with a few of the spatial data packages in R.
- Visualize static and interactive mappings with user-chosen spatial data.

Workshop Pre-Requisites

- Basic to intermediate base R skills.
- Basic tidyverse skills.
- Troubleshooting in R.

Geographic Data? Geospatial Data?

- Data that refer to and encode geographic locations into the datapoints.
- Ideally, each unit of analysis in geographic data would then have some geographic attributes attached to each observation.
    - For example, if we have a geographic dataset of every municipality in Illinois, we'd then expect that every municipality datapoint has associated geographic information of some form (e.g. latitude and longitude coordinates, zip codes, etc.).
- You might hear the term *geospatial data* instead, this is the same thing maybe just fancier sounding.

... GIS?

- The term GIS is a more specific reference to Geographic Information Systems.
- GIS and geospatial/geographic data *are not* interchangeable terms.

Types of Geospatial Data

- Vector: a series of points, lines, and polygons that represent a geographical feature or object
  - Good for representing discrete boundaries (school districts, electoral districts, neighborhoods and subdivisions)
- Raster: a series of grids with information based on satellite and other sensing techniques
  - Probably unlikely in the social sciences; useful for continuous data though we can still achieve this to an extent with vector data.

Types of Encoding

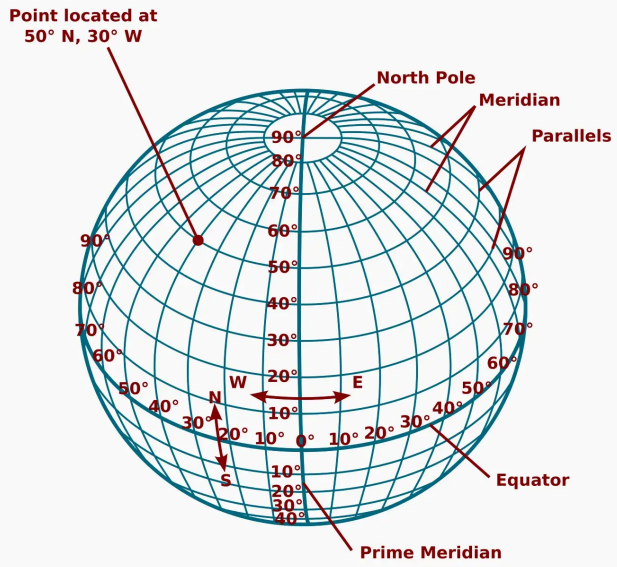
- Popular means of encoding standardized geographic information are familiar:
  - Addresses (e.g. 601 University Pl, Evanston, IL)
  - Zip Codes (e.g. 60208, 60625, 88130)
- Others are less familiar, or maybe used less day-to-day:
  - FIPS Codes for counties in the US
  - Census tracts
  - Coordinate Referencing System

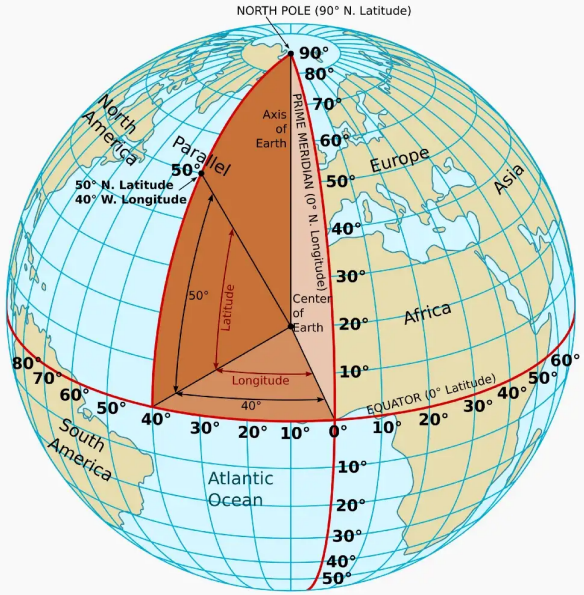
Coordinate Referencing System (CRS)

- In most cases the referencing system you will use is a *geographic coordinate system* , expressed as longitude and latitude.
- Basic familiarity and orientation on how these coordinates are transcribed is necessary.
- There are other CRS, which can make it tricky, but we'll go over the basics here. The other potential CRS are unlikely encounters for social science.[1]

[1] See this link to learn more: https://www.nceas.ucsb.edu/sites/default/files/2020-04/OverviewCoordinateReferenceSystems.pdf

Latitude and Longitude[1]





- Latitude: 0° to 90° North and South
- Longitude: 0° to 180° East and West

[2] Photos obtained from this nice explainer on Lat and Long.

Translating/Transcribing Coordinates

- Typically, when you look up a location's coordinates you'll get something like: 41.8766° N, 87.6384° W (The Old Post Office, Chicago, IL)
- However, we have to change how we think about this a little for coding purposes.
- Instead of thinking in N/S and E/W we have to translate into +/- across the equator and +/- across the prime meridian, respectively.
    - So (41.8766° N, 87.6384° W) → (41.8766°, -87.6384°)

Lat, Long Exercise

- Try this on a few more coordinates:
  - 30.3285° N, 35.4444° E (Petra, Jordan)
  - 33.8568° S, 151.2153° E (Sydney Opera House)
  - 50.4967° S, 73.1377° W (Perito Moreno Glacier, Argentina)

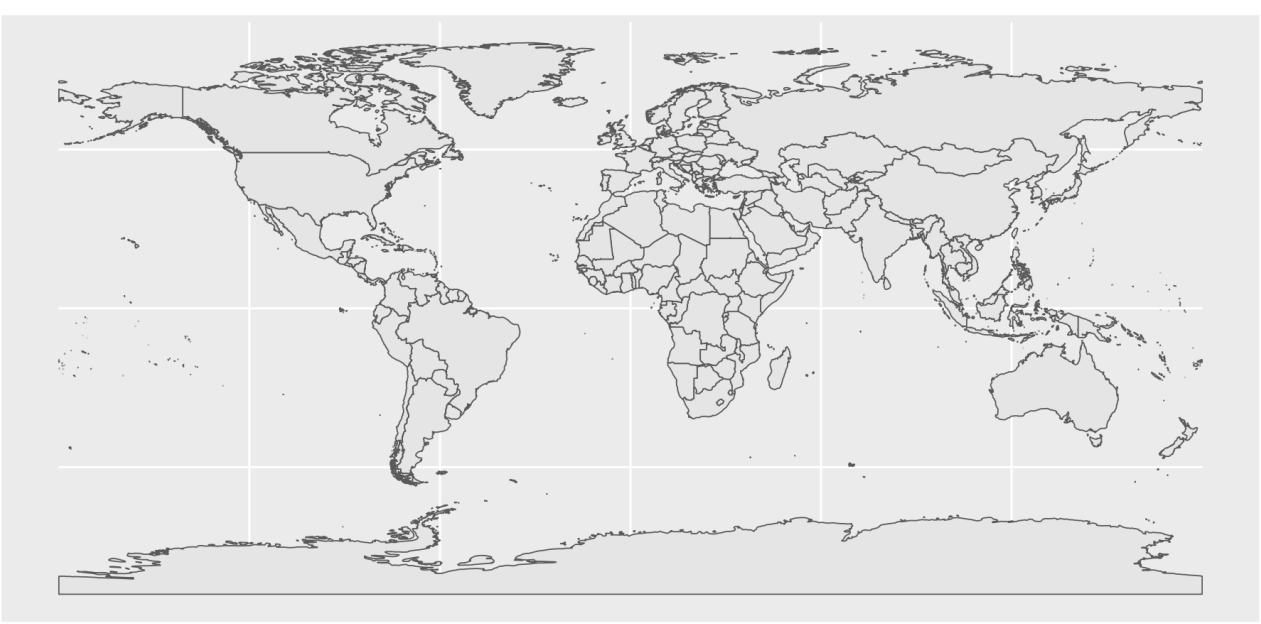
Important Packages

```
packages ← c("tidyverse", "ggmap", "sf", "rnaturalearth",
             "rnaturalearthdata", "leaflet", "tidycensus",
             "rgeos")

#install.packages(packages, dep = T)

invisible(lapply(packages, library, character.only = TRUE))
```

```
# may instead need these

install_github("r-spatial/sf", configure.args =
                 "--with-proj-lib=/usr/local/lib/")
```

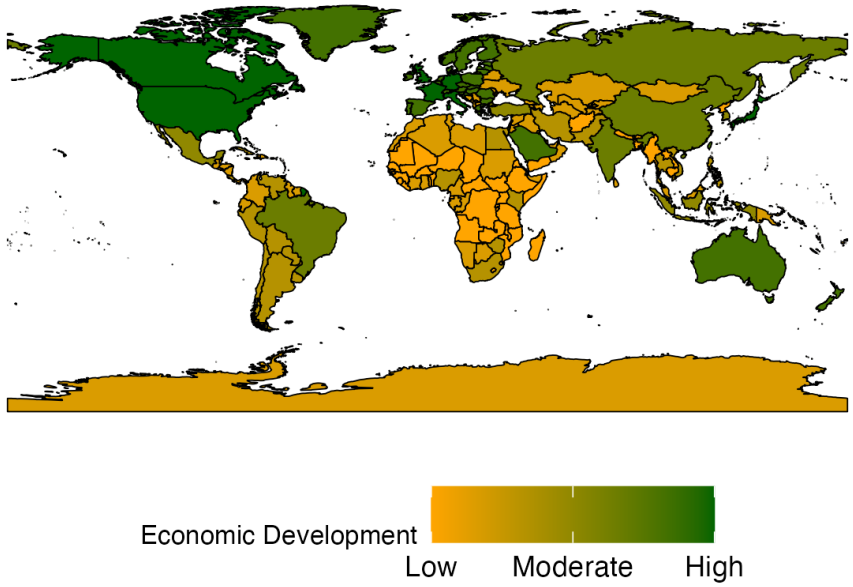
Built-In Options

```
world ← ne_countries(scale = "medium", returnclass = "sf")

ggplot(data = world) +
  geom_sf()→ world_map

world_map
```



```r
world$economy_c ← as.numeric(str_sub(world$economy, end= 2))

world$economy_c ← abs(world$economy_c-8)

ggplot(data = world, aes(fill = economy_c))+
   geom_sf(color = "black", lwd= 0.2) +
   scale_fill_gradient(low = "orange", high = "darkgreen",
                           name ="Economic Development",
                       breaks = c(1, 4, 7),
                       labels = c("Low", "Moderate","High")) +
   theme(
     panel.background = element_rect(fill = "white"),
     legend.position = "bottom",
     legend.title = element_text(size = 8)) +
   labs(title = "National Level Economic Development")→ econ_world
```

National Level Economic Development



Isolating countries

```
brazil ← ne_countries(geounit = "brazil
                      returnclass = "sf

ggplot(data = brazil)+
  geom_sf(fill = "#19AE47",
          alpha = 0.5) +
  theme(
    axis.text = element_blank(),
    panel.background = element_blank(),
    axis.ticks = element_blank()
  ) +
  labs(x = "", y = "", title = "Brazil")
```

Or highlighting countries

```
s_america ← ne_countries(continent = "s
                         returnclass =

s_america$brazil ← if_else(s_america$ge
                           1, 0)

ggplot(data = s_america, aes(fill =
                               as_factor
       )+
   geom_sf(alpha = 0.5) +
   scale_fill_manual(values = c("lightgr
                                "#19AE47
   annotate('text', x = -50, y = -12,
            label = "Brazil", size = 4,
            family = "serif") +
   theme(
     axis.text = element_blank(),
     panel.background = element_blank(),
     axis.ticks = element_blank(),
     legend.position = "none"
   ) +
   labs(x = "", y = "") → s_america_gg
```

Or showing the subnational admin.

```
brazil_states ← ne_states(country = "br

# brazil and its subnational administrat
ggplot(data = brazil_states)+
  geom_sf() +
  theme(
     axis.text = element_blank(),
     panel.background = element_blank(),
     axis.ticks = element_blank(),
     legend.position = "none"
   ) +
   labs(x = "", y = "",
        title = "States of Brazil")→ br
```

Beyond the built-in

It's unlikely that you will only ever want some geospatial data that is already built into a package in R. However, if you know you are dealing with some sort of administrative boundaries, it's always good to check if there is an existing package. For example, ACS data is easily pulled from existing packages in R.

Otherwise, it's necessary to find data files that contain the relevant information to create map plots.

- Most often you will encounter shapefiles (.shp). Even just querying something about your topic of interest plus "shape file" will get you a file with geospatial data that you can then import and use in R.
- Let's check out what happens when we download a shapefile of the Chicago School Districts.
    - When you download, you'll see that your file downloads as a .zip file with a bunch of other files. THESE ARE IMPORTANT! These are file dependencies that the .shp file requires to load so wherever you move the .shp file, these have to go along with it.

Anatomy of a ShapeFile

```
chicago_schools ← st_read("data/school/chicago-schools.shp", quiet = T)

chicago_schools %>%
  select(c(12,1,8,9,14:16))→ small_chicago
kableExtra::kbl(small_chicago[1:5,], full_width = F)
```

school_id	address	lat	long	ward_15	zip	geometry
400009	4647 W 47TH ST	41.80758	-87.74010	14	60632	POINT (-87.7401 41.80758)
400010	5410 S STATE ST	41.79612	-87.62585	3	60609	POINT (-87.62585 41.79612)
400011	3141 W JACKSON BLVD	41.87725	-87.70523	28	60612	POINT (-87.70523 41.87725)
400013	3986 W BARRY AVE	41.93730	-87.72710	30	60618	POINT (-87.7271 41.9373)
400017	3729 W LELAND AVE	41.96641	-87.72182	35	60625	POINT (-87.72182 41.96641)

ShapeFile Geometry Matters

```
ggplot(data = chicago_schools)+
  geom_sf() +
    theme(
     axis.text = element_blank(),
     panel.background = element_blank(),
     axis.ticks = element_blank(),
     legend.position = "none"
   )→ chicago_schools_map
```

```
chicago_schools_map
```

Adding Layers

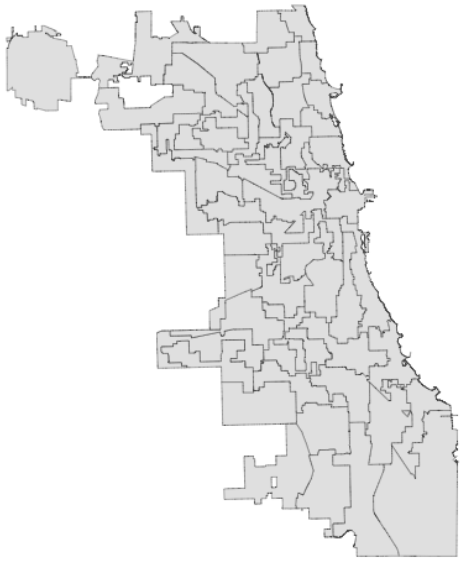
Since the last mapping only included the points, let's also figure out a way to add in the boundaries of Chicago. First, we have to load in another relevant shapefile.

Let's check out this shapefile, what is the geometry of these datapoints?

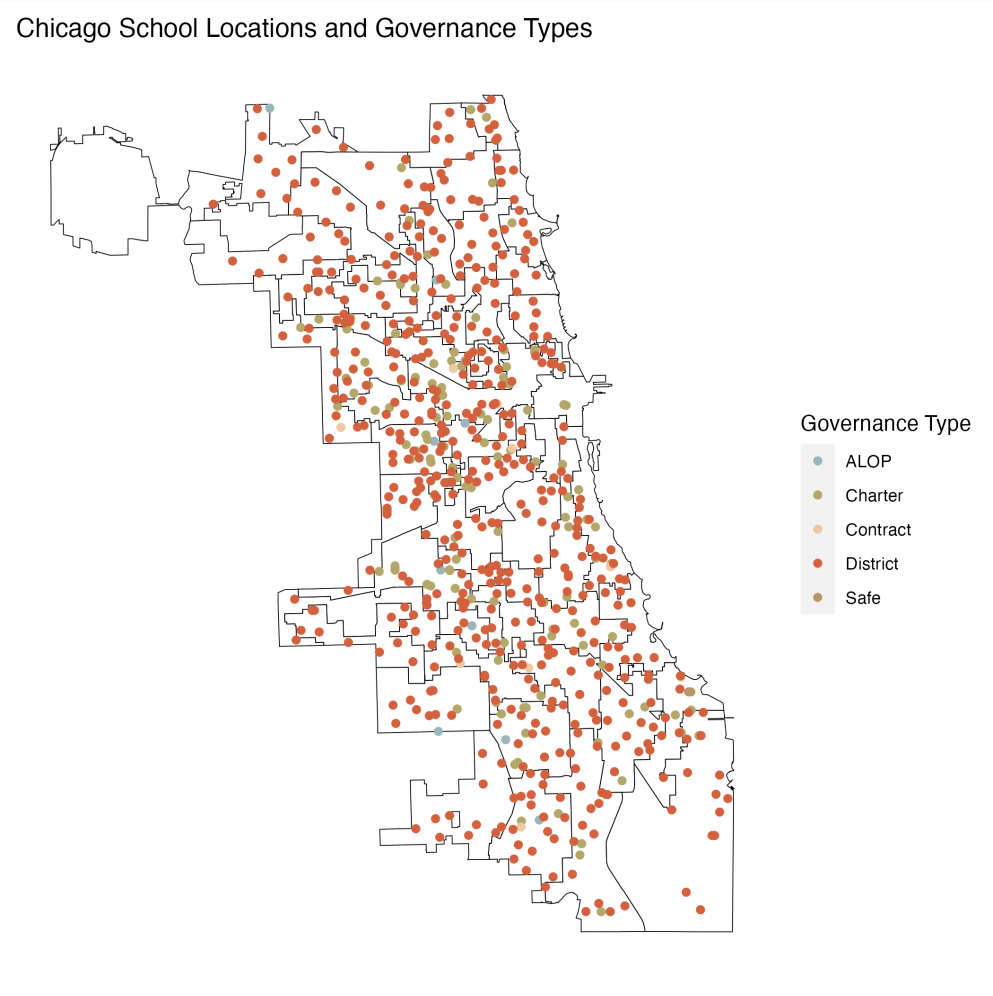
```
chicago_wards ← st_read("data/chicago/c
```

```
#chicago_wards
ggplot(data = chicago_wards)+
  geom_sf() +
    theme(
     axis.text = element_blank(),
     panel.background = element_blank(),
     axis.ticks = element_blank(),
     legend.position = "none"
   )→ chicago_wards_map
```

```
chicago_wards_map
```



```
ggplot() +
  geom_sf(data = chicago_wards,
          fill = "white", color = "black") +
  geom_sf(data = chicago_schools,
          aes(color = governance),
          alpha = 1) +
  paletteer::scale_color_paletteer_d("lisa::FernandoBotero",
                                     name = "Governance Type") +
  theme(
     axis.text = element_blank(),
     panel.background = element_blank(),
     axis.ticks = element_blank(),
  ) +
  labs(title = "Chicago School Locations and Governance Types")→ chicago_gov_schools
```


Chicago School Locations and Governance Types
Governance Type
ALOP
Charter
Contract
District
Safe

Exercise: ~10 min.

1. Install and load all of the packages on to your local machine. (Hint: Start out with the R script title "geodatar_exercises.R")

2. Import the Chicago Park District shapefile in the data subfolder "parks."

3. Create a layered map of (1) Chicago city wards in white with (2) Chicago parks in dark green.

4. Using some dplyr skills, as well as `st_join()` from the `sf` package, (1) count the number of parks per ward OR (2) sum the total acres of park space per ward and (3) fill in the ward with a gradient green based on the mutated variable that you created. (Hint: It's probably easiest and most efficient to first join these two dataframes.)

Exercise Code, Step 1 & 2

```
chicago_parks ← st_read("data/parks/chicago-parks.shp", quiet = T)

ggplot() +
  geom_sf(data = chicago_wards,
          fill = "white", color = "black") +
  geom_sf(data = chicago_parks,
          fill = "darkgreen") +
  theme(
     axis.text = element_blank(),
     panel.background = element_blank(),
     axis.ticks = element_blank(),
  ) +
  labs(title = "Chicago Park Disricts")→chicago_park_map

sf_use_s2(FALSE)
st_join(chicago_wards, chicago_parks, largest= T) → chicago_ward_parks
```

Option 1

```
# option 1
chicago_ward_parks %>%
  group_by(ward.y) %>%
  mutate(num_parks = n())→ chicago_ward_parks


ggplot() +
  geom_sf(data = chicago_ward_parks,
          aes(fill = num_parks)) +
  scale_fill_gradient2(low = "white",
                       high = "darkgreen",
                       name = "Number of Parks") +
    theme(
     axis.text = element_blank(),
     panel.background = element_blank(),
     axis.ticks = element_blank(),
  ) +
  labs(title = "Chicago Green Space by Ward")→chicago_ward_park_count
```

Option 2

```
# option 2
chicago_ward_parks %>%
  group_by(ward.y) %>%
  mutate(park_acres = sum(acres, na.rm = T))→ chicago_ward_acreage

ggplot() +
  geom_sf(data = chicago_ward_acreage,
          aes(fill = park_acres)) +
  scale_fill_gradient2(low = "white",
                       high = "darkgreen",
                       name = "Acres of Parks") +
    theme(
     axis.text = element_blank(),
     panel.background = element_blank(),
     axis.ticks = element_blank(),
  ) +
  labs(title = "Chicago Green Space by Ward")→chicago_ward_park_acreage
```

But what about different sources?

- Up until now, we have used data coming from the same source. This allows us to assume that we are dealing with similar projections and thus similiarly referenced coordinates.
- But, this is not always the case. Sometimes some data may be projected in Mercator, whereas another file that you need is projected in WGS84. Figuring out what projection you are working in between files is important. Mainly for analysis.
- `ggplot` is nice to us and reprojects all of our mapped objects into the same CRS anyway :). But here are some tips for checking to see if you're geodata are all in the same projection.
- In the event that you do need to transform something, the `st_transform()` function from the `sf` package is your best bet.

```r
library(tidycensus)

options(tigris_use_cache = TRUE)
nm_income ← get_acs(
  geography = "tract",
  variables = "B19013_001",
  state = "NM",
  year = 2020,
  geometry = T
  )

# check the line "geodetic CRS"
nm_income
st_crs(nm_income)

ggplot() +
  geom_sf(data = nm_income)

land_grants ← st_read("data/NM_grant_lands/nm_grant_lands.shp", quiet = T)

# check the line "geodetic CRS"
land_grants
st_crs(land_grants)
```

Beyond Visualization

- There are a lot of potential uses for geospatial data in R beyond visualization that I don't have time for here.
- You might be familiar with other tools like ArcGIS or QGIS that are more point-and-click style software for spatial analyses.
- However, R's specific strengths are the ability to *automate* and *reproduce* computational operations within a single interface where you are also performing other tasks such as visualization.[3]

[3] This site provides some more discussion on this topic.