Lab 9 - Spatial Data - Example Solution

Lab Purpose

This lab will enable us to practice working with spatial data. As you saw in the textbook reading and prep, working with spatial data often means making and customizing maps. Many of you chose to use maps in the Shiny app as well, so some of this will be review for some of you.

The packages we'll be working with for mapping are:

- maps: provides spatial data files for the world (`world`, `world.cities`, `lakes`), the US (`county`, `state`, `usa`, `us.cities`), France (`france`), Italy (`italy`), and New Zealand (`nz`). There is generally no need to load this package (the shapefiles are already loaded with **ggplot2**);
- sf: provides support for "simple features" objects, the standard data containers for spatial data;
- leaflet: allows us to create dynamic maps. If you are thinking about interactive maps for the final project, this is what you could use in Shiny.

Here are some additional potentially useful packages for working with spatial data or for obtaining shapefiles:

- ggspatial: adds additional annotations, geometries, and layers for building onto static maps made with `ggplot()`; and
- mapproj: converts latitude and longitude into projected coordinates, primarily with `mapproject()`.
- mapdata: an add-on to the **maps** package (includes `china`, `japan`, and `world2Hires` for a Pacific-centric world map);
- rnaturalearth: provides easy access to public domain map datasets from the Natural Earth project (tend to be higher resolution than data from the **maps** package)
- oz: map data for Australia and Australian states; and

- urbnmapr: US Census Bureau shapefiles (counties, states, territories)

IMPORTANT

Working with maps can generate many objects and the maps can take up a good bit of space/memory. In order to be able to view the objects easily, we want to keep our workspace clean. Before running any of the map code below, head to your Environment window, and hit the "broom" button. This will clear out your workspace. You will need to reload the package chunk at the top. In the lab below, due to the number of objects being generated, you will see another way to remove specific items from the Environment.

Cleaning out your workspace regularly is good practice. If it's always empty when you start working, it reduces the chance you'll run into issues with code "working, but at the same time not working" on saved objects that aren't loaded in the current .Qmd/.Rmd. Example: the data set from wrangling is in your Environment, but you're in a new .Qmd/.Rmd and didn't load it in. The new .Qmd/.Rmd won't compile, and you don't understand why. If the Environment was empty to start, it would remind you that you'd need to load in the data set.

1 - Using spatial data from the maps package

The **maps** package is loaded with **ggplot2** when we load **tidyverse** and provides a very limited set of map data . There are two ways to work with data from the maps package, outlined below.

The first approach require use of the `map_data()` function from **ggplot2**, which turns the spatial data from the **maps** package into a data frame. The first argument, `map` takes the shapefile of interest, and the second argument, `region`, can be used to identify subregions to include (the default is `region = "."`, which includes all subregions). To use this data, we add a `geom_polygon()` layer to the ggplot.

> part a - Run the code below to convert the `world` map data into a dataframe, take a peak at what the dataframe looks like, and then plot it with `ggplot()`.

```
# Get a dataframe with longitude and latitude
world_map_df <- map_data(map = "world")

head(world_map_df)
```

```
       long      lat group order region subregion
1 -69.89912 12.45200     1     1  Aruba      <NA>
2 -69.89571 12.42300     1     2  Aruba      <NA>
3 -69.94219 12.43853     1     3  Aruba      <NA>
4 -70.00415 12.50049     1     4  Aruba      <NA>
5 -70.06612 12.54697     1     5  Aruba      <NA>
6 -70.05088 12.59707     1     6  Aruba      <NA>
```

```
tail(world_map_df)
```

```
           long      lat group  order  region subregion
100959 12.43916 41.89839  1627 100959 Vatican   enclave
100960 12.43838 41.90620  1627 100960 Vatican   enclave
100961 12.43057 41.90547  1627 100961 Vatican   enclave
100962 12.42754 41.90073  1627 100962 Vatican   enclave
100963 12.43057 41.89756  1627 100963 Vatican   enclave
100964 12.43916 41.89839  1627 100964 Vatican   enclave
```

```
ggplot(world_map_df, aes(x = long, y = lat, group = group)) +
  geom_polygon(fill = "lightgrey", color = "white") +
```

```r
  # Use empty theme to remove background color, axes, ticks
  theme_void()
```


As an alternative, we turn the **maps** spatial data into an `sf` object and make use of a `geom_sf()` layer with `ggplot()`. This approach correctly preserves the aspect ratio of the map.

> part b - Run the code below to convert the `world` map data into a an `sf` object using `st_as_sf()` (from the **sf** package), take a peak at what this `sf` object looks like, and then plot the map with `ggplot()`.

Note: Because of the more common use of the `map()` function from **purrr**, it is good practice to explicitly identify the **maps** package when using its `map()` function.

```r
# Often preferred because it preserves aspect ratio
# Obtain sf object of world map
world_map <- maps::map("world", plot = FALSE, fill = TRUE) %>%
  st_as_sf()

head(world_map)
```

```
Simple feature collection with 6 features and 1 field
Geometry type: MULTIPOLYGON
```

```
Dimension:     XY
Bounding box:  xmin: -70.06612 ymin: -18.01973 xmax: 74.89131 ymax: 70.06484
Geodetic CRS:  +proj=longlat +ellps=clrk66 +no_defs +type=crs
                     ID                           geom
Aruba             Aruba MULTIPOLYGON (((-69.89912 1...
Afghanistan Afghanistan MULTIPOLYGON (((74.89131 37...
Angola           Angola MULTIPOLYGON (((23.9665 -10...
Anguilla       Anguilla MULTIPOLYGON (((-63.00122 1...
Albania         Albania MULTIPOLYGON (((20.06396 42...
Finland         Finland MULTIPOLYGON (((20.61133 60...
```

```r
ggplot(data = world_map) +
  geom_sf(fill = "lightgrey", color = "white") +
  theme_void()
```


part c - Your turn! Use data from the **maps** package to create a plot of New Zealand.

Solution:

```r
# same code as last chunk can be used, just change world to nz
```

```r
nz_map <- maps::map("nz", plot = FALSE, fill = TRUE) %>%
  st_as_sf()

head(nz_map)
```

```
Simple feature collection with 6 features and 1 field
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: 166.3961 ymin: -47.40573 xmax: 178.5629 ymax: -34.39895
Geodetic CRS:  +proj=longlat +ellps=clrk66 +no_defs +type=crs
                                          ID                           geom
North.Island                    North.Island  MULTIPOLYGON (((172.7433 -3...
South.Island                    South.Island  MULTIPOLYGON (((172.6391 -4...
Stewart.Island                Stewart.Island  MULTIPOLYGON (((167.8732 -4...
Great.Barrier.Island    Great.Barrier.Island  MULTIPOLYGON (((175.5359 -3...
Resolution.Island          Resolution.Island  MULTIPOLYGON (((166.4769 -4...
Little.Barrier.Island  Little.Barrier.Island  MULTIPOLYGON (((175.0999 -3...
```

```r
ggplot(data = nz_map) +
  geom_sf(fill = "lightgrey", color = "white") +
  theme_void()
```

part d - The **maps** package also allows us to plot counties (and states) within the US. Run the code below to get a county-level map for Massachusetts.

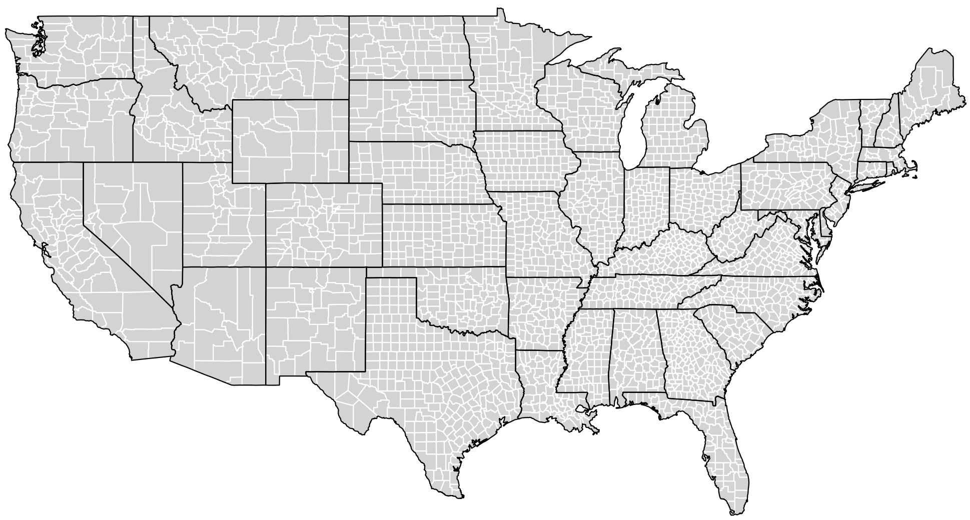
```
# Identify appropriate region value from county map data
county_df <- map_data("county")
# Run in console to see how data appear: View(county_df)

# Create sf object for Massachusetts counties
ma_map <- maps::map("county", regions = "massachusetts",
                    plot = FALSE, fill = TRUE) %>%
  st_as_sf()

ggplot(data = ma_map) +
  geom_sf(fill = "lightgrey", color = "white") +
  theme_void()
```


part e - See if you can create a light grey map of the US with states outlined in black and counties outlined in white.

Hints: remember that additional data can be read in within a geometry, and `fill = NA` may come in handy. For example, store county info in one data set and state information in another, then plot both.

Solution:

```r
# Load county and state map info
county_map <- maps::map("county", plot = FALSE, fill = TRUE) %>%
  st_as_sf()

state_map <- maps::map("state", plot = FALSE, fill = TRUE) %>%
  st_as_sf()

# plot using geom_sf twice, setting appropriate options as specified
ggplot() +
  geom_sf(data = county_map, fill = "lightgrey", color = "white") +
  geom_sf(data = state_map, fill = NA, color = "black") +
  theme_void()
```



2 - Choropleths

We may at times want to shade or color regions based on the value of a variable to create a map called a *choropleth*. Usually the process to do this is:

1. Identify a data source that provides the correct spatial information needed (this may be in the form of a shapefile already, as in the **maps** package, but this can be hard to find)
2. Identify a data source that provides the variable of interest for coloring the map
3. Join the two sources together (usually much harder than it sounds!)
4. Create the map!

Let's try this with data collected in July 2020 by The Chronicle for Higher Education and Davidson College's College Crisis Initiative (C2i) on colleges' reopening plans for Fall 2020. Our goal is to make a map of the US, with each state colored by the proportion of institutions in the state that were planning to be in person for Fall 2020.

We've already prepared the state spatial data (in e of Part 1), so now we are ready to prepare the Chronicle data. Run the code below to load the Chronicle data, conduct a little data wrangling to extract each school's plan (buried in the "X.1" column), and get a summary dataset with one row per state.

> part a - Take the time to look at the datasets that are created and try to make sense of the steps that were taken.

```
# Load data
college_plans <- read_csv("data/chronicle_plans.csv")

# Wrangle
college_plans <- college_plans %>%
  # Some plans are embedded within HTML tags within X.1 variable
  # so we want to extract all the text between ">" and "<"
  extract(col = X.1, into = "plans_extracted",
          regex = ">(.*)<", remove = FALSE) %>%
  # Combine extracted text with plain text of plans from X.1
  mutate(plans = case_when(is.na(plans_extracted) ~ X.1,
                           TRUE ~ plans_extracted)) %>%
  # Remove rows without plans
  filter(plans != "Link")

head(college_plans)
```

```
# A tibble: 6 x 6
  Institution                    Control State X.1          plans_extracted plans
```

```
  <chr>                          <chr>   <chr> <chr>          <chr>           <chr>
1 Abilene Christian University Private TX    "<a href=\"h~ Planning for i~ Plan~
2 Academy of Art University    Private CA    "<a href=\"h~ Proposing a hy~ Prop~
3 Adelphi University           Private NY    "<a href=\"h~ Proposing a hy~ Prop~
4 Adrian College               Private MI    "<a href=\"h~ Planning for i~ Plan~
5 Agnes Scott College          Private GA    "<a href=\"h~ Planning for i~ Plan~
6 Alabama State University     Public  AL    "<a href=\"h~ Planning for i~ Plan~
```

```
# Check summary of plans
college_plans %>% count(plans)
```

```
# A tibble: 5 x 2
  plans                              n
  <chr>                          <int>
1 Considering a range of scenarios    60
2 Planning for in-person           652
3 Planning for online               89
4 Proposing a hybrid model         250
5 Waiting to decide                 27
```

```
# Count colleges per state
colleges_per_state <- college_plans %>%
  count(State) %>%
  rename(n_colleges = n)

head(colleges_per_state)
```

```
# A tibble: 6 x 2
  State n_colleges
  <chr>      <int>
1 AK             2
2 AL            16
3 AR            12
4 AZ             5
5 CA           120
6 CO            14
```

```
# Count colleges in-person per state
college_plans_per_state <- college_plans %>%
```

```
  count(State, plans) %>%
  # Fill in 0s as needed (e.g., no schools in a state have in-person plans)
  ungroup() %>%
  complete(State, plans, fill = list(n = 0)) %>%
  filter(plans == "Planning for in-person") %>%
  rename(n_in_person = n)

head(college_plans_per_state)
```

```
# A tibble: 6 x 3
  State plans                  n_in_person
  <chr> <chr>                        <int>
1 AK    Planning for in-person           0
2 AL    Planning for in-person          15
3 AR    Planning for in-person          11
4 AZ    Planning for in-person           4
5 CA    Planning for in-person          24
6 CO    Planning for in-person           9
```

```
# Join for final dataset
college_plan_summary <- colleges_per_state %>%
  left_join(college_plans_per_state) %>%
  mutate(proportion_in_person = n_in_person/n_colleges)

head(college_plan_summary)
```

```
# A tibble: 6 x 5
  State n_colleges plans                  n_in_person proportion_in_person
  <chr>      <int> <chr>                        <int>                <dbl>
1 AK             2 Planning for in-person           0                0
2 AL            16 Planning for in-person          15                0.938
3 AR            12 Planning for in-person          11                0.917
4 AZ             5 Planning for in-person           4                0.8
5 CA           120 Planning for in-person          24                0.2
6 CO            14 Planning for in-person           9                0.643
```

Next, we combine the state-level college planning information with the state-level mapping information. The Chronicle of Higher Education dataset has a variable that contains two-letter abbreviations for states (e.g., "MA") whereas the state variable in the mapping dataset includes the full name of the state in lowercase letters (e.g. "massachusetts").

We can use the `state` datasets available in base R package (also used in a previous lab) to connect the state abbreviations to the state names.

> part b - Again, take the time to look at the datasets that are created and try to make sense of the steps that were taken.

```r
# Peek at data
head(college_plan_summary)
```

```
# A tibble: 6 x 5
  State n_colleges plans                  n_in_person proportion_in_person
  <chr>      <int> <chr>                        <int>                <dbl>
1 AK             2 Planning for in-person           0                0
2 AL            16 Planning for in-person          15                0.938
3 AR            12 Planning for in-person          11                0.917
4 AZ             5 Planning for in-person           4                0.8
5 CA           120 Planning for in-person          24                0.2
6 CO            14 Planning for in-person           9                0.643
```

```r
head(state_map)
```

```
Simple feature collection with 6 features and 1 field
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -124.3834 ymin: 30.24071 xmax: -71.78015 ymax: 42.04937
Geodetic CRS:  +proj=longlat +ellps=clrk66 +no_defs +type=crs
                     ID                           geom
alabama         alabama MULTIPOLYGON (((-87.46201 3...
arizona         arizona MULTIPOLYGON (((-114.6374 3...
arkansas       arkansas MULTIPOLYGON (((-94.05103 3...
california   california MULTIPOLYGON (((-120.006 42...
colorado       colorado MULTIPOLYGON (((-102.0552 4...
connecticut connecticut MULTIPOLYGON (((-73.49902 4...
```

```r
# State objects should appear in your Environment pane
data(state)

# Create data frame with state info
state_info <- data.frame(Region = state.region,
                         # Match state variable name in map data
```

```r
                         ID = tolower(state.name),
                         # Match state variable name in summary data
                         State = state.abb)
head(state_info)
```

```
  Region          ID State
1  South     alabama    AL
2   West      alaska    AK
3   West     arizona    AZ
4  South    arkansas    AR
5   West california    CA
6   West    colorado    CO
```

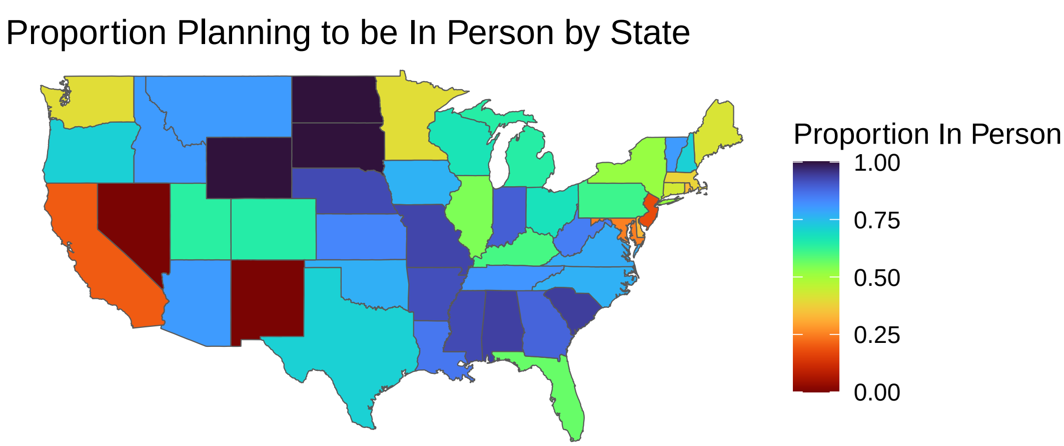
```r
# Join datasets from the left starting with the sf object ()
college_plans_map <- state_map %>%
  left_join(state_info) %>%
  left_join(college_plan_summary)

head(college_plans_map)
```

```
Simple feature collection with 6 features and 7 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -124.3834 ymin: 30.24071 xmax: -71.78015 ymax: 42.04937
Geodetic CRS:  +proj=longlat +ellps=clrk66 +no_defs +type=crs
           ID    Region State n_colleges                  plans n_in_person
1     alabama     South    AL         16 Planning for in-person          15
2     arizona      West    AZ          5 Planning for in-person           4
3    arkansas     South    AR         12 Planning for in-person          11
4  california      West    CA        120 Planning for in-person          24
5    colorado      West    CO         14 Planning for in-person           9
6 connecticut Northeast    CT         14 Planning for in-person           6
  proportion_in_person                           geom
1            0.9375000 MULTIPOLYGON (((-87.46201 3...
2            0.8000000 MULTIPOLYGON (((-114.6374 3...
3            0.9166667 MULTIPOLYGON (((-94.05103 3...
4            0.2000000 MULTIPOLYGON (((-120.006 42...
5            0.6428571 MULTIPOLYGON (((-102.0552 4...
6            0.4285714 MULTIPOLYGON (((-73.49902 4...
```

part c - Now, we can create a choropleth with `ggplot()`. Customize the plot below with a color palette of interest to you and useful labels and titles.

Solution:

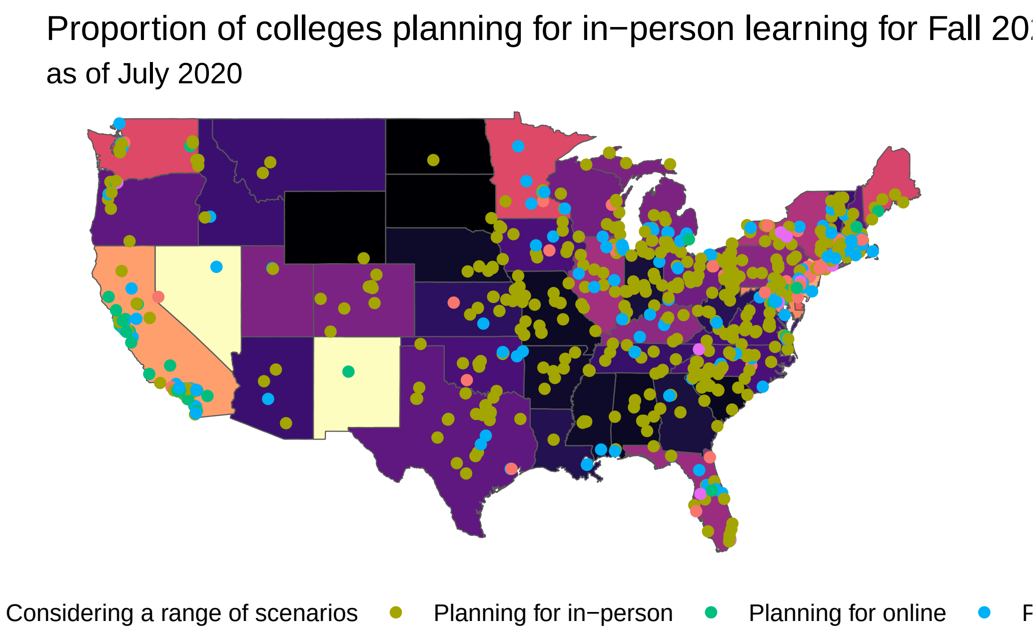
```
# default was viridis with magma, change to something else
ggplot(college_plans_map, aes(fill = proportion_in_person)) +
  geom_sf() +
  scale_fill_viridis(option = "turbo", direction = -1) +
  theme_void() +
  labs(title = "Proportion Planning to be In Person by State",
       fill = "Proportion In Person")
```



Instead of summarizing the college plans across institutions in a given state, we may want to plot a point for every college and add a visual cue to indicate each individual college plan (e.g., color the points by plan category). This requires getting spatial data for each institution, and then adding a layer to our graph. This information is not included in the Chronicle dataset, so we need to find it from somewhere else. The National Center for Education Statistics collects detailed location information for all of the higher education institutions in the US, and makes the data publicly available through IPEDS.

part d - Run the code below, again taking the time to look at the datasets that are created and try to make sense of the steps that were taken.

```r
colleges_initial <- read_csv("data/ipeds_directory_info.csv") %>%
  janitor::clean_names()

colleges <- colleges_initial %>%
  select(long = longitude_location_of_institution_hd2019,
         lat = latitude_location_of_institution_hd2019,
         Institution = institution_name,
         Type = control_of_institution_hd2019) %>%
  mutate(Type = factor(Type,
                       levels = c(1,2,3),
                       labels = c("Public",
                                  "Private, Not-for-profit",
                                  "Private, For-profit"))) %>%
  # Make sure coordinate projection matches our data
  st_as_sf(coords = c("long", "lat"),
           crs = 4326, agr = "constant")

colleges_map <- colleges %>%
  right_join(college_plans) %>%
  # State map is only of contiguous US (sorry, Alaska and Hawaii!!)
  filter(!(State %in% c("AK", "HI")))

ggplot(college_plans_map) +
  geom_sf(aes(fill = proportion_in_person)) +
  scale_fill_viridis(option = "magma", direction = -1) +
  geom_sf(data = colleges_map, aes(color = plans)) +
  theme_void() +
  labs(fill = "Proportion",
       color = "Plans",
       title = "Proportion of colleges planning for in-person learning for Fall 2020, by s
       subtitle = "as of July 2020") +
  theme(legend.position = "bottom")
```



part e - Not all of the institutions in the Chronicle of Higher Education's file matched to an institution in the IPEDS file. For instance, the Chronicle file has one row for Arizona State University, but the IPEDS file but has multiple rows for the same university to represent the location of the different campuses. What other types of mismatches are there? Can you think about how to clean up the mismatches?

Solution:

```
# for you to explore mismatches
# don't work to fix them, just strategize

# stated issues in question
college_plans %>%
  filter(str_detect(Institution,"Arizona State University")) %>%
  select(Institution)
```

```
# A tibble: 1 x 1
  Institution
  <chr>
1 Arizona State University
```

```r
colleges %>%
  filter(str_detect(Institution,"Arizona State University")) %>%
  select(Institution)
```

```
Simple feature collection with 5 features and 1 field
Attribute-geometry relationship: constant (1)
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: -112.16 ymin: 33.30714 xmax: -111.6772 ymax: 33.60726
Geodetic CRS:  WGS 84
# A tibble: 5 x 2
  Institution                                         geometry
  <chr>                                            <POINT [°]>
1 Arizona State University-Downtown Phoenix (-112.0735 33.45291)
2 Arizona State University-Polytechnic      (-111.6772 33.30714)
3 Arizona State University-Skysong          (-111.9239 33.46379)
4 Arizona State University-Tempe            (-111.9344 33.41772)
5 Arizona State University-West              (-112.16 33.60726)
```

For Arizona at least, we have options to separate, or do the join differently, since these look to be different campuses. This same issue appears for many other colleges with different campuses. Perhaps we need to do some separation of school and campus, and plot by campus if present.

3 - Your Turn - Create a map

Create a map of your choosing to display either country-level data on a world map, state-level data on a country map (keeping in mind the limitations of the **maps** package and of your time to figure out one of the other shapefile packages mentioned), or county-level data on a state map.

Note: If your group did a map as part of the Shiny app, you should choose a map that is DIFFERENT from what you already have experience with. For example, if you did a world map in the Shiny app, you should choose a state-level map for a country or county-level data on a state map here.

Don't spend too long looking for unit-level data you're interested in. I strongly recommend you use data readily available in an R package below (some sample code is provided to help you get started). For instance:

- the `gapminder` dataset from the **gapminder** package has (a few) country-level variables
- the `hate_crimes` dataset from the **fivethirtyeight** package has state-level variables
- the `states` data from base R has a matrix of state-level variables in the object `state.x77`
- example county-level data file - on unemployment from the USDA in the *data* subfolder.

You would need to install and make sure to load the package you aim to use.

Be sure that your figure has an appropriate title and legend for context, etc.

This map will be used for homework in Practice7. Again, be sure that the map you pick to work on is different in terms of what your group did for a map in the Shiny app (if your group had one).

```r
# Example of dataset with country information across years
library(gapminder)
data(gapminder)
head(gapminder)

# Example of dataset with state information from 2013-2014
library(fivethirtyeight)
data(hate_crimes)
head(hate_crimes)

# Example of another states dataset from Base R with state information from 1977
states_1977 <- data.frame(state.x77) %>%
  rownames_to_column(var = "State") %>%
  janitor::clean_names()
```

```r
# Example of dataset with county-level information from 2019
# (see second tab in excel file for variable explanations)
county_employment <- readxl::read_xls("data/usda-unemployment.xls",
                                      sheet = 1,
                                      skip = 7) %>%
  janitor::clean_names()
```

Solution:

Solutions will vary. This solution uses a world map, since we just focused on US maps for the last examples. This is the default if extra cleaning is not done - several countries are dropped due to issues with names not matching, including the USA, which is USA in one data set, but United States in the other. Those could be further investigated and fixed, if desired. Students should have something other than what was shown in their shiny apps if their group used a map.

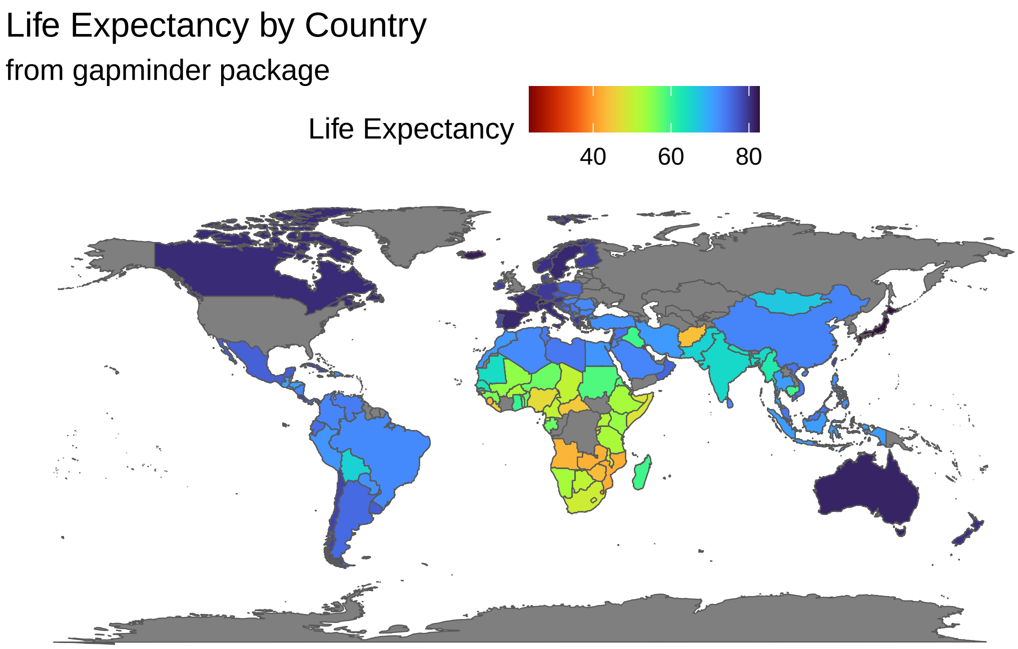
```r
# chosen data
library(gapminder)
data(gapminder)
head(gapminder)
```

```
# A tibble: 6 x 6
  country     continent  year lifeExp      pop gdpPercap
  <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
1 Afghanistan Asia       1952    28.8  8425333      779.
2 Afghanistan Asia       1957    30.3  9240934      821.
3 Afghanistan Asia       1962    32.0 10267083      853.
4 Afghanistan Asia       1967    34.0 11537966      836.
5 Afghanistan Asia       1972    36.1 13079460      740.
6 Afghanistan Asia       1977    38.4 14880372      786.
```

```r
# get world map again
world_map <- maps::map("world", plot = FALSE, fill = TRUE) %>%
  st_as_sf()

# join data sets
world_mapdata <- world_map %>% left_join(gapminder, by = c("ID" = "country"))
names(world_mapdata)
```

```
[1] "ID"        "continent" "year"      "lifeExp"   "pop"       "gdpPercap"
[7] "geom"
```

```r
ggplot(data = world_mapdata) +
  geom_sf(aes(fill = lifeExp)) +
  scale_fill_viridis(option = "turbo", direction = -1) +
   theme_void() +
  labs(fill = "Life Expectancy",
       title = "Life Expectancy by Country",
       subtitle = "from gapminder package") +
  theme(legend.position = "top")
```



Here is an example with the hate_crimes data and making a chloropleth map using one of the variables there.

```r
library(fivethirtyeight)
data(hate_crimes)
head(hate_crimes)
```

```
# A tibble: 6 x 13
  state      state_abbrev median_house_inc share_unemp_seas share_pop_metro
  <chr>      <chr>                   <int>            <dbl>           <dbl>
1 Alabama    AL                      42278            0.06             0.64
2 Alaska     AK                      67629            0.064            0.63
```

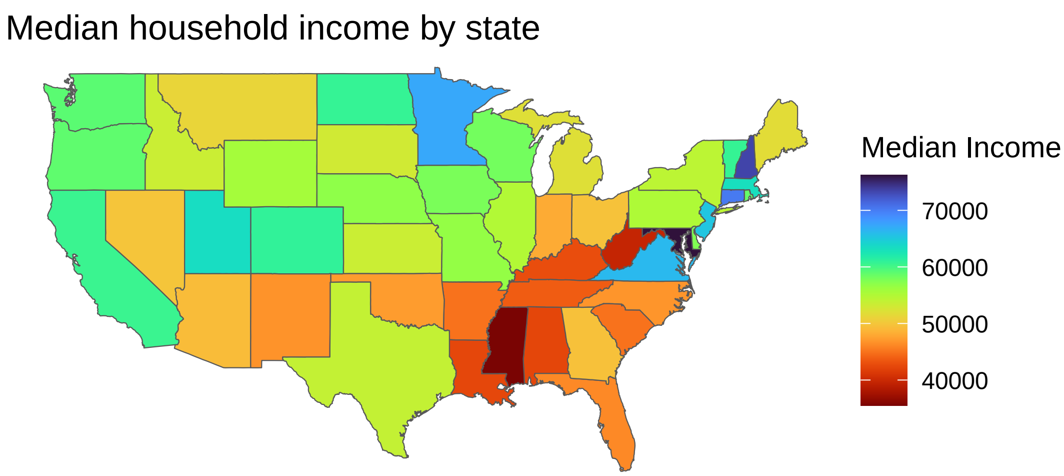
```
3 Arizona    AZ                            49254            0.063            0.9
4 Arkansas   AR                            44922            0.052            0.69
5 California CA                            60487            0.059            0.97
6 Colorado   CO                            60940            0.04             0.8
# i 8 more variables: share_pop_hs <dbl>, share_non_citizen <dbl>,
#   share_white_poverty <dbl>, gini_index <dbl>, share_non_white <dbl>,
#   share_vote_trump <dbl>, hate_crimes_per_100k_splc <dbl>,
#   avg_hatecrimes_per_100k_fbi <dbl>
```

Note that state here has capitalized letters, but in state_map below, it is lower case and the variable is called ID. We need to join these, but need to be careful in how we do it.

```
head(state_map)
```

```
Simple feature collection with 6 features and 1 field
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -124.3834 ymin: 30.24071 xmax: -71.78015 ymax: 42.04937
Geodetic CRS:  +proj=longlat +ellps=clrk66 +no_defs +type=crs
                     ID                         geom
alabama         alabama MULTIPOLYGON (((-87.46201 3...
arizona         arizona MULTIPOLYGON (((-114.6374 3...
arkansas       arkansas MULTIPOLYGON (((-94.05103 3...
california   california MULTIPOLYGON (((-120.006 42...
colorado       colorado MULTIPOLYGON (((-102.0552 4...
connecticut connecticut MULTIPOLYGON (((-73.49902 4...
```

```
# One option to join
hate_crimes <- hate_crimes %>% mutate(state = str_to_lower(state))

hate_crimes_map <- state_map %>%
  right_join(hate_crimes, by = c("ID" = "state")) %>%
  filter(!(ID %in% c("alaska", "hawaii")))

ggplot(hate_crimes_map) +
  geom_sf(aes(fill = median_house_inc)) +
  scale_fill_viridis(option = "turbo", direction = -1) +
  theme_void() +
  labs(fill = "Median Income",
       title = "Median household income by state")
```



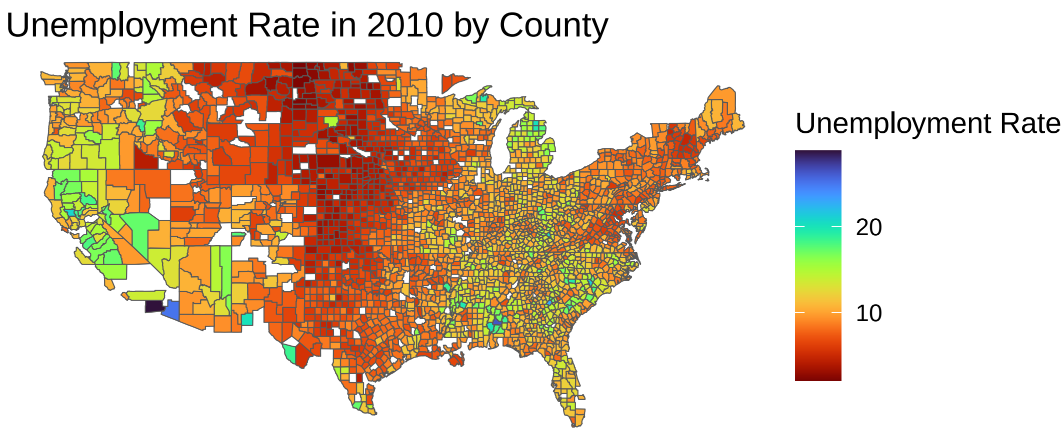
Finally, for a county map example, we need to do a little more wrangling.

```
# Identify appropriate region value from county map data
county_df <- map_data("county")
# Run in console to see how data appear: View(county_df)

# Create sf object for USA counties
county_map <- maps::map("county", plot = FALSE, fill = TRUE) %>%
  st_as_sf()
```

We see that ID is state,county. We'll need to make our data set match in order to merge.

```
# Example of dataset with county-level information from 2019
# (see second tab in excel file for variable explanations)
county_employment <- readxl::read_xls("data/usda-unemployment.xls",
                                      sheet = 1,
                                      skip = 7) %>%
    janitor::clean_names()


# get state info to merge in
data(state)
```

```r
state_info <- data.frame(Region = state.region,
                         # Match state variable name in map data
                         ID = tolower(state.name),
                         # Match state variable name in summary data
                         State = state.abb)


county_employment <- county_employment %>%
  left_join(state_info, by = c("stabr" = "State")) %>%
  drop_na()

county_employment_sub <- county_employment %>%
  select(ID, area_name, unemployment_rate_2010)

county_employment_sub <- county_employment_sub %>%
  separate(area_name,
           into = c("County", "State"),
           sep = ", ",
           remove = TRUE) %>%
  select(-State) %>%
  mutate(County = str_to_lower(County)) %>% #this is losing some locations, could be clean
  separate(County,
           into = c("county", "Text"),
           sep = " ",
           remove = TRUE) %>%
  select(-Text) %>%
  mutate(ID = paste(ID, county, sep = ","))


# One option to join
unemp_map <- county_map %>%
  inner_join(county_employment_sub)

ggplot(unemp_map) +
  geom_sf(aes(fill = unemployment_rate_2010)) +
  scale_fill_viridis(option = "turbo", direction = -1) +
  theme_void() +
  labs(fill = "Unemployment Rate",
       title = "Unemployment Rate in 2010 by County")
```


Unemployment Rate in 2010 by County
Unemployment Rate
20
10

4 - Leaflet

Explore interactivity with **leaflet**. What happens when you click on each state in the map below. Can you see where this happens in the code? Why is the map centered at Amherst to start?

Modify the example with Fall 2020 college reopening plans below (can you figure out how to fill the state color by proportion of colleges in person?), or try your hand at making your map from Part 3 interactive.

```
# Define a color palette over the values 0 to 1 (for proportion in person )
mypal <- colorNumeric(palette = "YlGnBu", domain = c(0,1))

# Identify Amherst College's location and pull the corresponding coordinates
amherst_coords <- colleges_initial |>
  filter(institution_name == "Amherst College") |>
  rename(long = longitude_location_of_institution_hd2019,
         lat = latitude_location_of_institution_hd2019)


# Create interactive map
leaflet(data = college_plans_map) %>%
  addTiles() %>%
  addMarkers(lng = amherst_coords$long,
             lat = amherst_coords$lat,
             popup = "Amherst College") %>%
  addPolygons(fillColor = ~ mypal(proportion_in_person),
              stroke = FALSE,
              popup = ~ paste0("State: ", ID %>% str_to_title(), "<br>",
                               "Number of schools reporting: ", n_colleges, "<br>",
                               "Number of schools planning for in-person learning: ",
                                  n_in_person, "<br>",
                               "Proportion planning for in-person learning: ",
                               proportion_in_person %>% round(2))) %>%
  setView(lng = amherst_coords$long,
          lat = amherst_coords$lat,
          zoom = 6)
```

Answers will vary based on what map you choose to make interactive and with what.

The example above has been changed with the fillColor in addPolygons using the defined palette “mypal”. You can define other palettes.