HW 14: Maps

Graphical Analysis of Biological Data

By the end of this assignment, you should be able to achieve the following tasks in R:

- use R notebooks and R markdown;
- insert, write, and evaluate code chucks;
- use pipes;
- use `ggplot2` to
  - create outline and filled maps at different scales,
  - obtain data from external databases, and
  - plot sample locations and distributions on the maps;
- customize plots to improve visualization;
- use a typical workflow to wrangle and plot data; and
- confidently stage, commit, and push with Git.

These achievements belong to Learning Outcomes 2, 3, 4, 5, 6.

By the end of this assignment, you should demonstrate an ability to

Click on any blue text to visit the external website.

This assignment has four parts.

Note: If you cannot get your code to run, open a new issue in the public discussion forum,, and describe the problem. Include the code that is not working and also tell us what you have tried.

Preparation

- No reading, no cheatsheets. Just pure mapping pleasure.

- Open your `.Rproj` project file in RStudio.

- Right-click and download hw14_data.zip. Unzip the file and move all of the data files into your data folder.

  Optional: If you want to keep your data folder a little more organized, create a folder called `shape_files` in your data folder. Move all of the files that begin with `cb_` into the `shape_files` folder. *Do not put any other data files in that folder. Then, in Part 2, use `here::here("data/shape_files", ""cb_2017_us_state_500k.shp")` when it comes time to import the shape files. You'll see where I did this in my code. If you don't want to do this, put all files in your `data` folder and delete the `/shape_files` from the `here()` argument.

- Create an `hw14` folder inside the same folder as your project file.

- Create a new notebook file and save it as `<lastname>_hw14.Rmd` inside your `hw10` folder.

- Add the YAML header as usual.

- Change the Knit directory to "Project Directory".

- Install these packages. **Do not install packages in a code chunk. Run the code in the console or `Tools > Install Packages...`** Including it in a code chunk would try to install the package every time you run the code. But then, you know that by now, right?

    - `maptools`
    - `maps`
    - `map_data`
    - `rgeos`
    - `sp`
    - `sf`
    - `ggmap`
    - `raster`
    - `marmap`
- Load these libraries, in the order shown. `tidyverse` must be loaded after `raster` so you can use its `select()` function. **Note:** Do not load all of the packages you just installed Some, like `raster` and `sp` will be accessed when needed by some functions.

```
library(here)
library(raster)
library(sf)
library(tidyverse)
library(ggthemes)
library(patchwork)
library(maptools)
library(ggmap)
library(marmap)
```

- Remember to following the code formatting guidelines.
- Commit early. Commit often. Push regularly.
- This assignment is part tutorial and part assignment I will give you an example, which you *must* run, and then you will do something similar as instructed. *Read the instructions carefully.* But then you knew that, right?

An important part of data visualization is not showing your data but showing were your data are from. You might need to show your sampling locations, the distribution of species, or demographic data for a population of people. You will do all of these in this assignment.

Part 1: Outline maps

Outline maps show outlines of regions of interest, such as the state boundaries of the U.S. states or the county boundaries of Missouri. In some cases, one or more regions might be filled to draw attention to a particular area.

Example: States and counties

I used this code to make a map for a colleague and her graduate student. The map is an outline map of the U.S. with Missouri filled in black and three other stated filled in gray. The figure was used for a new distributional record for a species of ant.

This example requires `tidyverse`, `ggthemes`, `maptools`, and `patchwork`. `maptools` accesses the `maps` package, which is a database of world countries, states, and U.S. county maps. `map_data` is a `ggplot2` function that accesses specific maps from the `maps` database.

U.S. map I used `map_data` and also `subset` to access specific regions (states, in this case) and store the results in three data frames.

```
# Outline of the 48 contiguous states
usa <- map_data(map = "state") # Tidyverse or ggplot2 must be loaded

# Outline of just Missouri
us_missouri <- map_data(map = "state",
                        region = "missouri")

# Outline of Alabama, Florida, Mississippi, and North Carolina
us_other <- subset(x = usa,
                   subset = region %in% c("alabama",
                                          "florida",
                                          "mississippi",
                                          "north carolina"))
```

The `usa` dataframe contains outlines of the 48 contiguous states. You can run `str(usa)` to see the structure or `head(usa)` to see the first few rows of the data. `us_missouri` contains the outline for the state of Missouri. `us_other` is a subset of the `usa` data set, containing outlines for the four states listed. **Note:** the region (state) names are lower case.

```
# You do not have to include this step in your code.
str(usa)
```

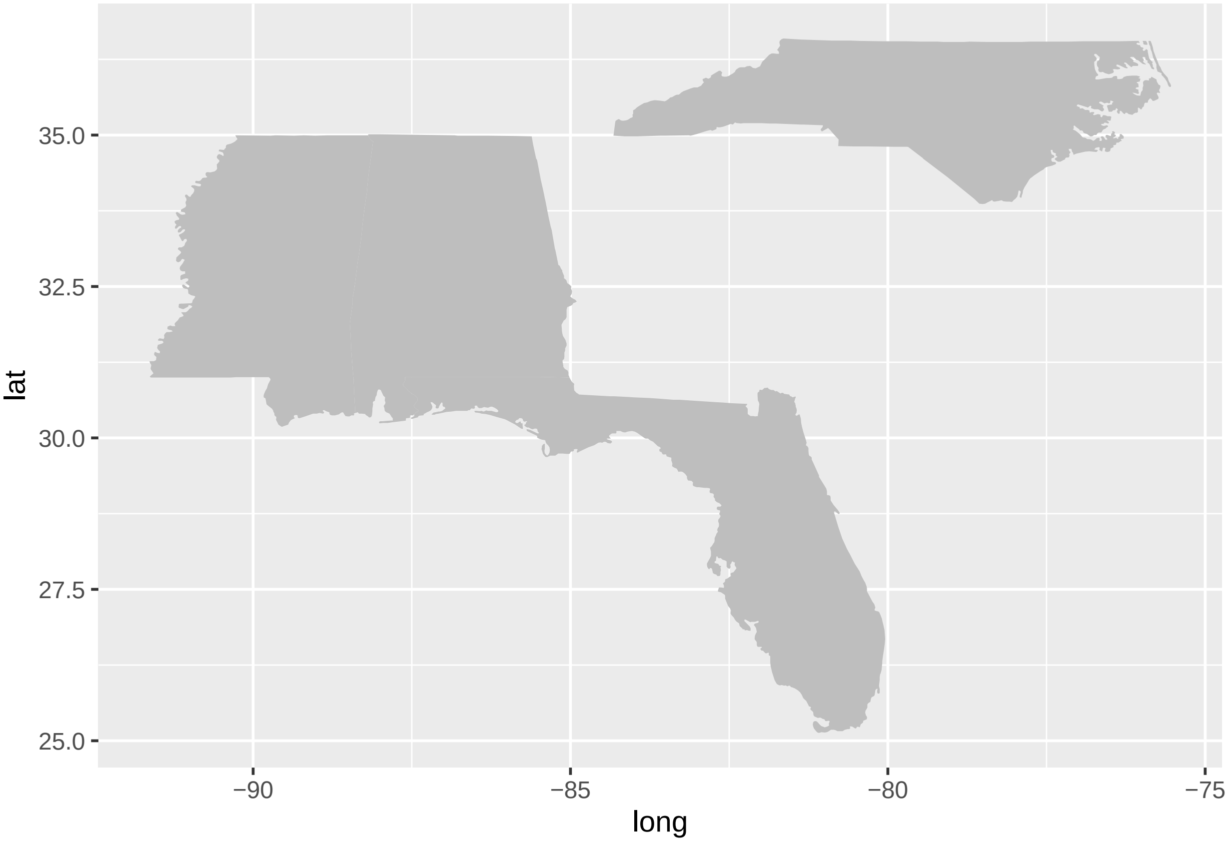
```
## 'data.frame':    15537 obs. of  6 variables:
##  $ long     : num  -87.5 -87.5 -87.5 -87.5 -87.6 ...
##  $ lat      : num  30.4 30.4 30.4 30.3 30.3 ...
##  $ group    : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ order    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ region   : chr  "alabama" "alabama" "alabama" "alabama" ...
##  $ subregion: chr  NA NA NA NA ...
```

The map data are pairs of sequential points, along with an `order` variable. The data are similar to connect the dots pictures you might have drawn as a child. The numbered points specify the order of connection. Here, the points of the map data are specified by `long` (longitude) and `lat` (latitude). The points are connected in order by a `geom_ploygon` layer.

I built the map in this example one layer at a time to show you the steps. When the time comes, you can plot it all in a single code chunk, like the final step of this example. I built the map up from the bottom layer, overlaying other layers on top. I plotted the outline of the 48 states last so that the state boundaries overlay everything else.

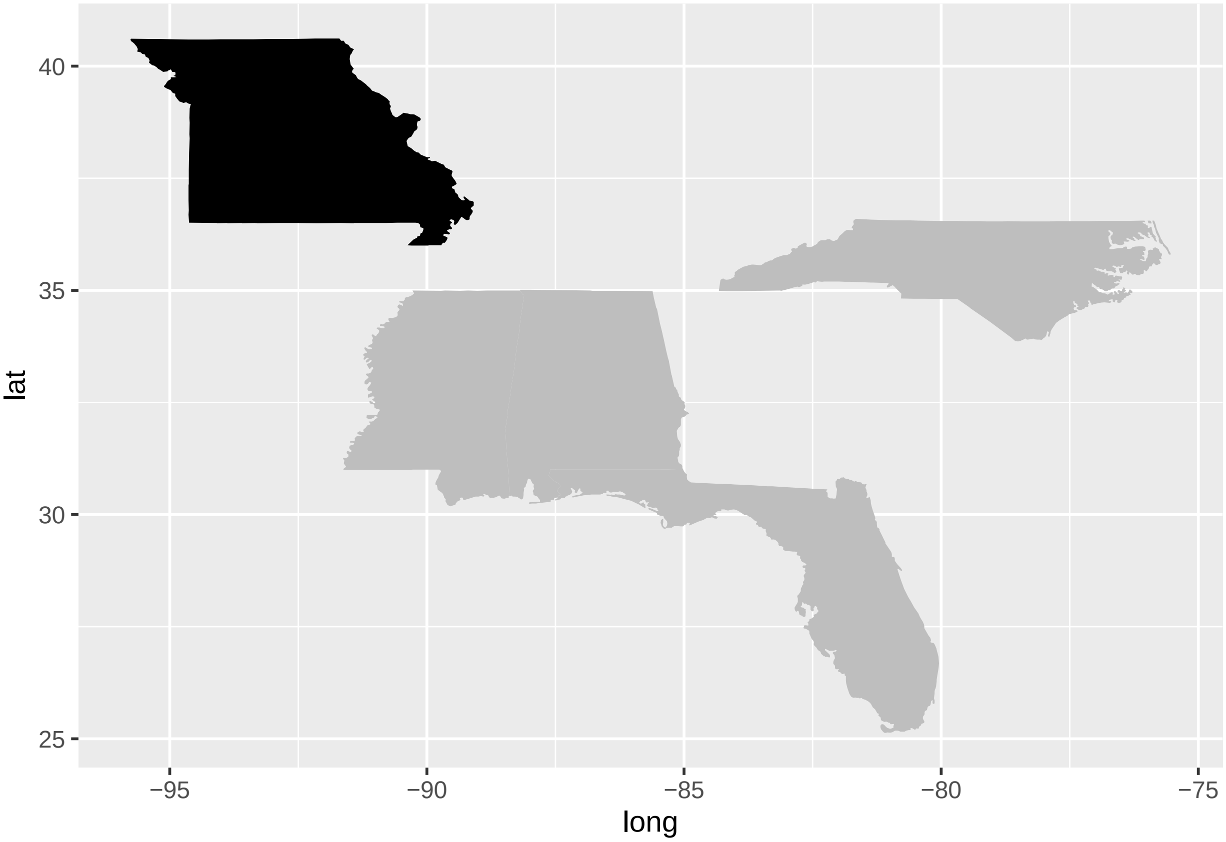
The first layer has the four non-Missouri states. The species of ant was known previously from these four states. The are filled with gray to highlight them without emphasizing them. You can recognize the states but they are distorted. We will fix the distortion at the very end.

```
ggplot() +
  geom_polygon(data = us_other,
               aes(x = long,
                   y = lat,
                   group = group),
               fill = "gray")
```



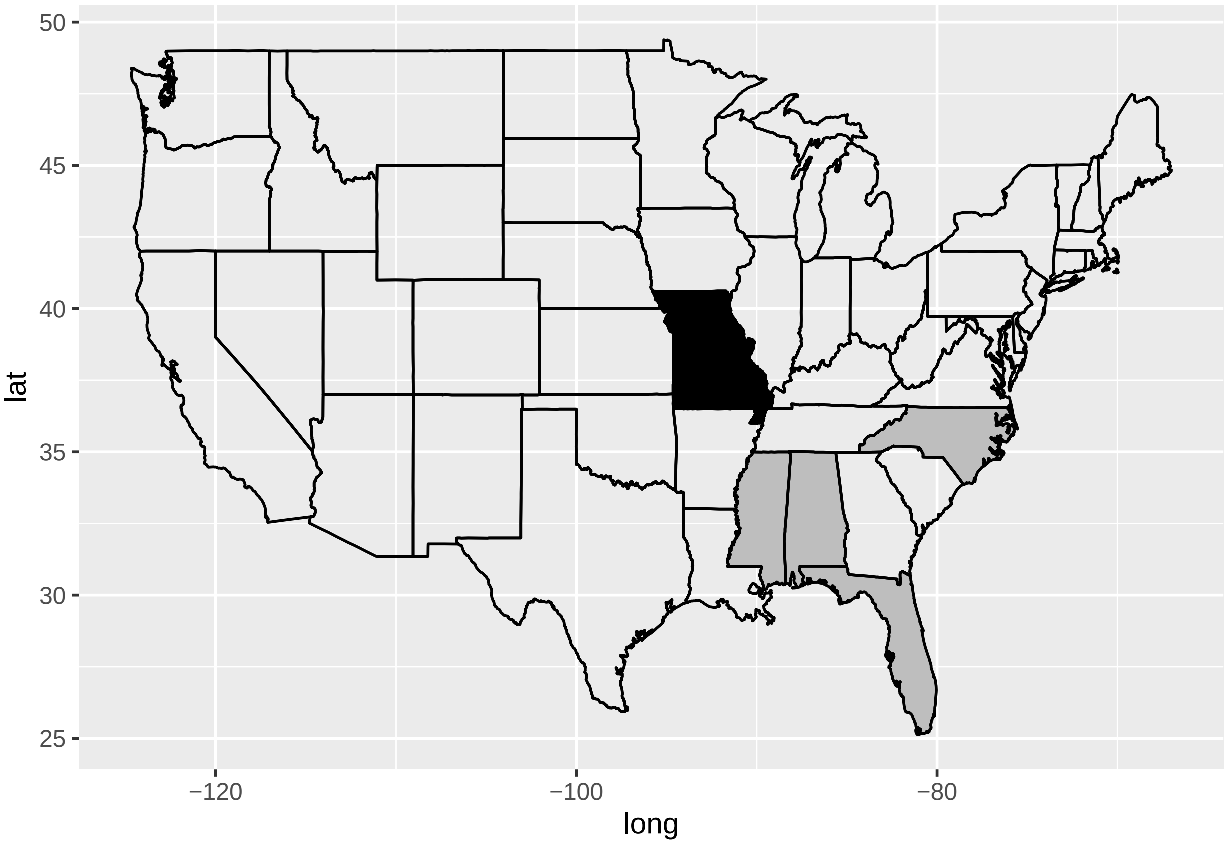
I added Missouri, filled with black to emphasize it over the other four states.

```
ggplot() +
 geom_polygon(data = us_other,
              aes(x = long,
                  y = lat,
                  group = group),
              fill = "gray") +
 geom_polygon(data = us_missouri,
              aes(x = long, y = lat,
                  group = group),
              fill = "black")
```



Then I added the outline of all 48 states.

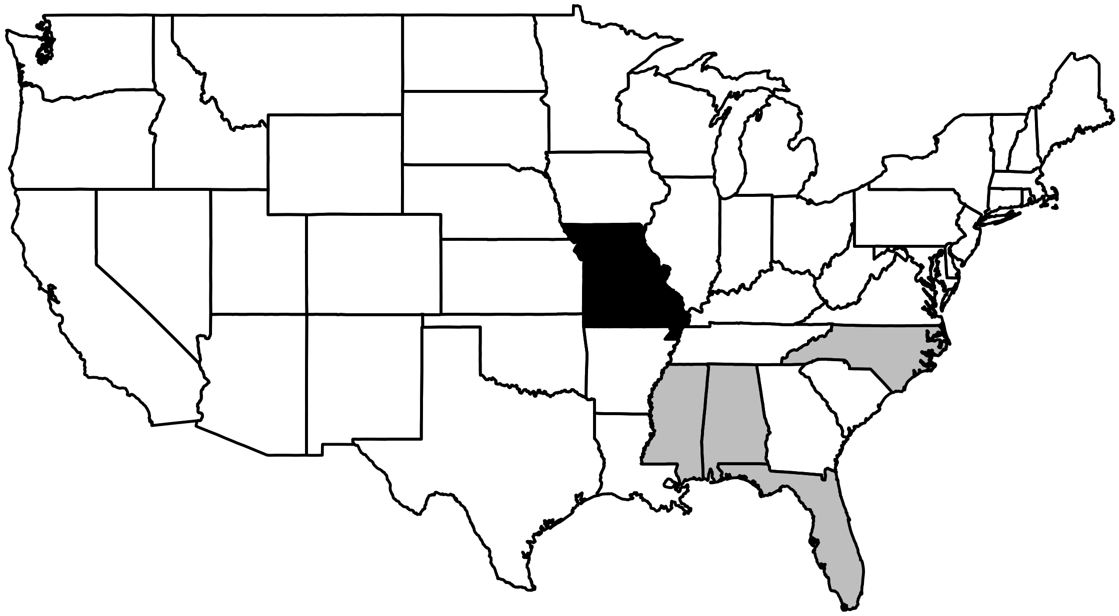
```
ggplot() +
  geom_polygon(data = us_other,
               aes(x = long,
                   y = lat,
                   group = group),
               fill = "gray") +
  geom_polygon(data = us_missouri,
               aes(x = long, y = lat,
                   group = group),
               fill = "black") +
  geom_polygon(data = usa,
               aes(x=long,
                   y = lat,
                   group = group),
               fill = NA,
               color = "black")
```



The map is still distorted and it has the default `theme_gray`. I replaced the default theme with `theme_map()` from the `ggthemes` package. I also added a `coord_fixed` layer with a value of `1.3`, which sets a fixed ratio for longitude and latitude. The longitude is fixed at 1.3 times longer than latitude. Coordinates for maps can be set using more accurate methods but `coord_fixed()` is OK for a map like this. I saved this plot to `us_map` for later plotting with `patchwork`.

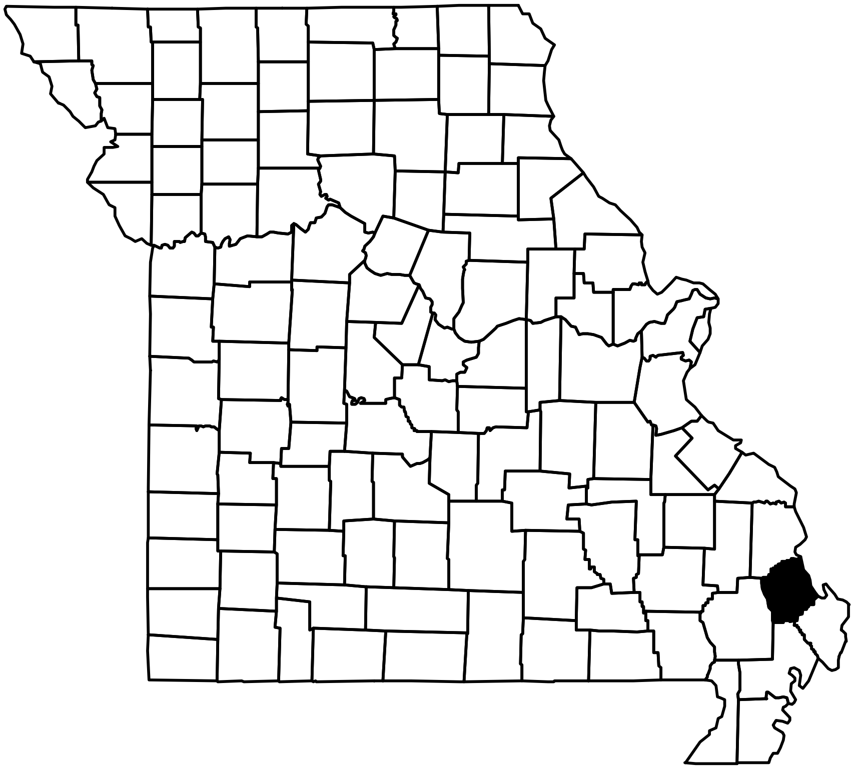
```r
# Add the filled states first so that the black outlines of all
# states are overlaid on top.
us_map <- ggplot() +
  geom_polygon(data = us_other,
               aes(x = long,
                   y = lat,
                   group = group),
               fill = "gray") +
  geom_polygon(data = us_missouri,
               aes(x = long, y = lat,
                   group = group),
               fill = "black") +
  geom_polygon(data = usa,
               aes(x=long,
                   y = lat,
                   group = group),
               fill = NA,
               color = "black") +
  theme_map() +
  coord_fixed(1.3)
```

```
us_map
```



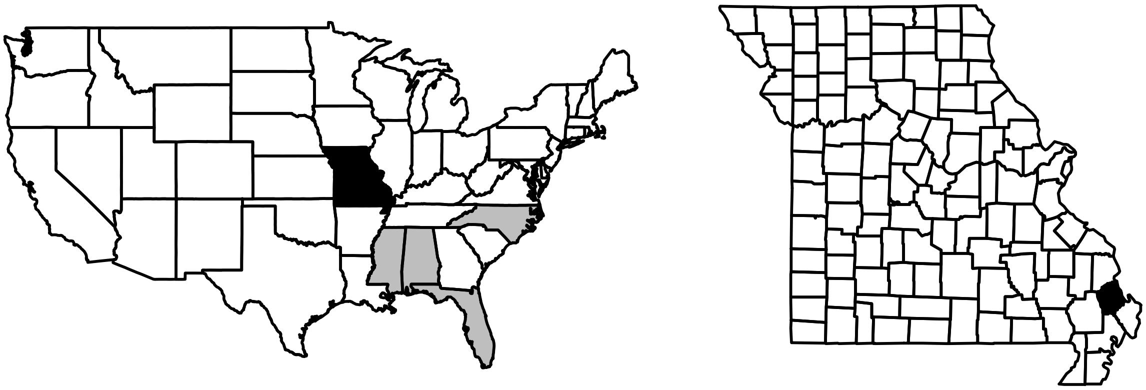
Missouri map I then built the Missouri map, highlighting Scott County to show the county where the ant species was collected for the first time ever. I used the same steps as above. I created two objects, one with the polygons for all Missouri counties and one with just the the Scott County polygon. I add the Scott County layer first, then the outline of all the counties. We will do this all in a single chunk.

```
missouri <- map_data("county", "missouri")
mo_scott <- subset(missouri, subregion %in% c("scott"))

mo_map <- ggplot() +
  geom_polygon(data = missouri, aes(x=long, y = lat, group = group), fill = NA, color = "black") +
  geom_polygon(data = mo_scott, aes(x = long, y = lat, group = group), fill = "black") +
  theme_map() +
  coord_fixed(1.3)

mo_map
```



I now had a `us_map` object and a `mo_map` object. The final step was to put them side-by-side using the `patchwork` package. I used `ncol = 2` but you could also use `nrow = 1`. The `widths` argument specifies how wide each column should be. I tried several values, deciding finally that the left column with the U.S. map should be 1.5 times wider than the right column with the Missouri map.

```
us_map + mo_map + plot_layout(ncol = 2, widths = c(1.5,1))
```



Stage, commit, and push.

Outline maps: your turn.

Your goal is to make an outline map similar to the above example, but with the following requirements.

- The U.S. map should highlight in black the state of your birth.
- The state map should highlight in black the county of your birth.
- The state map should also show in gray two additional counties, chosen based on the initials of your

first, middle and last names, assuming you have a middle name and are not a famous Brazilian football player.

For the non-birth counties, pick two county names that correspond with some combination of initials from your first, middle, and last names. For example, I was born in Missouri. In addition to my birth county (St. Louis), I could choose

- Macon and Schuyler,
- Madison and Taney,
- Shelby and Texas, or
- any other combination of counties that begin with M, S, or T.

If one of your names begins with a letter that does not match a county in your state (e.g., Ellen or Zachary would not match in Missouri), use the second letter of that name (e.g., L or A for Ellen and Zachary).

Wikipedia has lists of counties for each state.

Part 2: Choropleth maps

Choropleth maps are similar to outline maps but the regions are filled based on a statistic, such as average crime rate or per capita disease rate.

Example: Skin cancer mortality in the U.S.

If you haven't already, load these packages.

```
library(sf)
library(tidyverse)
```

First, we're going to import the skin cancer data that will form the basis of the fill colors of map. The data is a series of columns separated by white space. I could use the `read_fwf()` function but we can use the white space as a deliminter in the `read_delim` function.

```
skin_cancer_raw <-
  read_delim(file = here::here("data","skin_cancer.txt"),
             delim = " ",
             skip = 7,
             trim_ws = TRUE)
```

Next, we have to do some data wrangling to add spaces to the two-word state names. We also have to correct a typo on the data (**M**ewYork). The state names have to be corrected because they have to match the state names as spelled in the `fips` data we imported above

```
skin_cancer <- skin_cancer_raw %>%
  filter(State != "Wash,D.C.") %>%
  mutate(State = case_when(
    State == "NorthCarolina" ~ "North Carolina",
    State == "SouthCarolina" ~ "South Carolina",
    State == "NorthDakota" ~ "North Dakota",
    State == "SouthDakota" ~ "South Dakota",
    State == "NewHampshire" ~ "New Hampshire",
    State == "NewJersey" ~ "New Jersey",
    State == "NewMexico" ~ "New Mexico",
    State == "MewYork" ~ "New York", # Data has MewYork typo
    State == "RhodeIsland" ~ "Rhode Island",
    State == "WestVirginia" ~ "West Virginia",
```

```
    TRUE ~ State
  ))
```

We're going to calculate the skin cancer mortality rate relative to the mean rate. Negative numbers indicate states with below average mortality. Positive numbers are states with above average mortality. First, we calculate `mean_mort`, which is the average mortality for all 48 states. We then use `mutate()` to add a `relative_mort` column, substracting the `Mort` column from the `mean_mort` average. We're saving this adjusted data in a new tibble called `skin_cancer_adj`.

```
mean_mort <- mean(skin_cancer$Mort)

skin_cancer_adj <- skin_cancer %>%
  mutate(relative_mort = Mort - mean_mort)
```

Important note: The steps outlined from here through the plotting of the map, except for the `left_join` below, will be exactly the same in the *Your turn* section. You'll have to left join with the Lyme Disease data given to you.

We need a vector of the lower 48 states. We'll use the vector as a filter to remove Alaska, Hawaii, the District of Columbia, and U.S. territories from our shapefile.

```
# Used to trim Alaska, Hawaii, and territories from the shapefile.
lower_48 <- c("Alabama","Arizona","Arkansas","California","Colorado","Connecticut","Delaware","Florida"
```

Next, we use the `st_read` function from the `sf` package to read the shape file. Shape files can contain a wide range of geographic information, including the shapes of countries and states. Shape files are widely used in geographic information systems (GIS).

`Filter` keeps only states in the `states` object that listed in the `lower_48` vector. Any region that does not match is removed. `states_df` is the tibble with the state shapes we will plot.

```
states <- st_read(here::here("data/shape_files", "cb_2017_us_state_500k.shp"),
                  stringsAsFactors = TRUE)
```

```
## Reading layer `cb_2017_us_state_500k' from data source `/Users/goby/Documents/teach/485_repo/SEMO-GA
## Simple feature collection with 56 features and 9 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -179.1489 ymin: -14.5487 xmax: 179.7785 ymax: 71.36516
## geographic CRS: NAD83
```

```
states_df <- states %>%
  dplyr::filter(NAME %in% lower_48)
```

Each state (and U.S. territory) is given a numerical FIPS code. In a moment, we're going to import the list of state FIPS. However, we will need to use the FIPS as factors. The `states_df` tibble has a column called `STATEFP` with the state FIPS already factored. Shown here,

```
str(states_df$STATEFP)
```

```
##  Factor w/ 56 levels "01","02","04",..: 49 14 21 13 46 7 32 31 28 39 ...
```

The `levels()` function returns the levels associated with a factored variable. We'll capture those levels, which we then use to factor the FIPS in the imported data.

```
# Capture the FIPS levels from states_df
# Use to factor the FIPS and state names in the next step.
state_levels <- levels(states_df$STATEFP)
name_levels <- levels(states_df$NAME)
```

Now we import the state FIPS, remove an unneeded column, rename the FIPS column to STATEFP, then use `mutate()` to change the `STATEFP` column to an unordered factor.

Note the use of `dplyr::select(-abbr)`. The `raster` package also has a function called `select()` that can conflict with the `select` function from `dplyr`, one of the `tidyverse` packages. The `dplyr::` part of the code tells R to use `select()` from `dplyr`.

```r
library(tidyverse)
fips <- read_csv(here::here("data", "state_fips.csv")) %>%
  dplyr::select(-abbr) %>%
  rename(STATEFP = FIPS) %>%
  mutate(STATEFP = factor(STATEFP, levels = state_levels))
```

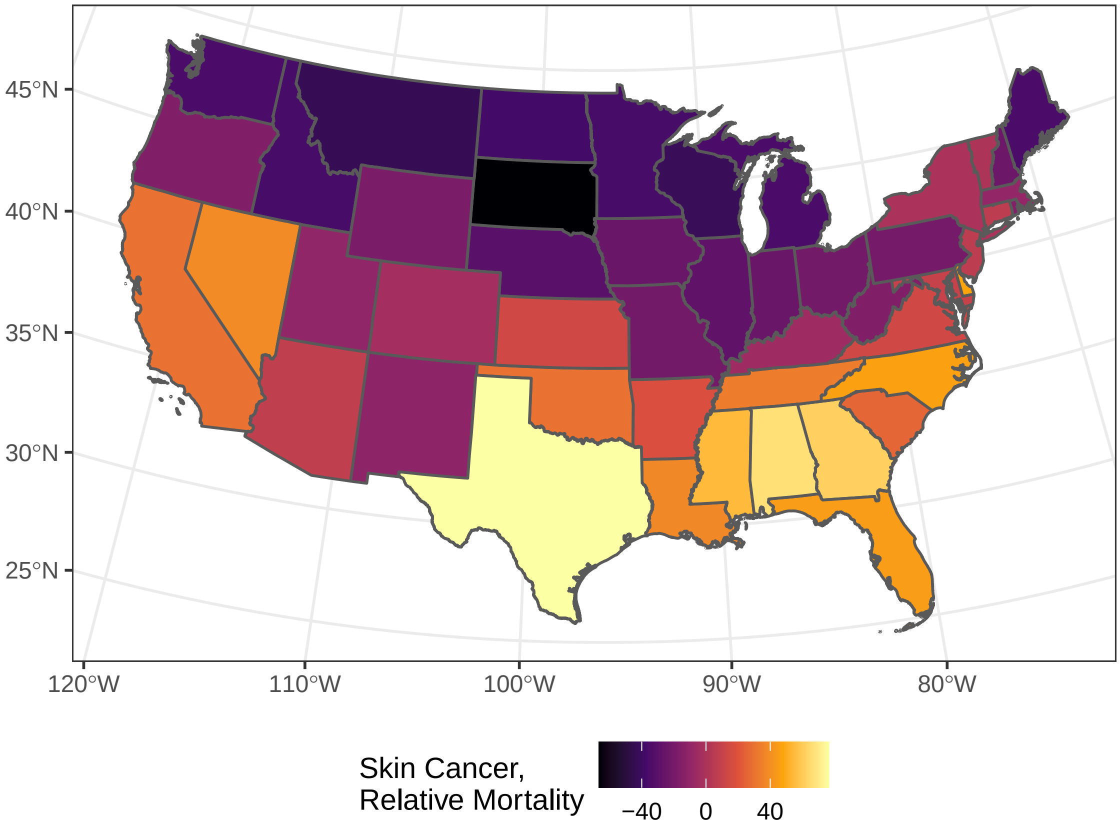
Next, we use two `left_join()` calls to merge the data into a single file. The first left join matches the state names and adds the state FIPS column (`STATEFP`) to our adjusted skin cancer data. That `STATEFP` column then matches the `STATEFP` column in the `states_df` tibble. We now have all of the data in a single tibble.

Note: You'll follow these steps in the *Your turn* section below but you will use the Lyme disease data instead of the skin cancer data.

```r
skin_cancer_adj <- left_join(skin_cancer_adj, fips)

states_df <- left_join(states_df, skin_cancer_adj)
```

That was fair amount of work but we're now ready to plot the map. Plotting uses a new geom and a new layertype from `ggplot`. `geom_sf` is a geom that knows how to plot sf data, which is one of the classes of our `states_df` tibble (run `class(states_df)` from the console). You've seen the viridis scales before. We're using the continuous scale for the relative mortality rate, with the inferno color palette. The `coord_sf` layer will give a slight curve to the plot to mimic the curvature of Earth. `theme_bw()` is one of the basic themes that comes with `ggplot2`. You could also use `theme_map()` from the `ggthemes` package. Finally, the `theme(legend.position = "bottom")` puts the legend horizontally at the bottom of the map rather than vertically on the side. You can try both to see which you prefer.

```r
ggplot(states_df) +
  geom_sf(aes(fill = relative_mort)) +
  scale_fill_viridis_c(name = "Skin Cancer,\nRelative Mortality",
                       option = "inferno") +
  coord_sf(crs = st_crs(5070)) +
  theme_bw() +
  theme(legend.position = "bottom")
```



The southern states have a skin cancer mortality higher than average while the northern states have a lower than average mortality rate.

Choropleth map: your turn

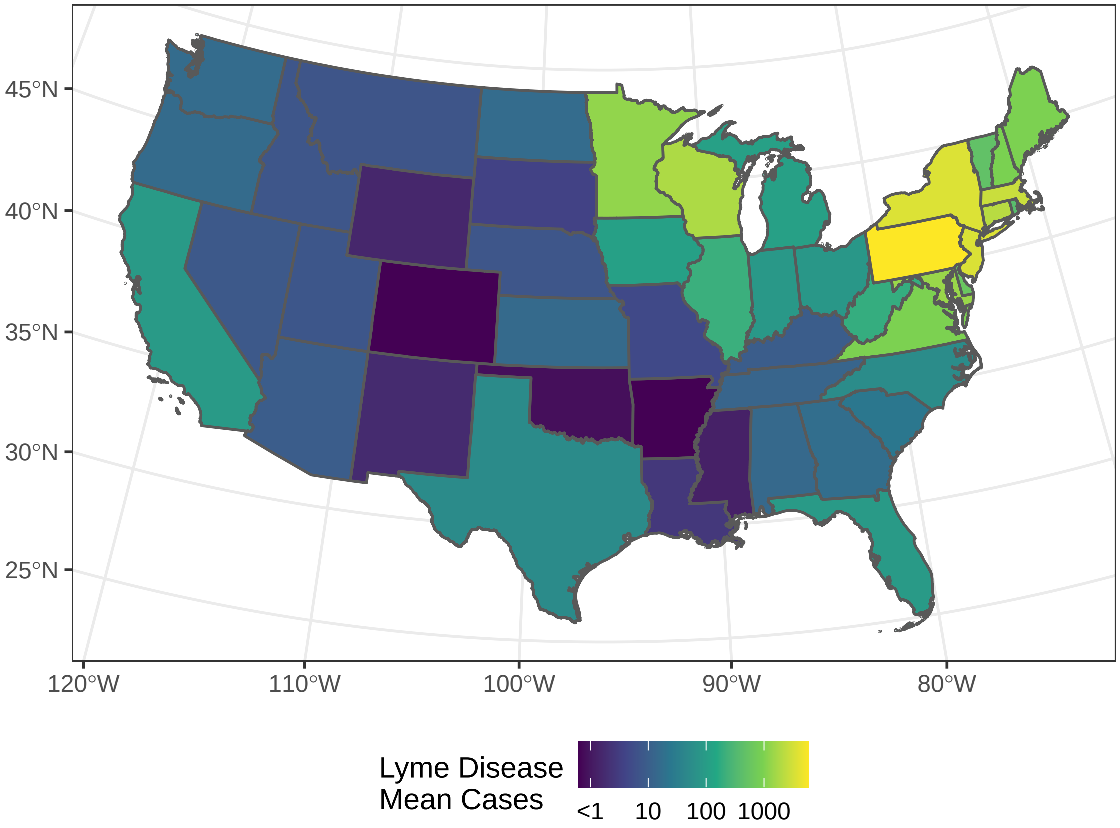
Make a choropleth map for lyme disease. The data have the number of cases of lyme disease reported to the Centers for Disease Control between 2007-2017. The year 2017 has two columns, confirmed and probable. We'll treat the probables as confirmed.

Do the following steps. You only need to do these steps. As long as you didn't quit RStudio and close your notebook, you do not need to repeat all of the steps from above to import the shape files, etc.. You just need to wrangle the data needed to produce the fill color and plot the map.

- Import the csv data. The spreadsheet uses `-` for missing data, which you'll have to convert to `NA` during import. You may have to do one or two other things during import, too. *Inspect your data!*
- Create a new column called `2017` that adds together `2017_Confirmed` and `2017_Probable`.
- Remove the `2017_Confirmed` and `2017_Probable` columns. Once added together, you don't need them and they'll get in the way of the next step.
- The data are not tidy so you will need to `pivot_longer` to put the years into a single column. Use appropriate names for `names_to` and `values_to` arguments.
- `group_by` state and use `summarize()` to calculate mean number of each cases, and then convert the mean to the log (base 10) using the `log10()` function. This converts the values to a reasonable scale for display.
- Use `left_join` as you did above to merge the `fips` data frame to your lyme disease data. Use `left_join` again to merge your lyme data to the `states_df` data.

- Plot your choropleth map using the same `ggplot` code for the cancer map but
    - Change `aes(file = ...)` for `geom_sf` to the column with the log of your mean lyme cases.
    - Change the name of the scale to something appropriate. You can play with other viridis palatte options or even try other continuous scales if you want.
    - For the `scale_fill_viris_c()` layer, I suggest adding `labels = c("<1", "10", "100", "1000", "5000")` as an argument. This makes it easier to match the colors to actual values rather than the log values.

If successful, your map should look something like this:



Part 3: Dot distribution maps

Example: Distribution of ***Gigantopithecus*** and ***Ursus***

Dot distribution maps show locations where a species has been sampled. This example is based on a publication by Lozier et al. 2009. They used ecological niche modeling to predict the distribution of Bigfoot. Srsly.

The Bigfoot data were used to make some points about potential pitfalls of the technique, while also showing how well the "niche" of Bigfoot matches the "niche" of black bears *Ursus americanus*.

The bigfoot data were obtained from the Bigfoot Field Researchers Organization. Srsly. Only Class A sightings were used. (I still have *some* standards.) The black bear data were obtained from the Global Biodiversity Information Facility.

If you haven't already, right-click and save bigfoot.csv and bears.csv to your `data` folder.

This example uses the following packages. If you loaded them for Part 1, then you do not have to load them now unless you quit and have now returned to continue working.

```
library(raster) # Load before tidyverse.
library(tidyverse)
library(ggthemes)
library(ggmap)
```

First, define some global vars that restricts the data to minimum and maximum latitudes and longitudes. We will also set the seed used by the random number generator.

Important: Software like R can generate "random" numbers but they are pseudorandom numbers that only approximate random numbers. Pseudorandom numbers are generated from a "seed", usually based on the computer's internal clock. However, if the seed is set to a fixed number, the "random" can be accurately replicated.

Setting the seed is necessary for reproducibility. Here, I used R's `setseed()` function so that my "random" sampling of bigfoot data from a much larger data set is reproducible. I used a smaller subset of data to keep the map from being too overcrowded with points. Any number can be used for the seed. I used a childhood phone number. If you use the same number, you will get the same results as me.

```
# Global vars ------------------------------------------------------------

# Restrict the map to the western US.
min_long <- -126
max_long <- -104
min_lat  <-   31
max_lat  <-   50

set.seed(9618973)
```

Load the Bigfoot data. The Bigfoot data has the longitude and latitude for "observations" across North America. The data set is large so I used the `sample()` function to "randomly" sample 300 records. I will get the same 300 records with every sample, as long as I do not change the seed. If I change the seed, then the 300 records will not all be the same.

I filtered the data to restrict the distribution to the longitude and latitude variables defined above. I sampled 300 records, changed the name to the **non-scientific** genus *Gigantopithecus*, and then selected only the name, longitude and latitude columns. The `sample` function is affected by `setseed`.

Note: the file paths assume that you have a `data` folder at the same level as your `Rproj` folder, and that the csv files are in that folder, per previous instructions.

Note: You may have to use `dplyr::select()` for the same reason you had to use `here::here()`: package conflicts.

```
# Bigfoot data ------------------------------------------------------------

bigfoot <- read_csv(here::here("data", "bigfoot.csv"))

bigfoot <- bigfoot %>%
  filter(long >= min_long & long <= max_long,
         lat >= min_lat & lat <= max_lat) %>%
  sample_n(300) %>%
  mutate(name = "Gigantopithecus") %>%
  dplyr::select(name, long, lat)
```

Next, read in the `bears.csv` data. I previously wrangled the data so it has only the lontitude and latitude in the range defined above.

```
# Bear data ---------------------------------------------------------------

bears <- read_csv(here::here("data", "bears.csv"))

both_species <- bind_rows(bigfoot, bears)

head(both_species)
```

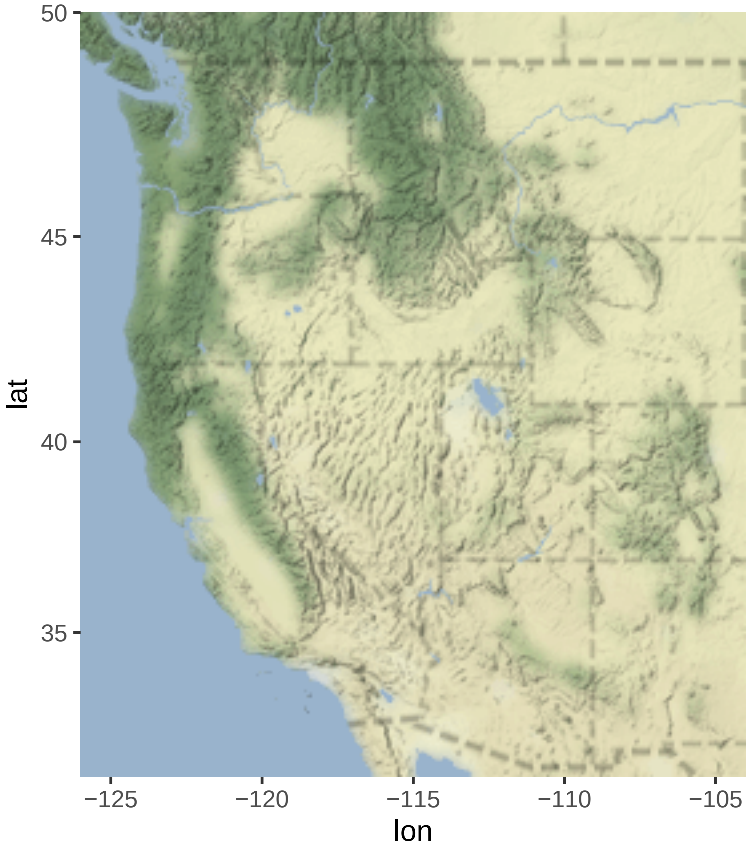
```
## # A tibble: 6 x 3
##   name             long   lat
##   <chr>           <dbl> <dbl>
## 1 Gigantopithecus -123.  42.4
## 2 Gigantopithecus -122.  45.3
## 3 Gigantopithecus -122.  40.8
## 4 Gigantopithecus -122.  40.8
## 5 Gigantopithecus -122.  45.1
## 6 Gigantopithecus -122.  40.9
```

```
tail(both_species)
```

```
## # A tibble: 6 x 3
##   name              long   lat
##   <chr>            <dbl> <dbl>
## 1 Ursus americanus -126.  49.2
## 2 Ursus americanus -110.  44.9
## 3 Ursus americanus -119.  34.9
## 4 Ursus americanus -114.  34.6
## 5 Ursus americanus -123.  43.4
## 6 Ursus americanus -125.  49.9
```

This code gets the terrain map from Stamen Maps (similar to Google Maps but does not require registration). This will take a few minutes. `zoom` defines the resolution. The higher the value, the greater the resolution. Whenever you use this technique, always start with a lower value like 3 or 4. Use a higher value like 7 when you make your final figure. (The actual final value depends on how large of a geographic area that you need to cover.)

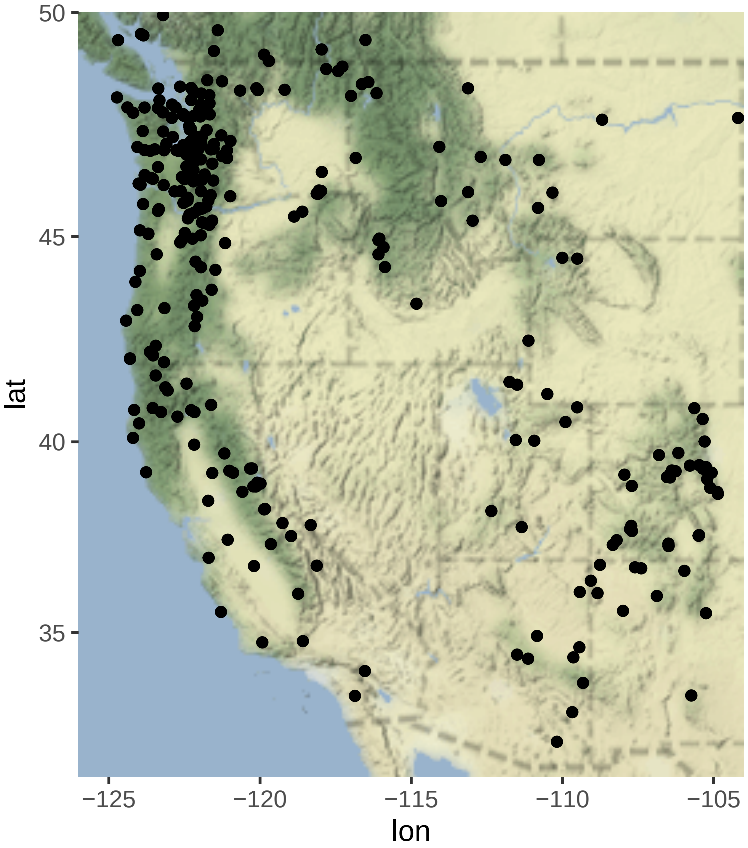
```
base = get_stamenmap(bbox = c(min_long,
                              min_lat,
                              max_long,
                              max_lat),
                     zoom = 4,
                     maptype = "terrain-background")



ggmap(base)
```



Dark green shows forested areas. The grayer colors show higher elevations.

`ggmap` uses `ggplot2` to plot this layer, so we can store the result into an object and add to it using familiar geoms. The first layer to add is the Bigfoot data.

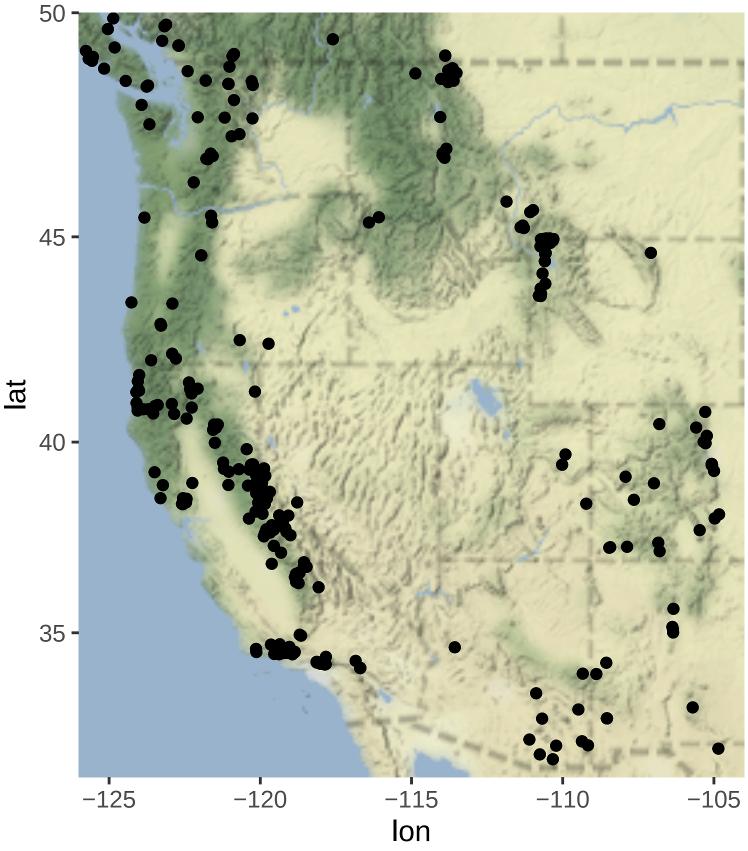
```
base <- ggmap(base)

base + geom_point(data = bigfoot,
                  aes(x = long,
                      y = lat))
```



Bigfoot clearly seems to prefer the forested areas, especially around Seattle. In case you are wondering, most sightings of Bigfoot were recorded before marijuana was legalized in Washington state.

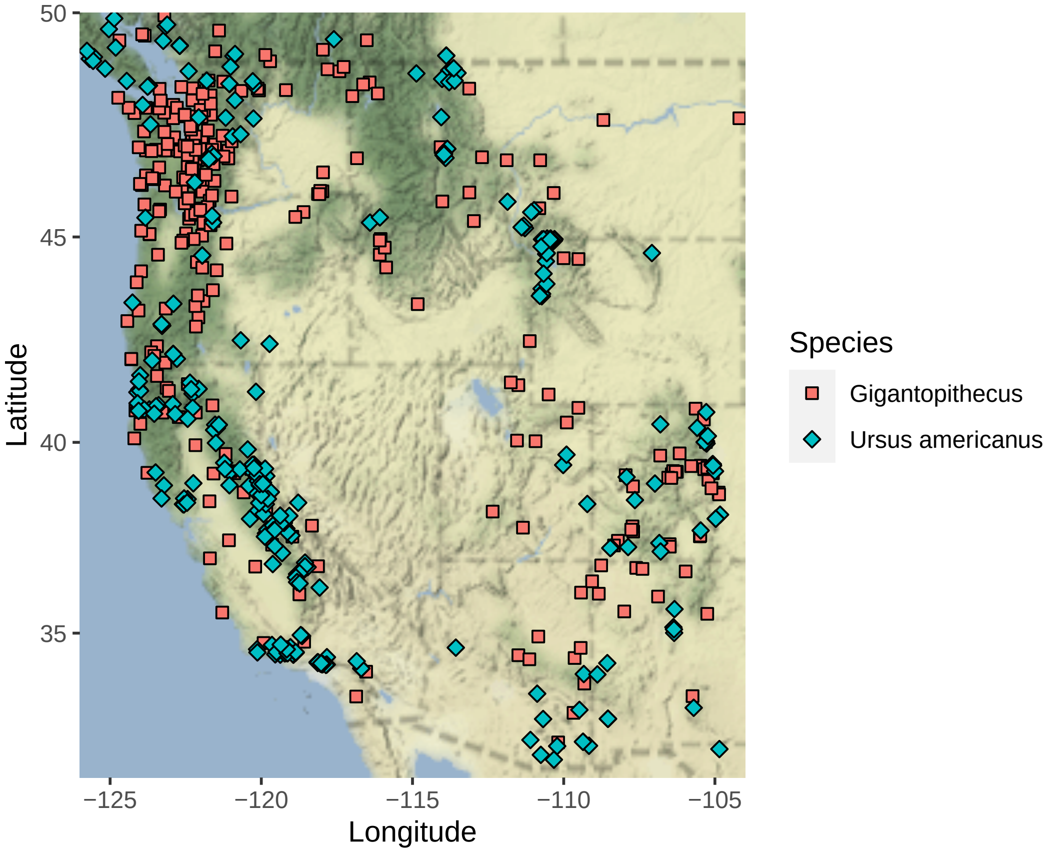
What about black bears?

```
base + geom_point(data = bears,
                  aes(x = long,
                      y = lat))
```



Black bears also seem to prefer the woods. Hmmm. . . . The final plot includes both species, from the `species` data frame. I used `scale_shape_manual` to choose fillable shapes. The shapes and fill color are matched by the `aes` aesthetic to each species. I ensured only a single legend was used by setting both shape and color to "Species" in the `labs` layer. I increased the point size slightly, outsize of `aes`.

```
base + geom_point(data = both_species,
                  aes(x = long,
                      y = lat,
                      shape = name,
                      fill = name),
                  size = 2) +
  scale_shape_manual(values = c(22:23)) +
  labs(x = "Longitude",
       y = "Latitude",
       fill = "Species",
       shape = "Species")
```



Distribution map: your turn

Recreate the Bigfoot and Black Bear distribution map but change the `setseed` number to your last seven digits of your S0 number. The seed determines the random sample of Bigfoot sightings but the bear distribution should not change.

Part 4: Bathymetry

Bathymetric maps are maps that show the changes of depth in a body of water. We will use bathymetry in combination with dot distribution maps. For this example, we will plot the distribution of earthquakes near the island nation of Fiji.

Example: Fiji earthquakes

Fiji is a group of islands located on the western edge of the Pacific Plate.. The edges of the Pacific plate has lots of seismic activity, in the form of volcanos and earthquakes. Earthquakes that have occurred around Fiji since 1964 are recorded in the `quakes` data set, one of the data sets included with R. The `quakes` data set include longitude, latitude, magnitude (strength), and depth.

The amount of data needed to plot bathymetry is large, so some plots will take several seconds to draw.

This part uses the `tidyverse` and `marmap` libraries. If you loaded them for Part 1, you do not need to load them now unless you quit previously and have returned to finish.

```
library(tidyverse)
library(marmap)
```

As above, I defined global variables with the longitude and latitude range for the map. The bathymetric data has depth in feet but the `quakes` data has depths in kilometers, so I converted quake depth to feet.

```
min_long <- -170
max_long <- 164
min_lat  <- -42
max_lat  <- -8

# This converts depth in km to depth in feet.
# Necessary because bathymetry data will be in feet
earthquakes <- quakes %>%
   mutate(depth = depth * 3280.84)
```

This chunk obtains the data directly from NOAA. The `getNOAA.bathy` function retrieves the data from a NOAA website. The `antimeridian = TRUE` argument is necessary here because Fiji sits on the 180° meridian where the longitude changes from East to West. The `keep = TRUE` argument causes the function to save the data to a local csv file. In the future, when you run the code, the function will load the data from the file rather than download it again. If you delete the file, the data will be downloaded and saved again.

```
# Will auto read the saved file if present.

fiji <- getNOAA.bathy(lon1 = min_long,
                      lon2 = max_long,
                      lat1 = min_lat,
                      lat2 = max_lat,
                      antimeridian = TRUE,
                      keep = TRUE)
```

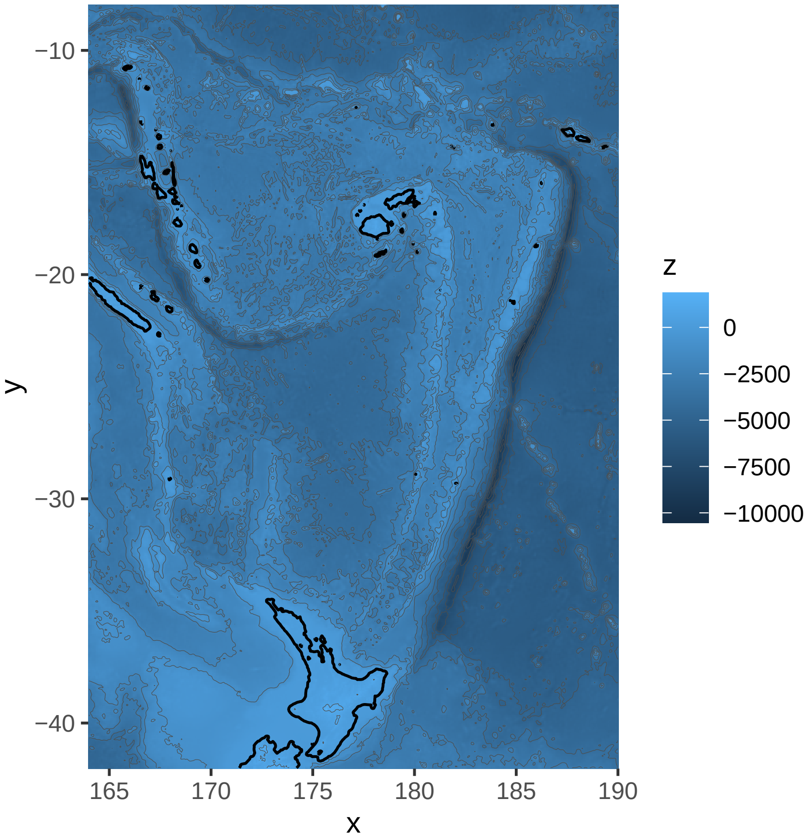
The `marmap` package includes an `autoplot.bathy()` function to automatically plot bathymetric data with ggplot2, using reasonable default settings. The `geom = c("raster", "contour")` tells `ggplot` to use `geom_raster` and `geom_contour`. The `size = 0.1` argument specifies the thickness of the contour lines. Try tweaking this value a little to see how changes affect the appearance of the plot.

Ignore the warning about ignoring the `size` parameter. It is not actually ignored.

```
# Could also just use autoplot() without the .bathy extension.
autoplot.bathy(fiji,
         geom = c("raster", "contour"),
         size = 0.1,
         na.rm = TRUE)
```

```
## Warning: Ignoring unknown parameters: size
```

```
## Warning: Raster pixels are placed at uneven horizontal intervals and will be
## shifted. Consider using geom_tile() instead.
```

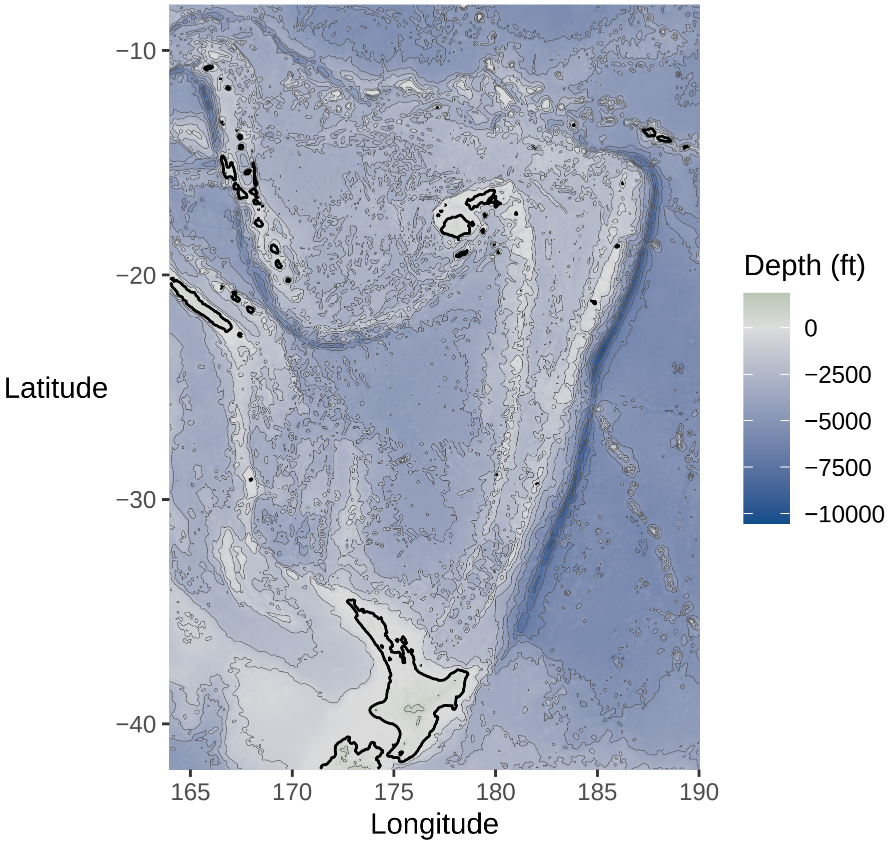


Very oceanic. I thought the default color gradient was too dark so I used a custom gradient by adding a `scale_fill_gradient2` layer. The `low` color is used for the greatest depths. I also labeled the x- and y-axes. **Note:** I used `theme(axis.title.y = ...)` to rotate the y-axis label. `vjust = 0.5` centers the label vertically on the axis.

```
# Using autoplot, which "knows" to use autoplot.bathy
autoplot(fiji,
         geom = c("raster","contour"),
         size = 0.1,
         na.rm = TRUE) +
  scale_fill_gradient2(low = "dodgerblue4",
                       mid = "gainsboro",
                       high = "darkgreen",
                       name = "Depth (ft)") +
  labs(x = "Longitude",
       y = "Latitude") +
  theme(axis.title.y = element_text(angle = 0,
                                    vjust = 0.5))
```

```
## Warning: Ignoring unknown parameters: size
```

```
## Warning: Raster pixels are placed at uneven horizontal intervals and will be
## shifted. Consider using geom_tile() instead.
```



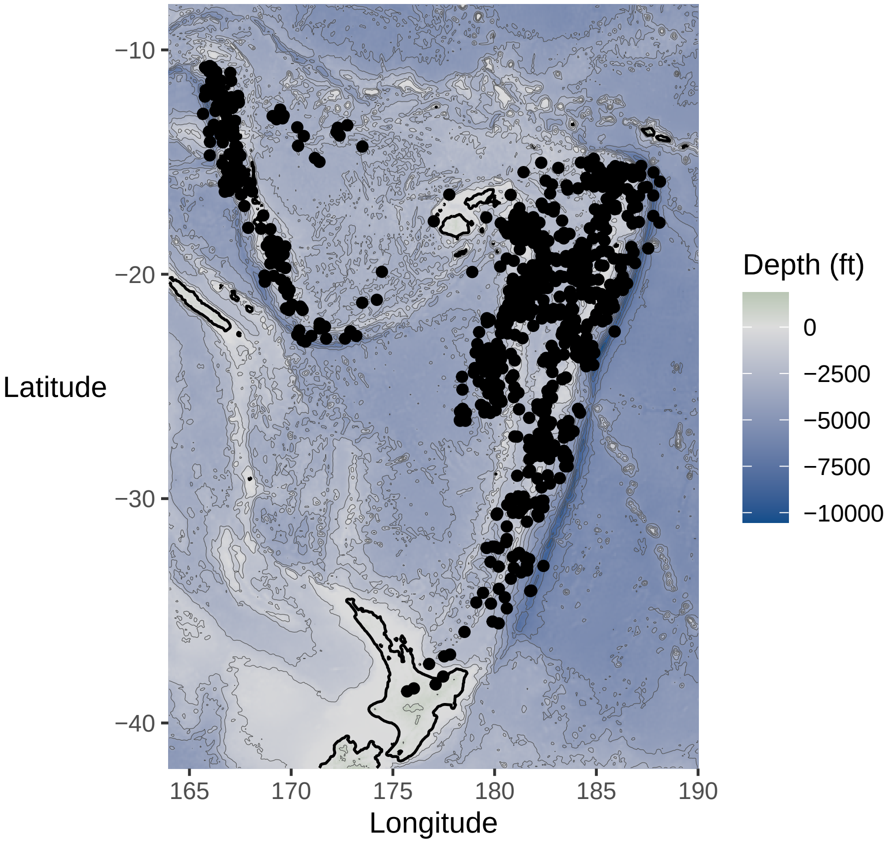
That's a good start. I saved the `autoplot` result to an object called `base_map`, and then add the Fiji earthquakes data using familiar `ggplot` layers. I used `geom_point` to plot the location of each seismic event.

```
base_map <- autoplot(fiji,
                     geom = c("raster","contour"),
                     size = 0.1,
                     na.rm = TRUE) +
  scale_fill_gradient2(low = "dodgerblue4",
                       mid = "gainsboro",
                       high = "darkgreen",
                       name = "Depth (ft)") +
  labs(x = "Longitude",
       y = "Latitude") +
  theme(axis.title.y = element_text(angle = 0,
                                    vjust = 0.5))
```

```
## Warning: Ignoring unknown parameters: size
```

```
base_map +
  geom_point(data = earthquakes,
             aes(x = long,
                 y = lat))
```

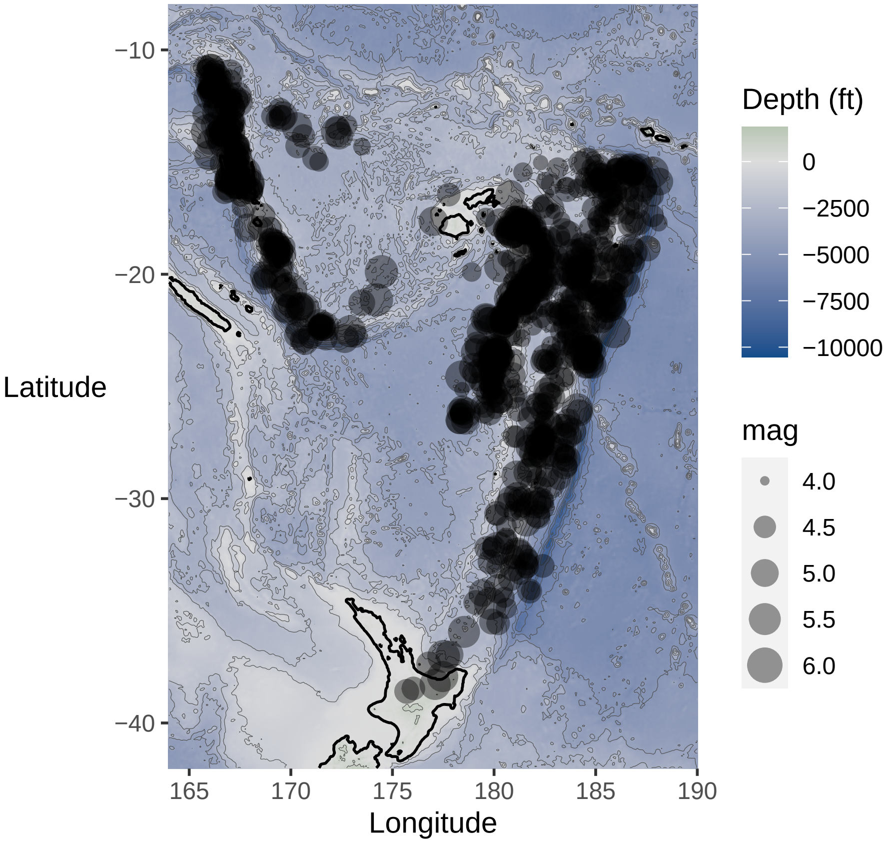
```
## Warning: Raster pixels are placed at uneven horizontal intervals and will be
## shifted. Consider using geom_tile() instead.
```



That's a lot of earthquakes. The points are rather crowded. I also want to adjust the size of the points to reflect the magnitude of the each quake. I used the `alpha` argument to make the points more transparent.

```
base_map +
  geom_point(data = earthquakes,
             aes(x = long,
                 y = lat,
                 size = mag),
             alpha = 0.4)
```

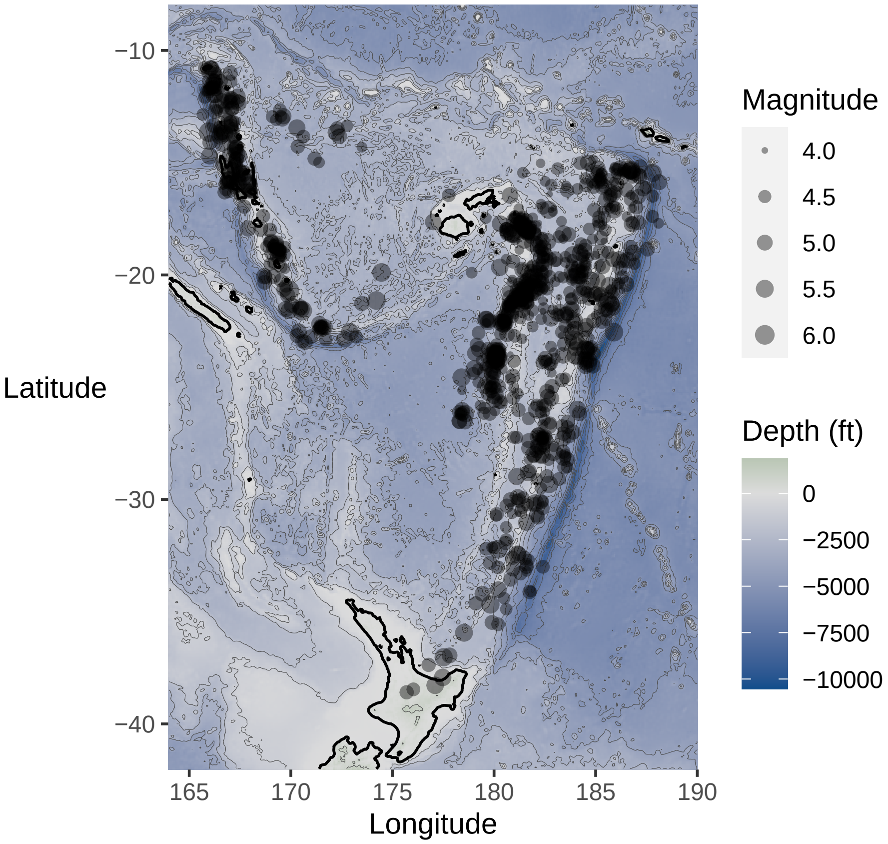
```
## Warning: Raster pixels are placed at uneven horizontal intervals and will be
## shifted. Consider using geom_tile() instead.
```



Hmm. I thought the points were too large, but I still wanted the point size to reflect magnitude. I used the `scale_size_continuous` layer to set the minimum and maximum point sizes used for magnitude. I added the `name` argument to name this legend.

```
base_map +
  geom_point(data = earthquakes,
             aes(x = long,
                 y = lat,
                 size = mag),
             alpha = 0.4) +
  scale_size_continuous(range = c(0.5,3), name = "Magnitude")
```

```
## Warning: Raster pixels are placed at uneven horizontal intervals and will be
## shifted. Consider using geom_tile() instead.
```



That's better. It's easy to see where the concentration of earthquakes is highest, without overwhelming the plot.

Stage, commit, push.

Bathymetry: now you try it

- This part uses the `tidyverse` and `marmap` libraries. You do not need to load them unless you quit and have returned to finish up the assignment.
- Set the following global variables:
    - `min_long`: -90
    - `max_long`: -58
    - `min_lat`: 8
    - `max_lat`: 28

Load `blennies.csv` into a `blennies` object. Use `getNOAA.bathy()` function like we did in the Fiji earthquakes example to get the bathymetric data for the Caribbean Sea.

- Set `antimeridian` to FALSE.
- Set `keep = TRUE` to that the data are saved after the first run.
- Save the results to a `carib_sea` object.

Use `autoplot` to plot the base bathymetric map. Add a `labs()` layer to label the x- and y- axes as "Longitude" and "Latitude", respectively. Replace the `scale_fill_gradient2` layer with this layer.

```
scale_fill_etopo(guide = FALSE)
```

Once you are happy with the results, save the base map to a `base_map` object.

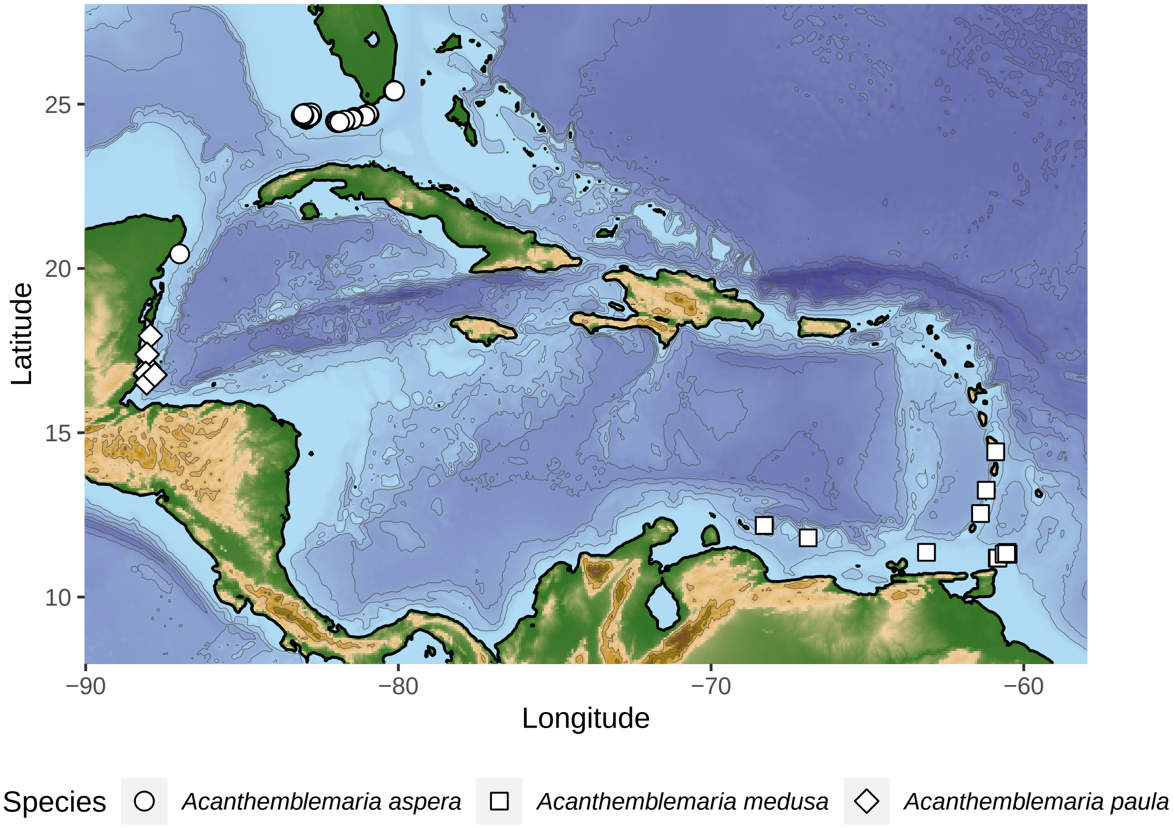
Add the blennies

Add a point layer to show the distribution of the blennies, with these parameters.

- Map shape to name in the aesthetics layer.
- Use fillable shapes 21-23.
- Specify a color for the fill. I used white. You can use other colors but be sure the colors contrast well with the background.
- Increase the point size a bit. Try a range between 2-4. You can use decimal point sizes, like 2.8 or 3.1. Find a value that you think helps the points to stand out without overwhelming the plot.

  Note: for the best sense of the final point size, click on the small "Show in New Window" button just above the top right of the plot in your notebook. The points tend to look a little large in the small plots in your notebook.
- Change the legend name to "Species".
- Add `theme(legend.position = "bottom")` to put the legend below the map.
- Figure out how to add a theme element to italicize the species names in the legend.

Your final plot should look like this, although your point size and fill colors may vary.



Stage, commit, push.

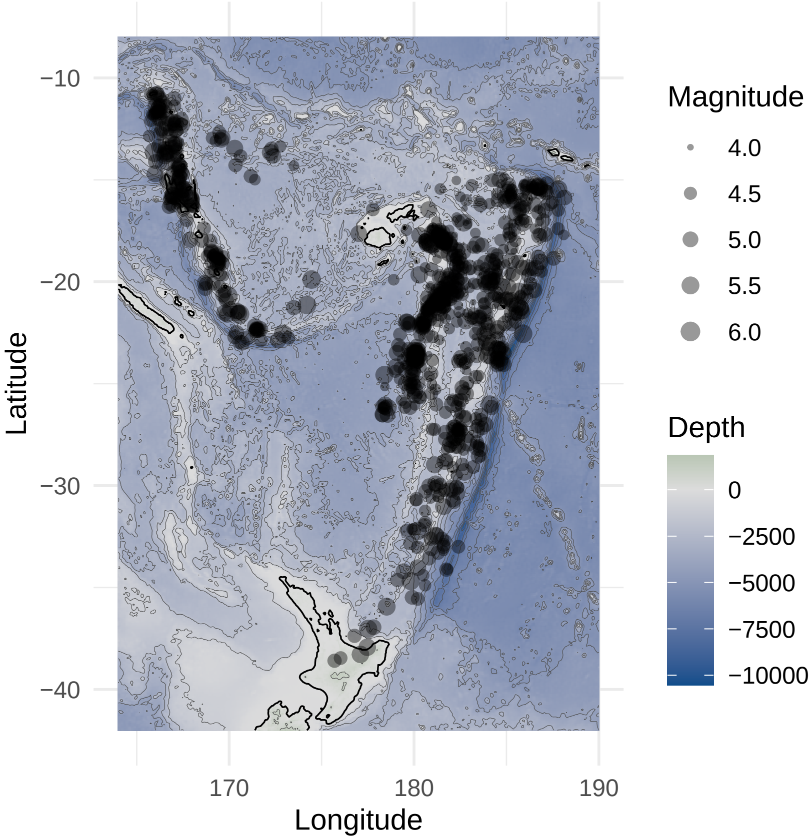
Full code for the Fiji Quakes

This is for reference only. You do not have to run this code.

This code builds the Fiji earthquakes plot from scratch using `ggplot2`. The `autoplot()` function of `marmap` works well but building the map up by individual layers gives you more control of the details.

The bathymetry data must be converted to a dataframe object, which is done here using `fortify`, a function in `ggplot2`. In this case, `fortify` creates an `x` column with longitude, a `y` column with latitude, and a `z` column with depth. Depth values are negative to show depth below the surface (z = 0).

```r
fiji <- getNOAA.bathy(lon1 = -170,
                      lon2 = 164,
                      lat1 = -42,
                      lat2 = -8,
                      antimeridian = TRUE,
                      keep = TRUE)

# Convert the bathymetry data to a data frame.
fiji_df = fortify(fiji)

ggplot(fiji_df,
       aes(x = x,
           y = y,
           z = z,
           fill = z)) +
  geom_tile() +
  geom_contour(data = NULL, lwd = 0.1, color = "gray30") +
  geom_contour(colour = "black",
               linetype = "solid",
               size = 0.3,
               breaks = 0, alpha = 1) +
  scale_fill_gradient2(low = "dodgerblue4",
                       mid = "gainsboro",
                       high = "darkgreen", name = "Depth") +
  geom_point(data = earthquakes,
             inherit.aes = FALSE,
             aes(x = long,
                 y = lat,
                 size = mag),
             alpha = 0.4) +
  coord_quickmap() +
  theme_minimal() +
  scale_size_continuous(range = c(0.5,3)) +
  labs(size = "Magnitude",
       color = "Magnitude",
       x = "Longitude",
       y = "Latitude")
```


Magnitude
4.0
4.5
5.0
5.5
6.0
Depth
0
−2500
−5000
−7500
−10000
−10
−20
−30
−40
Latitude
170
180
190
Longitude

---

et Vóila