# 11: Mapping

Water Data Analytics | Kateri Salk

Spring 2022

## Lesson Objectives

1. Define the basic components of spatial mapping
2. Create maps of water quality data in R
3. Analyze and communicate the findings of spatial analysis using mapping tools

## Opening Discussion

What are some examples of spatial data in the aquatic sciences? Why might you want to map these?

## Spatial visualization

The geometry of a given spatial data point consists of coordinates in 2-, 3-, or 4-dimensional space. These dimensions are:

- **x**: longitude (required)
- **y**: latitude (required)
- **z**: elevation (optional)
- **m**: measurement (optional)

A feature (object) most often falls into one of these three categories (more complex examples exist; see the `sf` package documentation for more):

- **Point**
- **Line**
- **Polygon**

## Mapping in R

Conducting spatial visualization in R presents several benefits:

1. R is an open-source software, making code and output accessible without a software license. Closed-source software such as ArcGIS offers more complex functionality, but for many purposes we can use R to the same effect.
2. Community-sourced packages are improving the functionality of R's spatial capabilities all the time.
3. Coding in R replaces traditional click-based programming in ArcGIS, moving toward reproducible workflow and data analysis pipelines.
4. Spatial tools in R are integrated with other statistics, visualization, and data science tools that you may already use.

We will be using the `sf` ("simple features") package today. A helpful vignette for this package can be found here: https://r-spatial.github.io/sf/articles/sf1.html. We will also be using `maps`, which includes several useful layers of political geography around the world. We will be using their `map` function to plot outlines of U.S. states.

## Session Set Up

```r
getwd()
```

```
## [1] "/Users/katerisalk/Box Sync/Courses/Water Data Analytics/Lessons"
```

```r
library(tidyverse)
library(lubridate)
library(cowplot)
library(LAGOSNE)
# install.packages("sf")
library(sf)
# install.packages("maps")
library(maps)
# install.packages("gganimate")
library(gganimate)

theme_set(theme_classic())
options(scipen = 4)

# Load LAGOSNE data into R session
LAGOSdata <- lagosne_load()
```

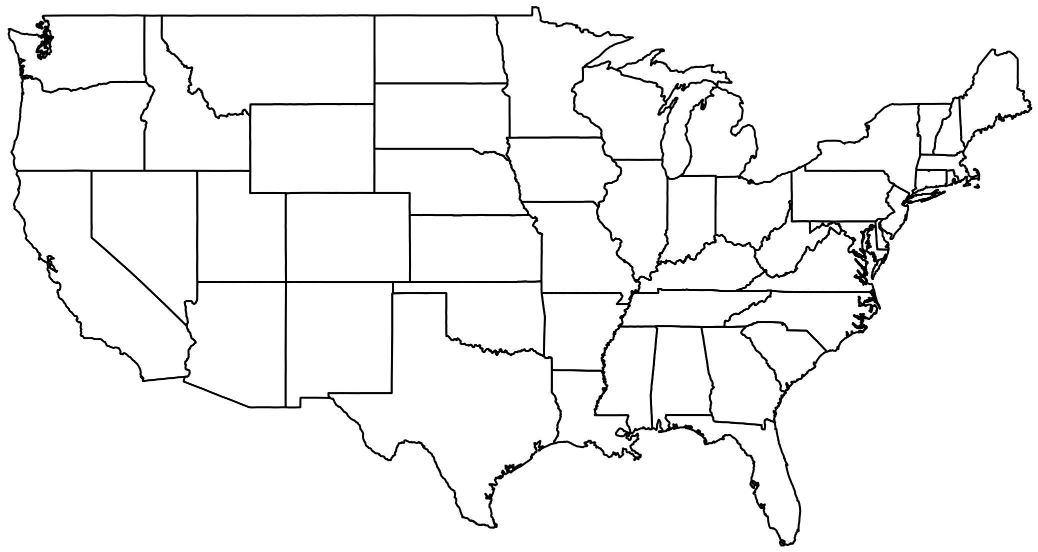
```
## Warning in (function (version = NULL, fpath = NA) : LAGOSNE version unspecified,
## loading version: 1.087.3
```

```r
# If the lagosne_get function has not worked, use this code:
# load(file = "../Data/Raw/LAGOSdata.rda")
```

## Generating spatial data frames

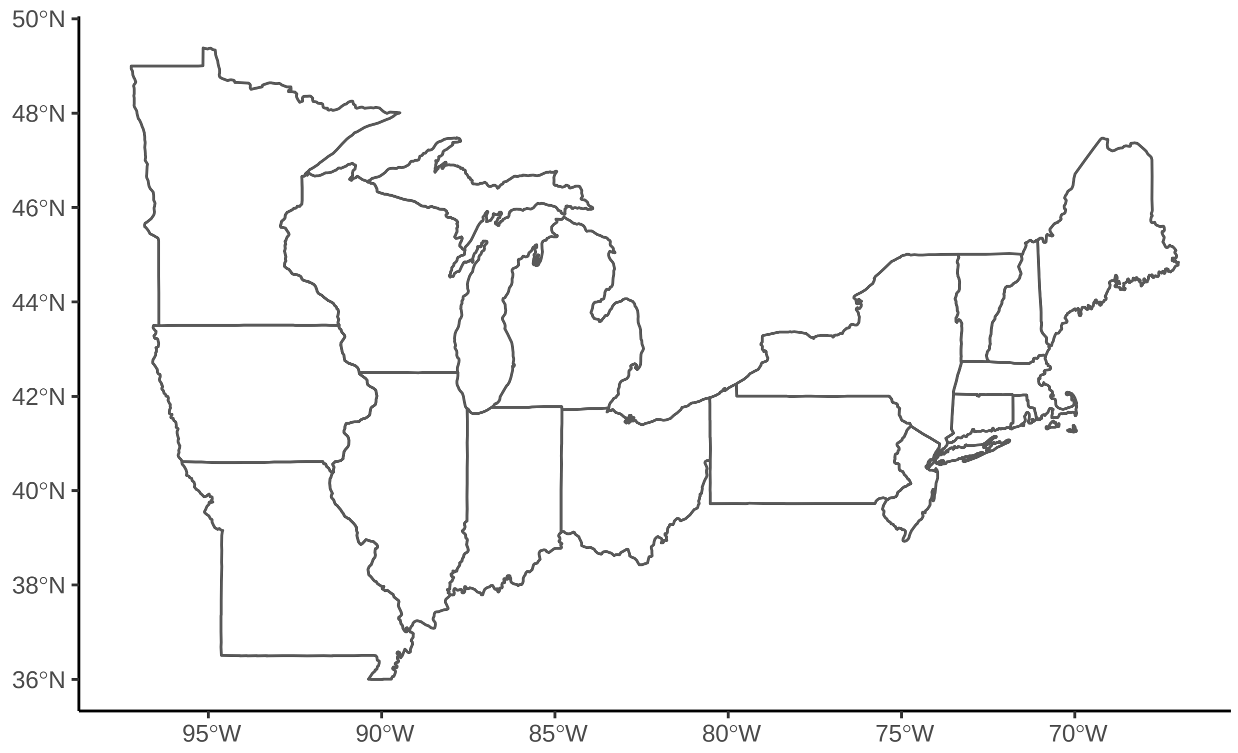
We will be using the `st_as_sf` function today, which converts an object to a simple features object. Notice in the "states" object contains a column called "geometry", which contains a list of coordinates. Notice each cell in this column is a list, which can be expanded to look at the different coordinates used to draw a polygon. We plot the polygons with the `geom_sf` function.

```r
# generate a map of U.S. states
states <- st_as_sf(map(database = "state", plot = TRUE, fill = TRUE, col = "white"))
```



```r
# filter only states that are included in the LAGOSNE database
states.subset <- filter(states, ID %in%
                          c("minnesota", "iowa", "wisconsin", "illinois",
                            "missouri", "michigan", "indiana", "ohio",
                            "pennsylvania", "new york", "new jersey",
                            "connecticut", "new hampshire", "rhode island",
                            "massachusetts", "vermont", "maine"))

# visualize state plot
ggplot(states.subset) +
  geom_sf(fill = "white")
```



## Data wrangling

```
help.search("datasets", package = "LAGOSNE")

# load LAGOSNE data frames
LAGOSlocus <- LAGOSdata$locus
LAGOSstate <- LAGOSdata$state
LAGOSnutrient <- LAGOSdata$epi_nutr
LAGOSlimno <- LAGOSdata$lakes_limno


# Create a data frame to visualize secchi depth
LAGOScombined <-
  left_join(LAGOSnutrient, LAGOSlocus) %>%
  left_join(., LAGOSlimno) %>%
  left_join(., LAGOSstate) %>%
  filter(!is.na(state)) %>%
  select(lagoslakeid, sampledate, secchi, lake_area_ha, maxdepth, nhd_lat, nhd_long, state)
```

```
## Joining, by = "lagoslakeid"

## Joining, by = c("lagoslakeid", "nhdid", "nhd_lat", "nhd_long")

## Joining, by = "state_zoneid"
```

Notice that in the absence of specifying specific columns to join by, the `left_join` function will choose columns itself. The resulting data frame has the same amount of rows as the LAGOSnutrient data frame, minus any observations that listed state as NA. Be careful when relying on this functionality! Always double check your final data frame to make sure it contains the correct data and that your joins have proceeded as planned.

Let's create a new data frame of average Secchi depth for each lake. Notice that in the `summarise` function we also compute the "mean" of max depth, latitude, and longitude for each lake. These should all be the same for every observation at a given lake, so taking the mean just uses that one value.

```r
secchi.summary <- LAGOScombined %>%
  group_by(lagoslakeid) %>%
  summarise(secchi.mean = mean(secchi),
            area = mean(lake_area_ha),
            depth = mean(maxdepth),
            lat = mean(nhd_lat),
            long = mean(nhd_long)) %>%
  drop_na()
```

## A Word on Projections

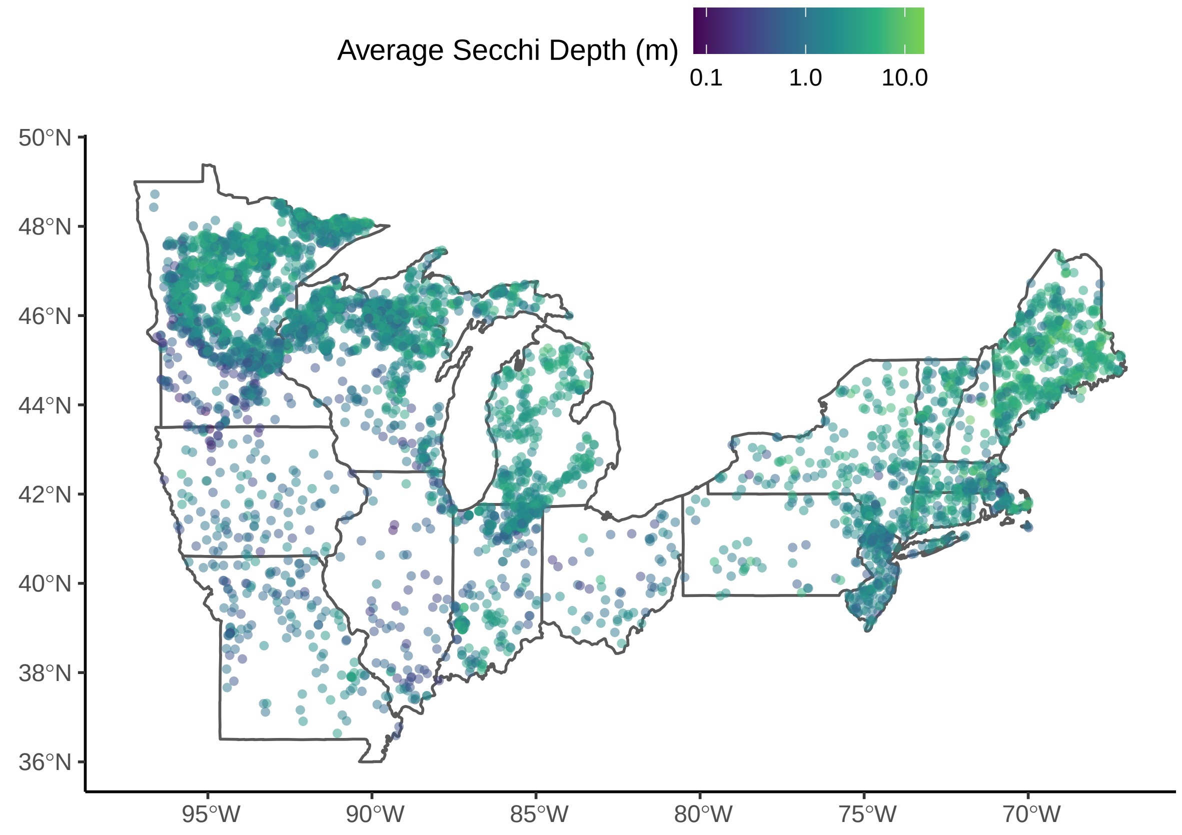
The Earth is round, meaning that if we want to make a map in 2d space we need to make a projection. This becomes a particular issue when we map large areas, because the edges of the map become distorted the further away we get from the center. A great illustration of projections and their pitfalls can be found here: https://source.opennews.org/articles/choosing-right-map-projection/.

For today, we will use the EPSG projection 4326, also known by WGS 84. This projection is the reference system for the Global Positioning System (GPS) and functions well for the LAGOSNE database.

## Plotting secchi depths across LAGOSNE lakes

```r
secchi.spatial <- st_as_sf(secchi.summary, coords = c("long", "lat"), crs = 4326)

ggplot() +
  geom_sf(data = states.subset, fill = "white") +
  geom_sf(data = secchi.spatial, aes(color = secchi.mean),
          alpha = 0.5, size = 1) +
  scale_color_viridis_c(trans = "log10", end = 0.8) +
  labs(color = "Average Secchi Depth (m)") +
  theme(legend.position = "top")
```



## Secchi depth mapping challenge.

Your turn! Notice that Maine has a large number of lakes with secchi depth measurements as well as a large range in secchi depth. Let's zoom in on this state for a closer look.

1. Filter the states and secchi depth datasets so that they contain Maine only. For the secchi depth dataset, create a summary dataset with just the mean secchi depth.
2. Create a plot of mean secchi depth for lakes in Maine, with mean secchi depth designated as color and the lake area as the size of the dot. Remember that you are using size in the aesthetics and should remove the size = 1 from the other part of the code. Adjust the transparency of points as needed.
3. Create a second plot, but this time use maximum depth of the lake as the size of the dot.
4. Plot these maps in the same plot with the `plot_grid` function. Don't worry about adjusting the legends (if you have extra time this would be a good bonus task).
5. What relationships do you see between secchi depth, lake area, and lake depth? Which of the two lake variables seems to be a stronger determinant of secchi depth? (you can make a scatterplot or run a regression to test this if time remains)

## Visualizing Secchi depth over time

We might want to visualize how monitoring efforts and Secchi depths have changed over time. Below we will divide each Secchi depth measurement by the decade in which it was taken (this can be an effective way to reduce long time series).

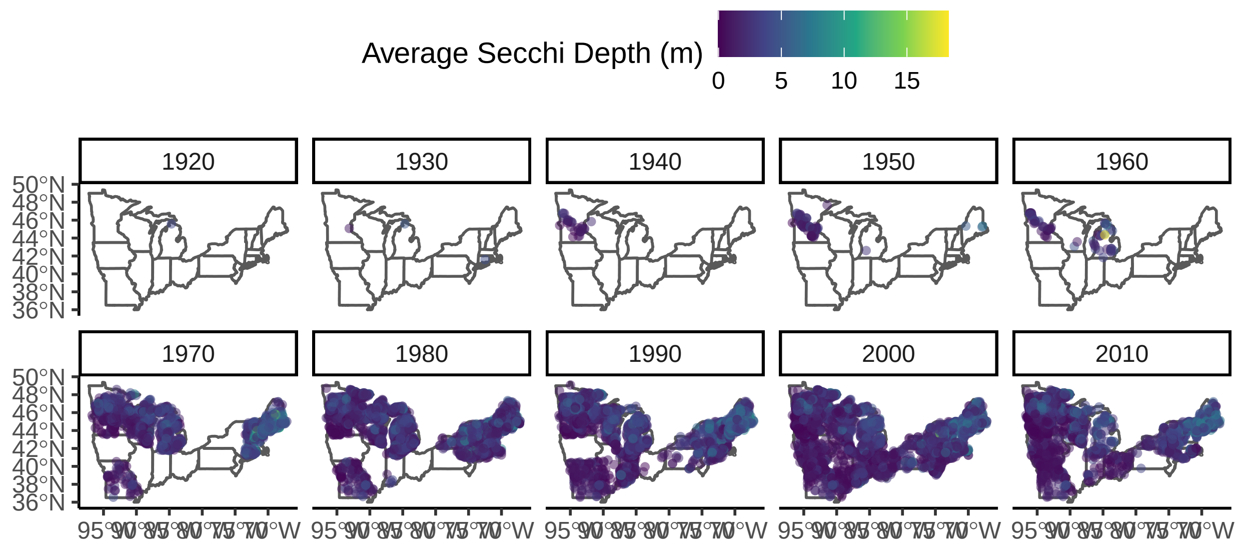
```r
# add a "decade" column
LAGOScombined <- LAGOScombined %>%
  mutate(decade = floor_date(sampledate, years (10)),
         decade = year(decade))

# create a new summary data frame, with each lake divided by decade
secchi.summary.decade <- LAGOScombined %>%
  group_by(lagoslakeid, decade) %>%
  summarise(secchi.mean = mean(secchi),
            lat = mean(nhd_lat),
            long = mean(nhd_long)) %>%
  drop_na()
```

```
## `summarise()` has grouped output by 'lagoslakeid'. You can override using the `.groups` argument.
```

```r
# create a simple features object
secchi.decade.spatial <- st_as_sf(secchi.summary.decade, coords = c("long", "lat"), crs = 4326)

# plot the data with decades as separate facets
ggplot() +
  geom_sf(data = states.subset, fill = "white") +
  geom_sf(data = secchi.decade.spatial, aes(color = secchi.mean),
          alpha = 0.5, size = 1) +
  facet_wrap(vars(decade), ncol = 5) +
  scale_color_viridis_c() +
  labs(color = "Average Secchi Depth (m)") +
  theme(legend.position = "top")
```



Faceting can be an effective way to visualize data over time. But, R has additional functionality that we can visualize these changes over time in one single graph, with the package `gganimate`. Let's build an animated plot and create a GIF.

Note: we have installed and loaded `gganimate`, but RStudio may prompt you to install additional packages. Do this as needed until the code runs without error.

```
Secchi.Animation <- ggplot() +
  geom_sf(data = states.subset, fill = "white") +
  geom_sf(data = secchi.decade.spatial, aes(color = secchi.mean),
          alpha = 0.5, size = 1) +
  scale_color_viridis_c() +
  theme(legend.position = "top") +
  #gganimate code here:
  labs(title = 'Decade: {closest_state}', color = "Average Secchi Depth (m)") +
  transition_states(decade, state_length = 1, transition_length = 0) +
  enter_appear() +
  exit_disappear()

# anim_save("./Lessons/Secchi.Animation.gif",
#            animate(Secchi.Animation, height = 400, wid = 600, renderer = gifski_renderer(loop = TRUE))
```

## Bonus: Random Forest

If we wanted to evaluate the potential for many different spatial variables to predict Secchi depth (or another response variable), we could run a variable selection model to determine which variables are the best predictors of the response. A popular method for variable selection is **random forest**, a machine learning method that can be used for both classification and regression.

See the following resources to read more about random forest analysis: randomForest package documentation Classification and Regression by randomForest R Bloggers