

Spatial data 1

There are multiple R packages that can be used to load geographic data stored in the *simple features* format. `rnaturalearth` stores data at the global scale: countries and, at finer resolution, their states and provinces. `tigris` stores data from the US Census at a much finer grain of detail: counties, census blocks, zip codes, etc. It is updated yearly.

1. Using the `rnaturalearth` package, download map data for a country of your choosing (not the US or Mexico) in `sf` format. How many simple features are in the data set? What datum was used to record it (see “Geodetic CRS”)?
2. Using the `sf` and `ggplot2` packages, make a map of your country¹.
3. Using the `tigris` package, download a map of California that shows county outlines as of 2024. How many simple features are in the data set? What datum was used to record it?

```
library(tigris)
ca_counties <- counties(state = "CA", cb = TRUE, year = 2024)
```

¹If you are unsatisfied with the resolution of your map, you can tinker with the `scale` argument of the `ne_countries()` function.

4. Make a map of the counties of California two ways: unprojected, in the longitude and latitude the data were recorded in, and projected with the [Albers Equal Area](#) projection². How do the two maps compare?

5. UC Berkeley is host to the Northern California Earthquake Data Center (<https://ncedc.org>). Its records of the earthquakes of magnitude 3 or more in California and nearby, from September 1 to October 16, 2025, are in a csv file. [Download](#) `ca-quakes.csv` and read it in with the following.

```
quakes <- read_csv("ca-quakes.csv")
```

Add the locations of these earthquakes to both of your CA maps using `geom_point()`.

²Each projection has a numeric code (its EPSG code). The Albers Equal Area projection for California is 3310. Pass that code to the `crs` argument of the `st_transform()` function in `sf`.

6. As you will have noticed, on the Albers map all of the earthquakes land on a single dot in central California. `geom_sf()` drew the counties in Albers coordinates, which are meters east and north of a point near 120° W, 38° N. `geom_point()` took the earthquakes' longitudes and latitudes to be meters too, so it placed every earthquake within about 130 meters of that point. Remake your Albers map the way we fixed the storms in lecture: convert the earthquake data frame to an sf object in WGS 84 (EPSG 4326) with `st_as_sf()`, then transform it to the Albers projection (EPSG 3310) with `st_transform()`³.

³Hint: since `quakes_albers` is an sf object instead of a tibble, `geom_point()` will no longer work. You can add it in a second `geom_sf()` layer (the first will be for the basemap).

Spatial II

The following exercises serve as an introduction to the functionality of the `sf` package. Anytime you use a new function, use `?` to learn how it works. Some of these exercises ask you to plot the result. Plots are provided to check your answer.

7. The map of California in lecture was drawn from a shapefile. [Download CA_State.zip](#), unzip it into the folder you're working in, and read the shapefile in with the following.

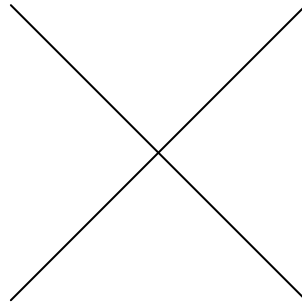
```
library(sf)
ca <- st_read("CA_State/CA_State.shp")
```

Replicate the map of California from lecture, then compute the centroid of the state and add it to the map as a point.

8. *Optional Challenge:* Which county does the centroid of California lie in? Find out computationally, using the `ca_counties` that you downloaded with `tigris` in the Spatial I problems⁴.

⁴Compare the CRS of `ca_counties` with the CRS of your centroid.

9. Following the approach used in lecture, use `st_linestring` to create two separate line segments called `l1` that stretches from (0, 0) to (1, 1) and `l2` that stretches from (0, 1) to (1, 0). View them using base R `plot()`⁵.



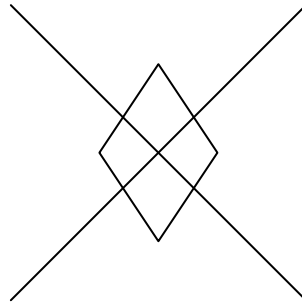
10. Form a spatial data frame (an object with classes `sf` and `data.frame`) called `my_lines` with two columns:

1. `id`: an attribute with values `1` and `2`.
2. `geom`: a *simple feature (geometry) column* containing the two lines.

Use the coordinate reference system “OGC:CRS84”.

⁵Each object will require a separate call to `plot()`. You can add `add = TRUE` to `plot()` so that the contents of the second get added to the first. Note that these plot additions don’t carry over from code cell to code cell.

11. Create a second spatial data frame, this one with a diamond-shaped polygon. The diamond should have corners at (.5, .8), (.3, .5), (.5, .2), and (.7, .5) and use the same crs. It should have an attribute called `price` with the value `100`. Call your spatial data frame `my_diamond` and plot it atop your lines.



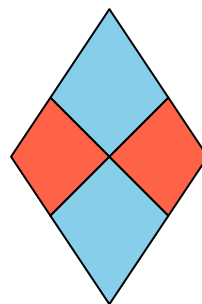
12. Calculate the dimension, length, and area of `my_lines`.

13. Calculate the dimension, length, and area of `my_diamond`.

14. Use the `st_split()` function inside the `lwgeom` package to split the diamond into four diamonds using your lines and name the resulting object `split_diamond`. After inspecting the result, run the following line to extract each of the polygons into a separate feature (row) of a spatial data frame.

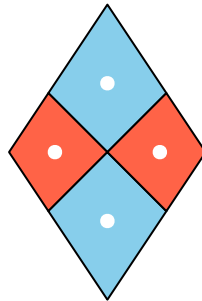
```
split_diamond <- split_diamond |>  
  st_collection_extract("POLYGON")
```

Plot the resulting `sf` collection of 4 features, with each of the split diamonds filled with a color⁶.

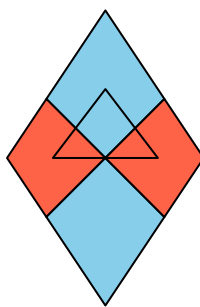


⁶You can add colors to `plot()` using the `col` argument and call `colors()` at the console to see the color names available in base R. I used `col = c("skyblue", "tomato")` but you're welcome to pick your own colors.

15. Compute the centroids of the four split diamonds, call them `my_cents`, and add them to your plot. Set their `col` and `pch` as you like.

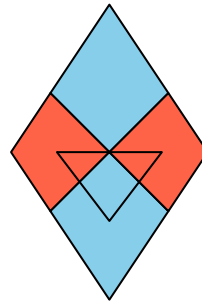


16. Using what you know about data frame subsetting, remove the bottom-most centroid. Then take the union of the three remaining points, form a convex hull from them, and save the polygon to `my_tri_up`⁷. Plot `my_tri_up` on top of your split diamond.



⁷The functions for these set operations in `sf` are exactly as you'd expect: the name of the operation prepended by `st_`. Note also that the union operation will drop the `sf` class. You'll need to run your object through `st_sf()` to get it back.

17. Repeat the previous exercise, but form an sf object called `my_tri_down` that has the top-most centroid removed.



18. Under the definition of “intersects”, which of your split diamonds intersect the `my_tri_up`? Check by doing a spatial join.

Gerrymandering

Recall the three quantitative approaches to measuring gerrymandering discussed in lecture:

1. Mean - Median Score (Grofman, 1983)
2. Efficiency Gap (Stephanopoulos and McGee, 2015)
3. Ensembles / Extreme Outliers

Use the following election results to practice applying the first two.

District	D Votes	R Votes	Result
1	62	38	D wins
2	55	45	D wins
3	44	56	R wins
4	47	53	R wins
5	46	54	R wins
Total	254	246	

To work with them in R, create the following data frame.

```
library(tidyverse)
votes <- tibble(District = 1:5,
                d_votes = c(62, 55, 44, 47, 46),
                r_votes = c(38, 45, 56, 53, 54),
                result = c(rep("D wins", 2), rep("R wins", 3)))
```

19. Calculate the mean-median score associated with this particular district breakdown of votes.

20. Calculate the efficiency gap (EG) associated with this particular district breakdown of votes⁸.

⁸It may be helpful to write a function that takes a vector of votes across districts for party A and a vector of votes for party B and returns a vector of the number of votes wasted by party A across districts.