Working with Spatial Data

In the last modules, we made calculations based on data from many data tables and pulled data from the American Community Survey and the Census of Agriculture. Now, we will move on to a different category of data – spatial data. Spatial data gives us access to many new sources of information that we can't access from traditional survey methods like land cover types, land management agency control areas, and other remotely sensed features.

Objectives

1. Use the `sf` package to explore and visualize vector data.
2. Use the `terra` package to explore and visualize raster data.
3. Use `sf` and `tigris` to retrieve a shapefile of a case study county.
4. Use a shapefile to crop a raster.
5. Calculate summary statistics for a raster.

Getting Started

This activity will help you through the most important spatial concepts for this specific project, but there are a lot of important topics that can't be addressed here. If you have not encountered spatial data before, I highly encourage you to read through the Introduction to Spatial Concepts lessons at https://datacarpentry.org/organization-geospatial/index.html.

If you are interested in using R for spatial data beyond this project, I encourage you to look into some of the many available online resources for learning to work with spatial data. Please note that some of these resources will teach you different coding approaches than the ones presented here.

1. The geospatial data Carpentries workshop at https://datacarpentry.org/geospatial-workshop. This is a workshop designed to be taught in person, but is a useful tutorial to work through independently.
2. Spatial Data Science with Applications in R is a free online textbook at https://r-spatial.org/book. It goes beyond the simple spatial calculations we will be doing to teach spatial statistical models as well. (Note: this book teaches the `stars` package instead of the `terra` package that we will use.)
3. Developers of the packages we work with have posted explanations and examples online to help people use their code. Articles about `sf` can be found at https://r-spatial.github.io/sf/articles and articles about `terra` can be found at https://rspatial.org/index.html.

(1) Open RStudio and your R project. **(2)** Create a new Quarto file.

Vector Data in sf

For this project, we can think of vector data as a polygon or a set of polygons. To get a feel for how this data is structured, we will use a built-in dataset from the `sf` package.

(3) Install the `sf` package with `install.packages` in the Console. **(4)** Create a code chunk and read in the `sf` package using `library`.

(5) Create a vector data object called `nc` and read in the built-in data using the function `st_read`. This data provides information about the counties of North Carolina. We use `system.file` because this is a dataset that comes downloaded with the `sf` package.

```
nc <- st_read(system.file("shape/nc.shp", package="sf"))
```

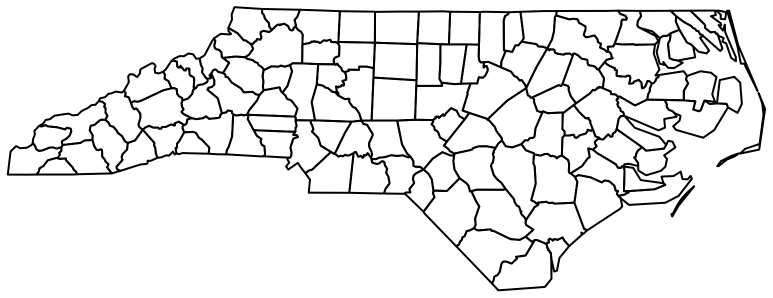
(6) Print the `head` of `nc`.

You should notice that `sf` objects are very similar to data frames, and we can use the same data cleaning techniques we used in other modules with this type of data.

However, there are a few key additions that make this data spatial. First, there's a header to the table with information like geometry type, dimension, bounding box, and geodetic CRS. Second, the last column is a special column type called `geometry` that holds the locations of all the points that make up the polygons. All the information in the data frame is connected to a polygon.

(7) To plot the shapes with no extra information, use `st_geometry` inside the `plot` function.

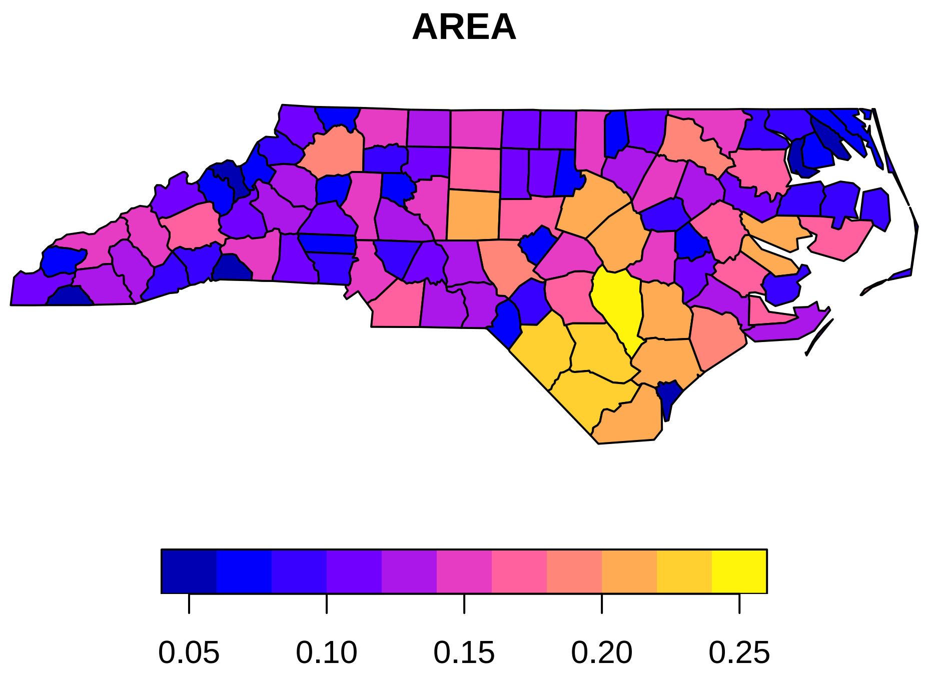
```r
plot(st_geometry(nc))
```



If you don't use `st_geometry`, `R` will make a map for every column of data. If there's too much data, this can crash your program. It's always best to plot the geometry or one column of data when exploring your data.

(8) Make a plot where the counties of North Carolina are colored by their area.

```r
plot(nc["AREA"])
```



You can use this function to get a quick look at how a certain column of information is distributed spatially.

Raster Data in terra

While vector data stores data about each shape, raster data stores data in pixels. For example, a satellite image is a raster. It stores data about the color of the land in each pixel.

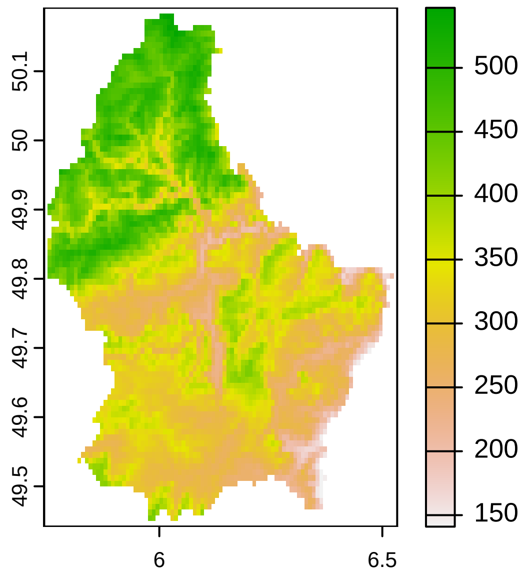
The package we use for raster data is called `terra`. **(9)** Install the `terra` package with `install.packages` in the Console. **(10)** Create a code chunk and read in the `terra` package using `library`.

(11) Create a raster data object called `elev` and read in the built-in data using the function `rast`. This data provides information about the elevation above sea level of Luxembourg. Again, we use `system.file` because this is a dataset that comes downloaded with `terra`.

```
elev <- rast(system.file("ex/elev.tif", package="terra"))
```

(12) The best way to see what's going on in a raster is to plot it. Use `plot` to visualize this raster. `terra` has built in several options like an automatic legend to make this process easier.

```
plot(elev)
```



We can also use `summary` on a raster to get some summary statistics about the pixel values. **(13)** Use `summary` to find the minimum and maximum elevation of Luxembourg.

```
summary(elev)
```

```
   elevation
 Min.   :141.0
 1st Qu.:291.0
 Median :333.0
 Mean   :348.3
 3rd Qu.:406.0
 Max.   :547.0
 NA's   :3942
```

A Note on CRS (Coordinate Reference System)

Both vector and raster data come with an attached CRS (coordinate reference system). A CRS provides `R` (and any other spatial analysis softwares) with information on the **projection** of data. Since Earth is round and maps and computer screens are flat, it's impossible to accurately depict the correct (a) area, (b) shape, and (c) distance between points. There are many projections that strike a balance between these. It's important to choose a projection that makes sense for your data and the area of the world you are working in. For more information about projections and examples of how they look for a world map, check out this blog post: https://blog.mapchart.net/misc/quick-guide-to-map-projections.

It's important to use only one CRS for a project because different spatial datasets can't interact until they have the same CRS. I have chosen an appropriate CRS for this project: EPSG 5070. This is a projection with units in meters that is created for the United States.

Changing the CRS

Our first step every time we import data into `R` is to transform it into the correct CRS (EPSG 5070). **(14)** Use the code below to re-project the vector and raster data you worked with before.

```r
nc_5070 <- st_transform(nc, "epsg:5070")

elev_5070 <- project(elev, "epsg:5070")
```

(15) Plot the old objects (`nc`, `elev`) and the re-projected objects (`nc_5070`, `elev_5070`) to see what changed.

Notice that Luxembourg looks very different! EPSG 5070 is built for the United States, so it severely warps areas that are on a different side of the world.

(16) Check that the CRS changed to EPSG 5070 with the code below.

```r
st_crs(nc_5070)$epsg
```

```
[1] 5070
```

```r
st_crs(elev_5070)$epsg
```

```
[1] 5070
```

Cropping Rasters

Just like we made subsets in the last modules to narrow down the data to one county, we need to crop large rasters to a single county to subset spatial data. To do this, we're going to crop a raster of Idaho's elevation to your county.

(17) Install the `tigris` package in the Console. **(18)** Read the `tigris` library into your Quarto document.

(19) Get a vector of all the counties in Idaho with the code below:

```r
library(tigris)

idaho <- counties(state = "ID")
```

(20) Plot the `idaho` object (refer to step (7) if needed).

```r
____(st_geometry(_____))
```

(21) Since vector data is a data frame, use `head` and/or `colnames` to find the column that holds the county names. **(22)** Create a `subset` of `idaho` where that column matches your county. This should create a data frame with one row.

```r
bear_lake <- # subset of idaho that corresponds to Bear Lake County
```

(23) Plot your county object (hint: refer to step 20).

(24) Download raster data for land value from this GitHub folder (fair_market_land_value_Idaho_2020.tif) into your `data` folder.

(25) Read the data into your `R` session with the `rast` function.

```r
land_value <- rast("data/fair_market_land_value_Idaho_2020.tif")
```

(26) Plot the raster.

This raster shows an estimate of the fair market value (land cost) in the natural log of US dollars per hectare (more information is available in Nolte, 2020). We will use it to get an approximation of each county's average land cost by cropping to a county, transforming the data, and finding the mean pixel value.

(27) Check the CRS (hint: step 16).

There are a few steps needed to crop a raster with a polygon in R:

1. Make sure the raster and polygon have the same CRS.
2. If the CRS doesn't match, `st_transform` the polygon to the CRS of the raster. Re-projecting polygons leads to less error than re-projecting rasters.
3. Use the `crop` function to the extent of the polygon.
4. Use the `mask` function to crop the raster to the exact boundary line of the polygon.

(28) Check the CRS of your county polygon object created in step 22. If the CRS doesn't match the CRS of your raster, use `st_transform` to re-project it. You can use the `st_crs` of the raster as an input for `st_transform`:

```
bear_lake_proj <- st_transform(bear_lake,
                               crs = st_crs(land_value))
```

(29) Plot the `st_geometry` of your county to check that it looks right.

(30) Use the `crop` function with the Idaho land use raster and your county polygon.

```
land_value_crop <- crop(x = land_value,
                        y = bear_lake_proj)
```

(31) Plot the cropped raster.

Notice that it has just cropped to a box, and not the actual shape of the county. The `crop` function crops a raster to a box that contains the polygon, not the exact shape. It's an important first step, but to do county-specific statistics, we need to crop to the exact shape of the county. To do that, we will use the `mask` function.

(32) Mask your *cropped* raster to your county polygon object.

```
land_value_mask <- mask(x = land_value_crop,
                        mask = bear_lake_proj)
```

(33) Plot your masked raster. Notice that the boundaries of the raster match those of your county.

Under the hood, you can think of a raster as a giant table of numbers, and every pixel one of those numbers, but with extra spatial information attached. We plot these numbers as colors, but we can also do math with these numbers.

First, let's transform the data into dollars per hectare instead of the natural log. We can add mathematical operations to a raster just as we would to a vector of numbers. To reverse a natural log ($ln(x)$), we use the inverse function e^x. The `R` function for this equation is `exp()`.

(34) Transform your county's cost data to \$/ha.

```
land_value_mask_trans <- exp(land_value_mask)
```

(35) Calculate the average land value for your county. Use the `values(land_value_mask_trans)` function to tell `R` to just analyze the numbers in the raster we called `land_value_mask_trans`, and set `na.rm` to true so that `R` ignores the pixels we masked out.

```
avg_land_value <- mean(values(land_value_mask_trans), na.rm=TRUE)
```

(36) Round the average land value to zero decimal places and add the result to the shared Google Sheet.

(Bonus: 36.1) Calculate the `median()` land value and `quantile()` information as well and include it in this report. It would be good to check if there are high land values skewing the mean and if there are other metrics we should record. Don't forget to use the `values()` function.

Finishing up

(37) At the end of your report, copy this list and paste it outside of a code chunk to create bullet points. Fill in the data you calculated for your county. Make sure that all of this information is also in the team Google Sheet for comparison with other counties. If you calculated anything extra that not on this list, be sure to add it so you have a record for later.

```
- Avg land value, 2020:
```

(38) Go back through your report and add short explanations for what each code chunk does in your own words if you haven't done so already. **(39)** Render your report to a PDF and email it to [INSERT EMAIL HERE].

Statement of original and referenced work:

The entirety of this module is original work authored by Carolyn Koehn.

License