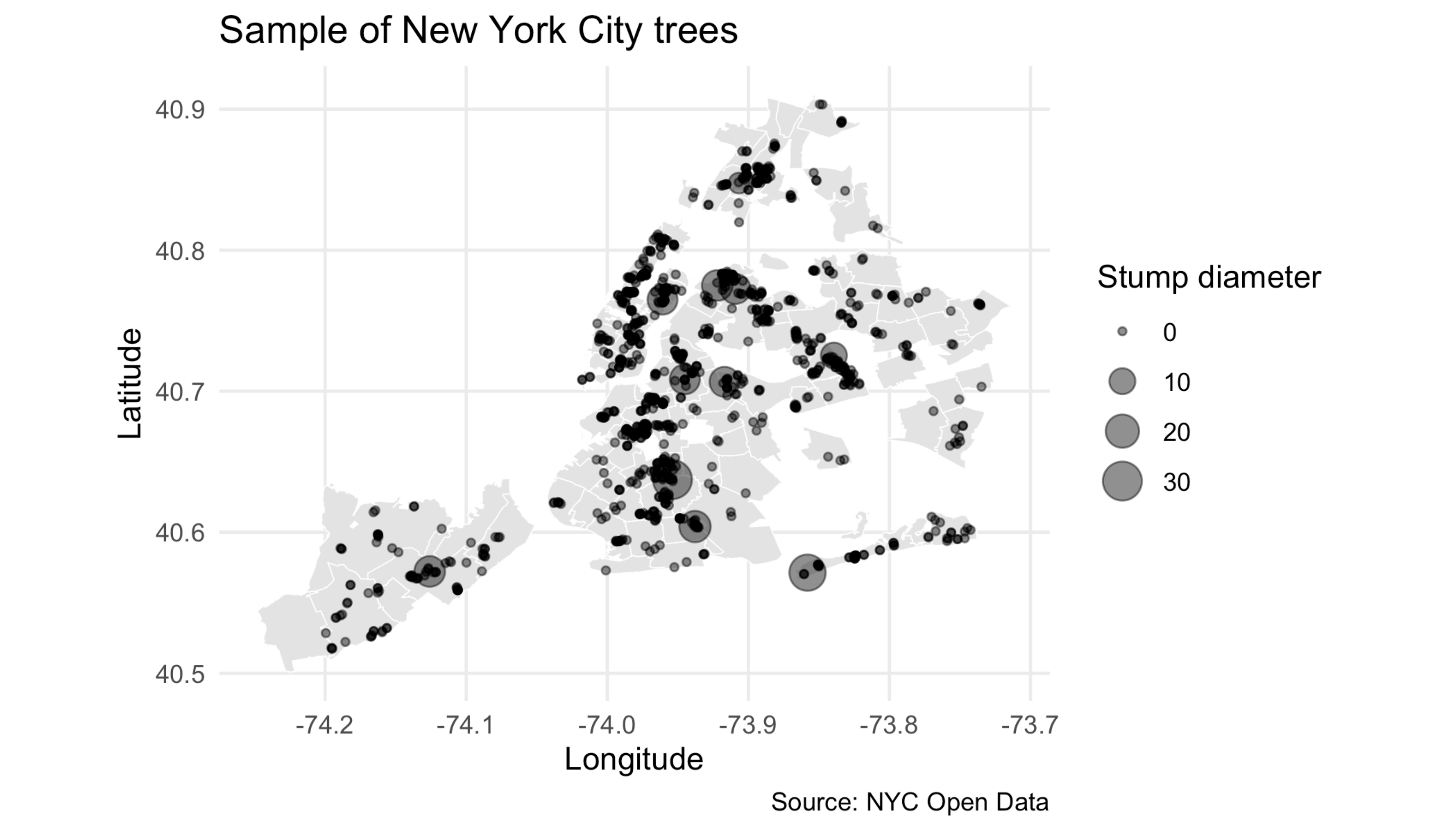
Data Science for Economists

Lecture 13: Data Visualization - part 2, Maps

Drew Van Kuiken

University of North Carolina | ECON 370

Maps



✱ Slides adapted from Grant McDermott's EC 607 at University of Oregon.

Maps in R

Packages for today:

```r
## Load and install the packages that we'll be using today
if (!require("pacman")) install.packages("pacman")
pacman::p_load(sf, tidyverse, tigris, tidycensus, jsonlite, httr, maps, spData)
```

New packages: sf, tigris, tidycensus

Features

We'll use the `sf` and `tigris` packages to get our map data today. Be forewarned: this is a quick and dirty exposition, enough to get you and up and running, but not a full treatment of maps.

`sf`:

- Stands for *simple features*
- Treats features (e.g., zip codes, borders, trees, you name it) as data frames
- Great practice for data wrangling!

Shapefiles

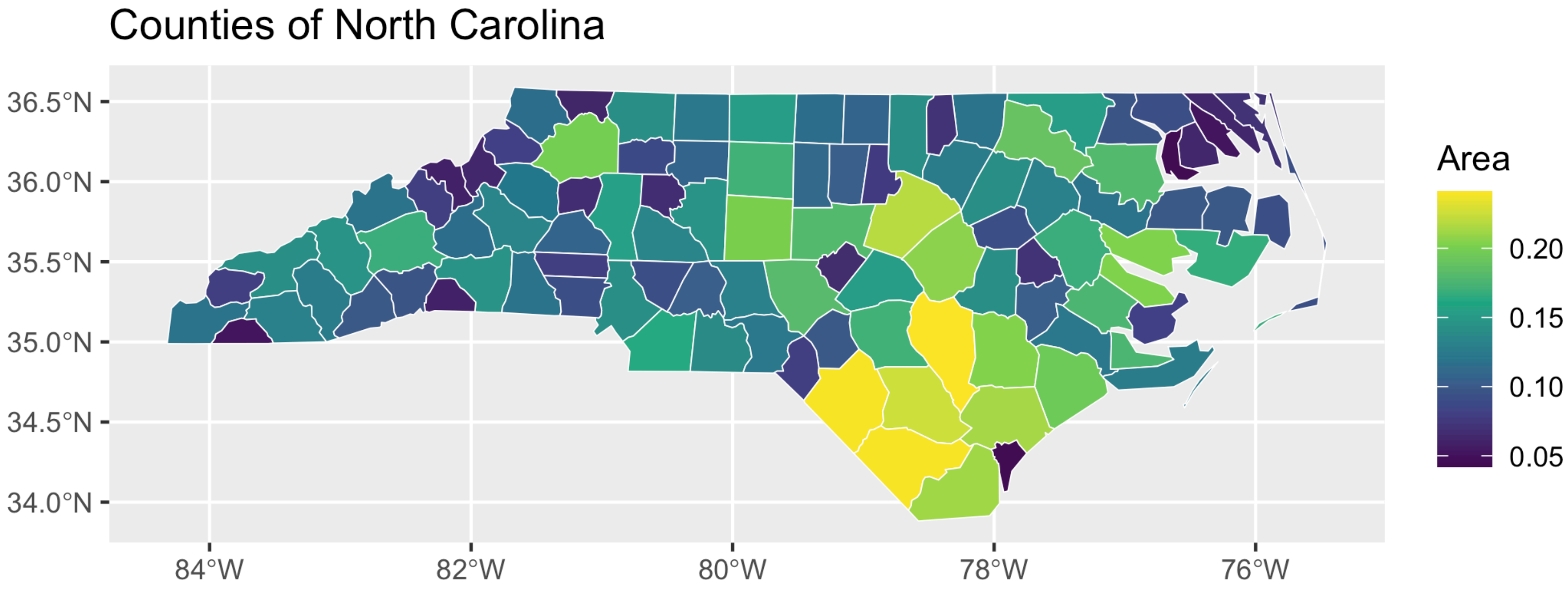
The sf package has some maps pre-loaded for us. Let's start by looking at the counties of North Carolina.

```r
file_loc = system.file("shape/nc.shp", package="sf")
nc = st_read(file_loc, quiet = TRUE)
head(nc)
```

```
## Simple feature collection with 6 features and 14 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -81.74107 ymin: 36.07282 xmax: -75.77316 ymax: 36.58965
## Geodetic CRS:  NAD27
##    AREA PERIMETER CNTY_ CNTY_ID        NAME  FIPS FIPSNO CRESS_ID BIR74 SID74
## 1 0.114     1.442  1825    1825        Ashe 37009  37009        5  1091     1
## 2 0.061     1.231  1827    1827   Alleghany 37005  37005        3   487     0
## 3 0.143     1.630  1828    1828       Surry 37171  37171       86  3188     5
## 4 0.070     2.968  1831    1831   Currituck 37053  37053       27   508     1
## 5 0.153     2.206  1832    1832 Northampton 37131  37131       66  1421     9
## 6 0.097     1.670  1833    1833    Hertford 37091  37091       46  1452     7
##   NWBIR74 BIR79 SID79 NWBIR79                       geometry
## 1      10  1364     0      19 MULTIPOLYGON (((-81.47276 3...
## 2      10   542     3      12 MULTIPOLYGON (((-81.23989 3...
## 3     208  3616     6     260 MULTIPOLYGON (((-80.45634 3...
```

Our first plot

```
ggplot(nc) + geom_sf(aes(geometry=geometry, fill=AREA), col="white") + scale_fill_vir
```



Geodetic CRS

Otherwise known as the **projection** we're using. There are lots of projections we can use.

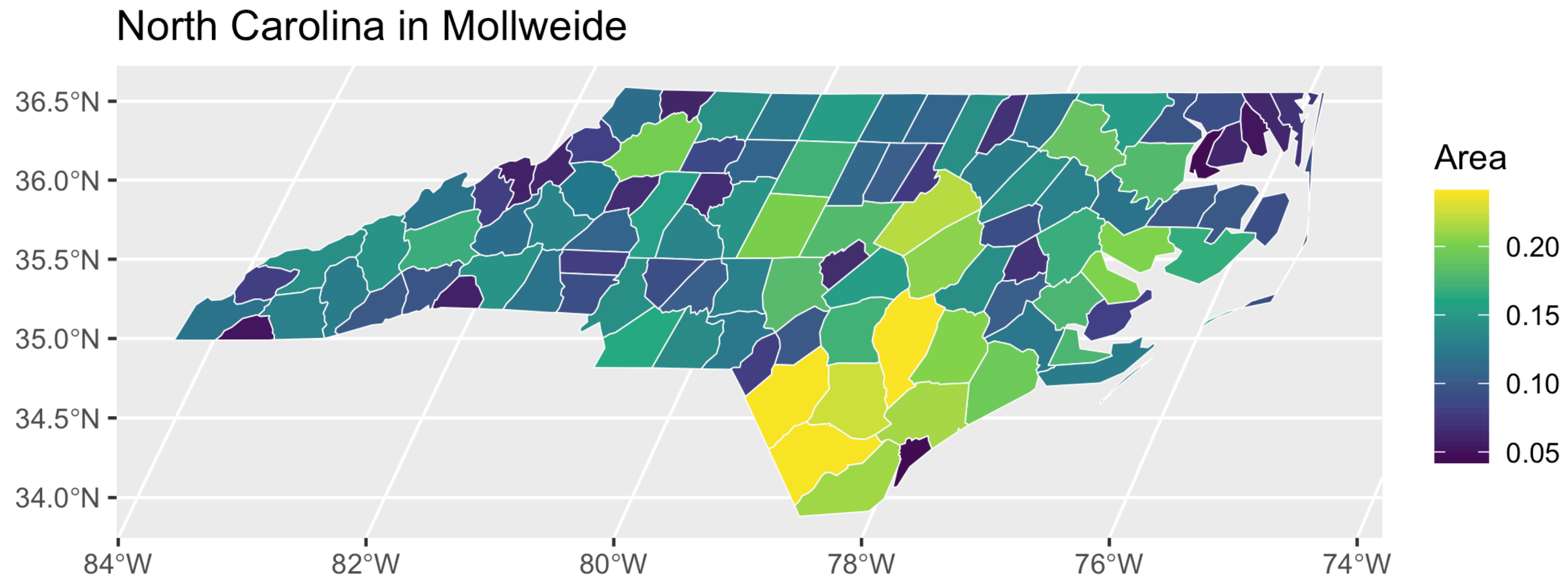
To change projection, we can use the `st_transform` function:

```
nc_moll ← nc ▷
  st_transform(crs = "+proj=moll") # using the Mollweide projection
head(nc_moll)
```

```
## Simple feature collection with 6 features and 14 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -7160488 ymin: 4345895 xmax: -6638106 ymax: 4404766
## Projected CRS: +proj=moll
##    AREA PERIMETER CNTY_ CNTY_ID        NAME  FIPS FIPSNO CRESS_ID BIR74 SID74
## 1 0.114     1.442  1825    1825        Ashe 37009  37009        5  1091     1
## 2 0.061     1.231  1827    1827   Alleghany 37005  37005        3   487     0
## 3 0.143     1.630  1828    1828       Surry 37171  37171       86  3188     5
## 4 0.070     2.968  1831    1831   Currituck 37053  37053       27   508     1
## 5 0.153     2.206  1832    1832 Northampton 37131  37131       66  1421     9
## 6 0.097     1.670  1833    1833    Hertford 37091  37091       46  1452     7
##   NWBIR74 BIR79 SID79 NWBIR79                       geometry
## 1      10  1364     0      19 MULTIPOLYGON (((-7145982 43 ...
## 2      10   542     3      12 MULTIPOLYGON (((-7118092 43 ...
## 3     208  3616     6     260 MULTIPOLYGON (((-7056268 43 ...
```

Mollweide?

Looks like this:



Not super useful. Keep the projection thing in mind though, it will come up later.

Wrangling

We can use the tidyverse to work with sf objects:

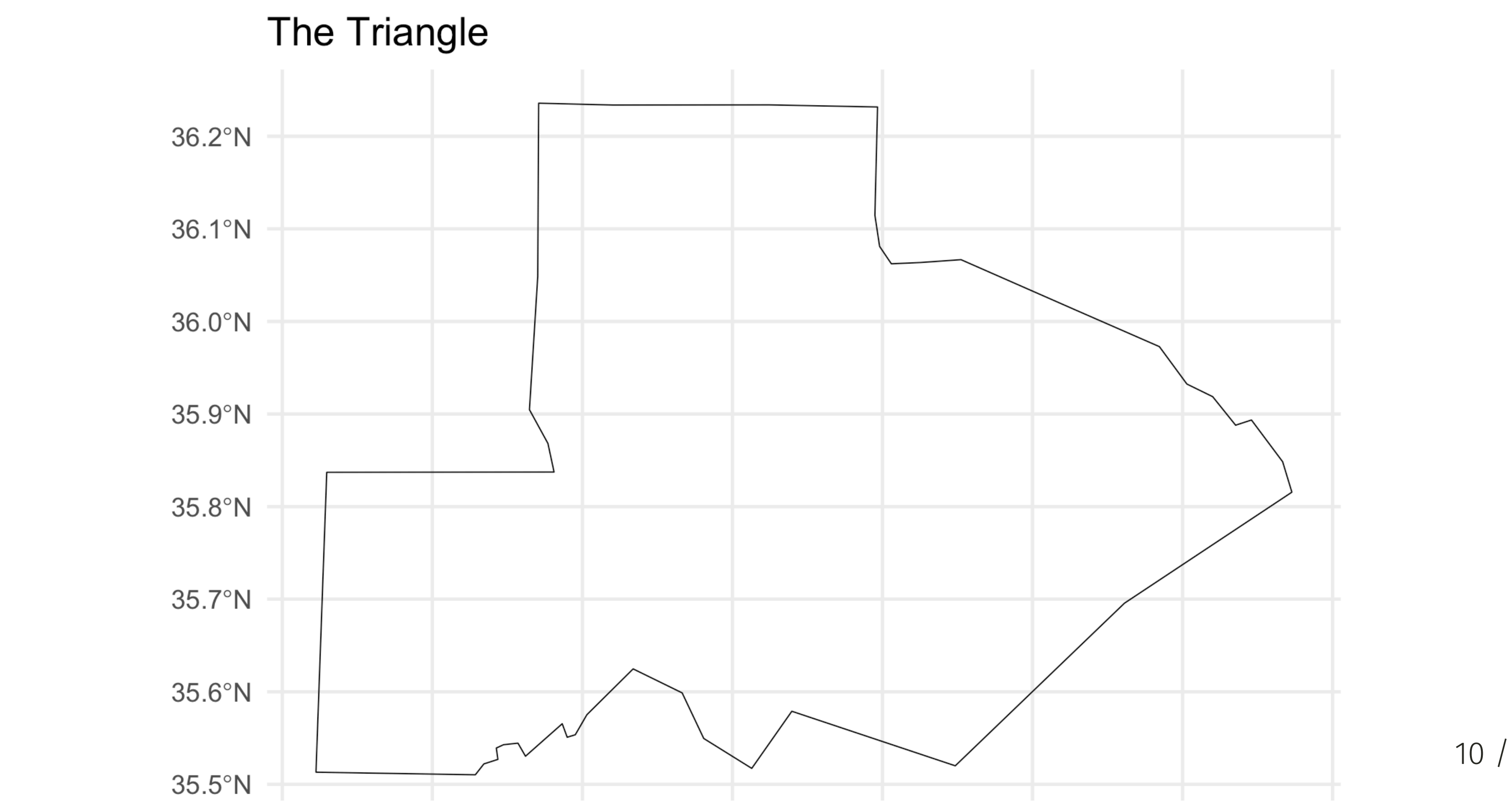
```
triangle ← nc ▷
  filter(NAME %in% c("Durham","Wake","Orange", "Chatham")) ▷
  mutate(AREA = AREA*1000) ▷
  select(NAME,AREA, everything())
head(triangle)
```

```
## Simple feature collection with 4 features and 14 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -79.55536 ymin: 35.51024 xmax: -78.25455 ymax: 36.23569
## Geodetic CRS:  NAD27
##      NAME AREA PERIMETER CNTY_ CNTY_ID  FIPS FIPSNO CRESS_ID BIR74 SID74
## 1  Orange  104     1.294  1907    1907 37135  37135       68  3164     4
## 2  Durham   77     1.271  1908    1908 37063  37063       32  7970    16
## 3    Wake  219     2.130  1938    1938 37183  37183       92 14484    16
## 4 Chatham  180     2.142  1973    1973 37037  37037       19  1646     2
##   NWBIR74 BIR79 SID79 NWBIR79                       geometry
## 1     776  4478     6    1086 MULTIPOLYGON (((-79.01814 3...
## 2    3732 10432    22    4948 MULTIPOLYGON (((-79.01814 3...
## 3    4397 20857    31    6221 MULTIPOLYGON (((-78.92107 3...
## 4     591  2398     3     687 MULTIPOLYGON (((-79.55536 3...
```

Special Operations

We can melt geographies into larger geographic units using `st_union`:

```
triangle |> st_union() |> ggplot() + geom_sf(fill=NA,col="black") + labs(title = "The
```



Other Operations

We're not going to focus on the full suite of operations available today. Here's a quick overview:

- `st_area`: get area of shape
- `st_centroid`: get geographic center of area
- `st_boundary`: returns boundary of geometry
- `st_buffer`: creates buffer around each observation
- `st_distance`: distance between two objects (or more)

Multiple Datasets

Let's pick up two new datasets:

```
france = st_as_sf(map('france', plot=FALSE, fill=TRUE))
france = france[-c(95),] # observation causing some issues, not important
head(france)
```

```
## Simple feature collection with 6 features and 1 field
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 0.06215676 ymin: 48.86568 xmax: 5.372333 ymax: 51.09752
## Geodetic CRS:  +proj=longlat +ellps=clrk66 +no_defs +type=crs
##                            ID                           geom
## Nord                     Nord MULTIPOLYGON (((2.557093 51...
## Pas-de-Calais   Pas-de-Calais MULTIPOLYGON (((2.105322 51...
## Somme                   Somme MULTIPOLYGON (((1.623103 50...
## Ardennes             Ardennes MULTIPOLYGON (((4.220728 49...
## Seine-Maritime Seine-Maritime MULTIPOLYGON (((1.419646 50...
## Aisne                   Aisne MULTIPOLYGON (((3.15867 50....
```

```
data("seine", package = "spData")
head(seine)
```

```
## Simple feature collection with 3 features and 1 field
```

Merging

A couple thoughts:

1. `st_as_sf`: turns non-shapefile data into shapefile data. `st_as_sf(x,coordinates=c("longitude","latitude"))` turns any lat/lon data into a sf object.
2. `st_intersect`: creates a dataset with the exact points of overlap between two objects.
3. `st_join`: joins together two sf objects based on any overlapping geometries
   - Example of 2 and 3 in a sec

Try taking the intersection of the france and seine sf objects now.

Didn't Work, Huh

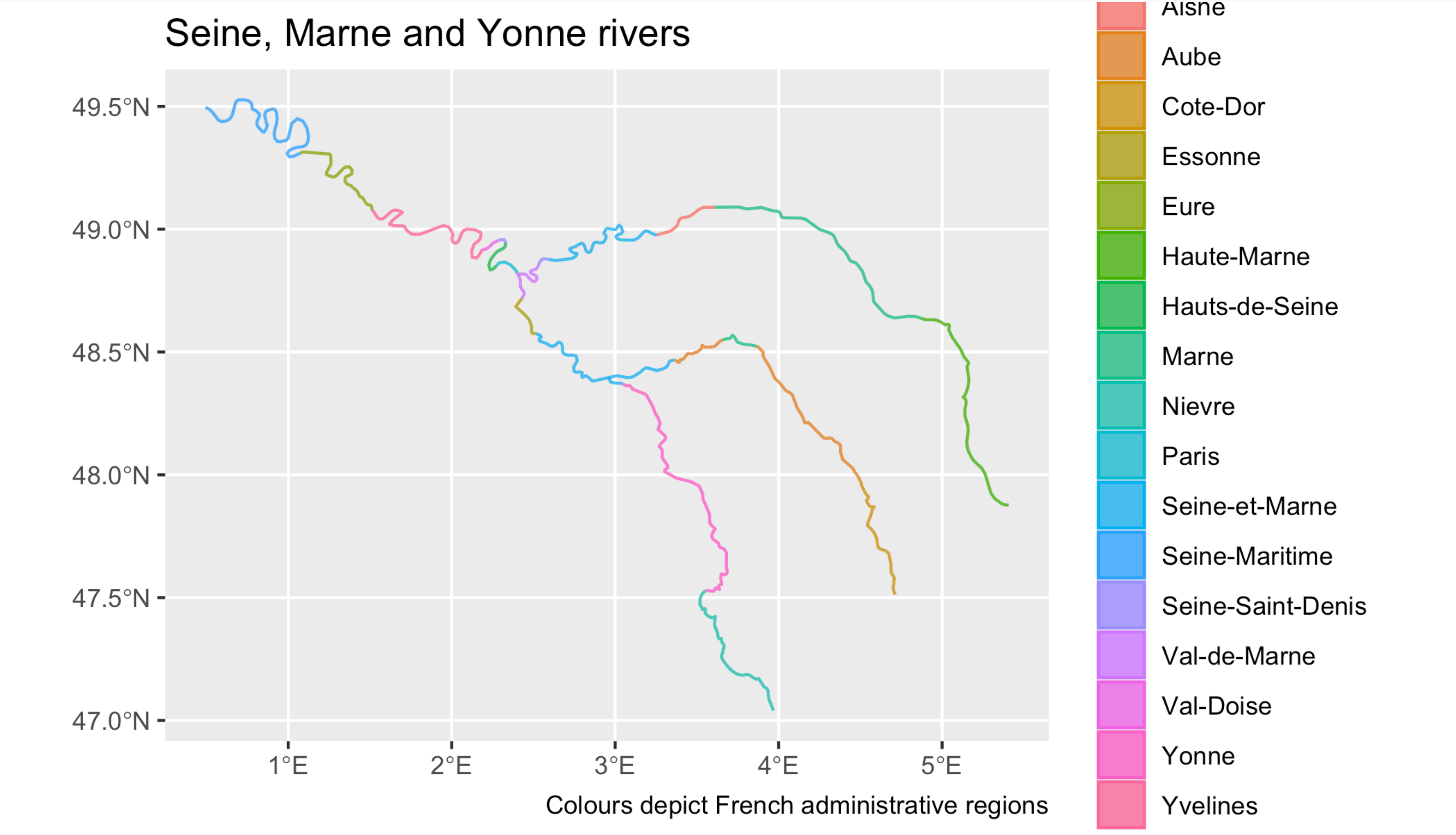
Why didn't it work?

```
seine_crs = st_transform(seine, crs = st_crs(france))
france_intersected = st_intersection(france, seine_crs)
```

```
## Warning: attribute variables are assumed to be spatially constant throughout
## all geometries
```

The projections were different!

So what'd we create?

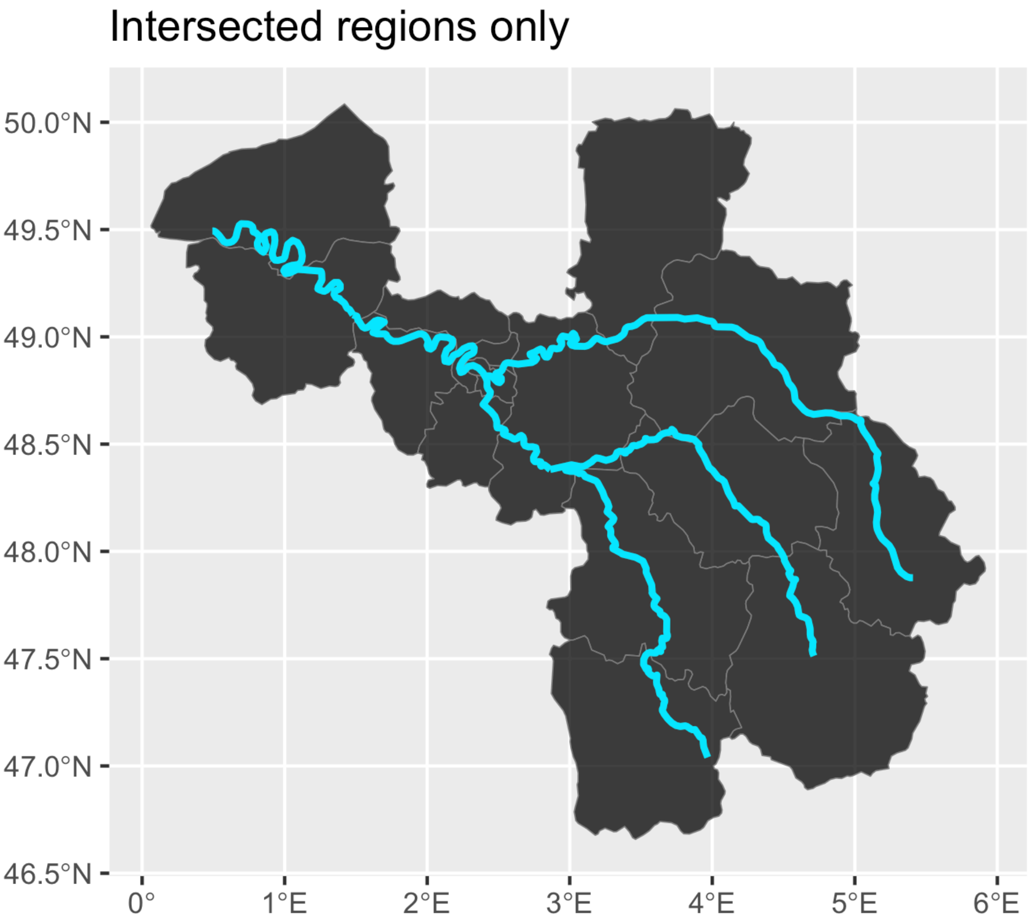


Trying out joins

```
france_river <- st_join(france,seine_crs) |> # what kind of join is this?
  filter(!is.na(name)) |>
  distinct(ID, .keep_all = T) # some rows merge twice because 2 branches of river
head(france_river)
```

```
## Simple feature collection with 6 features and 2 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 0.06215676 ymin: 48.44897 xmax: 5.032294 ymax: 50.08539
## Geodetic CRS:  +proj=longlat +ellps=clrk66 +no_defs +type=crs
##                             ID  name                           geom
## Seine-Maritime Seine-Maritime Seine MULTIPOLYGON (((1.419646 50...
## Aisne                   Aisne Marne MULTIPOLYGON (((3.15867 50....
## Eure                     Eure Seine MULTIPOLYGON (((0.4339198 4...
## Marne                   Marne Marne MULTIPOLYGON (((4.059515 49...
## Val-Doise           Val-Doise Seine MULTIPOLYGON (((1.706263 49...
## Yvelines             Yvelines Seine MULTIPOLYGON (((1.624106 49...
```

Put It In a Graph



tidycensus

The tidycensus package lets you avoid setting up even an API call to get census data, it's sort of amazing.

We'll only scratch the surface of the tidycensus package. If you wanted to get ACS data (zipcode-level survey on all kinds of stuff), you'd need to grab an API key from the census. The instructions to do this are in the class script for today.

One thing that's available in the tidycensus: TIGER/Line shapefiles (to access these, we loaded the `tigris` package)

Tiger shapefiles can give us shapefiles for things like: nations, regions, states, counties, census tracts, school districts, zip codes, and way more

Zip Codes in NY

Let's grab the zip codes for the urban area New York City:

```r
ny_zips <- zctas(state = "NY", class = "sf", year = 2010) # zips only available in 20
urb <- urban_areas(year=2020) |> filter(grepl("New York",NAME10))
ny_urb_zips <- st_join(ny_zips,urb) |> filter(!is.na(NAME10))
```

Your task -- create 2 graphs:

1. Zip codes of NYC which contain trees downloaded from NYC Open Data
2. All zip codes in NY urban area, with points representing trees in NYC