wk05-01_inclass

Patrick Bitterman

2025-02-09

Week 05.01 in-class

Today's exercise is a bit different. This notebook will demonstrate some new techniques while also allowing you to become more comfortable with the R Markdown (Rmd) format. You can click the little green arrow for each of the code blocks to run everything IN THAT BLOCK.

First, let's add the packages we'll need. NOTE, you may also need to install some of these packages onto your computer BEFORE you're able to use them. Do you remember the command to install a package?

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```r
library(sf)
```

```
## Warning: package 'sf' was built under R version 4.4.1
```

```
## Linking to GEOS 3.11.0, GDAL 3.5.3, PROJ 9.1.0; sf_use_s2() is TRUE
```

```r
library(terra)
```

```
## Warning: package 'terra' was built under R version 4.4.1
```

```
## terra 1.8.10
##
## Attaching package: 'terra'
##
## The following object is masked from 'package:tidyr':
##
##     extract
```

```r
library(tidyterra)
```

```
## Warning: package 'tidyterra' was built under R version 4.4.1
```

```
##
## Attaching package: 'tidyterra'
##
## The following object is masked from 'package:stats':
##
##     filter
```

```r
library(tmap)
```

```
## Warning: package 'tmap' was built under R version 4.4.1
```

Let's load some data. Note, this is a different file format than you're (probably) used to. Check out https://www.geopackage.org if you want to learn more (and you should).

You may also notice the path is structured slightly differently that before. When in standard R script (for example, myscript.R), the "." notation refers to the location of the RStudio project file. HOWEVER, when using Rmd files, the starting location is where the .Rmd file is. Therefore, we need to edit our path a bit. ".." means "go up a level" (in this case, FROM the src directory and TO the root of the project) THEN find the `data` directory, then the `ohio` directory, then find the file.

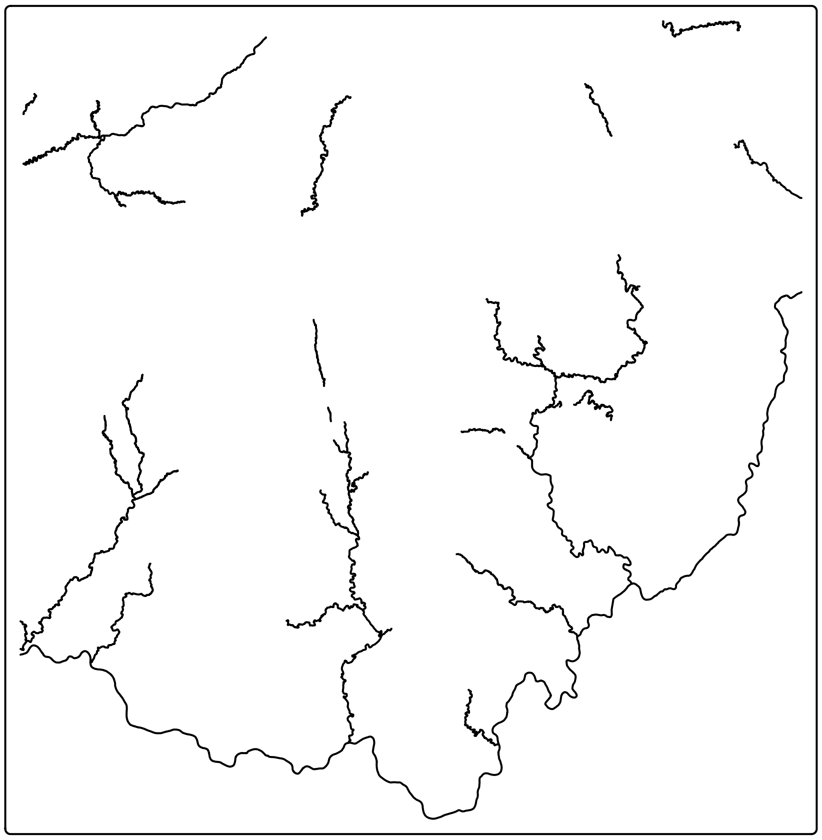
Anyways, now we have some stream data. I like to always check the projection information. What's the projection?

```r
oh_streams %>% sf::st_crs()
```

```
## Coordinate Reference System:
##   User input: WGS 84 / Pseudo-Mercator
##   wkt:
## PROJCRS["WGS 84 / Pseudo-Mercator",
##     BASEGEOGCRS["WGS 84",
##         ENSEMBLE["World Geodetic System 1984 ensemble",
##             MEMBER["World Geodetic System 1984 (Transit)"],
##             MEMBER["World Geodetic System 1984 (G730)"],
##             MEMBER["World Geodetic System 1984 (G873)"],
##             MEMBER["World Geodetic System 1984 (G1150)"],
##             MEMBER["World Geodetic System 1984 (G1674)"],
##             MEMBER["World Geodetic System 1984 (G1762)"],
##             MEMBER["World Geodetic System 1984 (G2139)"],
##             ELLIPSOID["WGS 84",6378137,298.257223563,
##                 LENGTHUNIT["metre",1]],
##             ENSEMBLEACCURACY[2.0]],
##         PRIMEM["Greenwich",0,
##             ANGLEUNIT["degree",0.0174532925199433]],
##         ID["EPSG",4326]],
##     CONVERSION["Popular Visualisation Pseudo-Mercator",
##         METHOD["Popular Visualisation Pseudo Mercator",
##             ID["EPSG",1024]],
##         PARAMETER["Latitude of natural origin",0,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8801]],
##         PARAMETER["Longitude of natural origin",0,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8802]],
##         PARAMETER["False easting",0,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8806]],
##         PARAMETER["False northing",0,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8807]]],
##     CS[Cartesian,2],
##         AXIS["easting (X)",east,
##             ORDER[1],
##             LENGTHUNIT["metre",1]],
##         AXIS["northing (Y)",north,
##             ORDER[2],
##             LENGTHUNIT["metre",1]],
##     USAGE[
##         SCOPE["Web mapping and visualisation."],
##         AREA["World between 85.06°S and 85.06°N."],
##         BBOX[-85.06,-180,85.06,180]],
##     ID["EPSG",3857]]
```

And then we can map it. I'm introducing a new package `tmap` today. This package does thematic mapping (hence, tmap) with various spatial data. The syntax uses the `+` notation similar to (but not exactly like) `ggplot`. You'll notice it's MUCH faster than the standard `plot()` command.

```
tm_shape(oh_streams) + tm_lines()
```



Let's grab some more data

```r
oh_counties <- read_sf("../data/ohio/oh_counties.gpkg")
oh_counties %>% glimpse()
```

```
## Rows: 88
## Columns: 19
## $ STATEFP  <chr> "39", "39", "39", "39", "39", "39", "39", "39", "39", "39", "~
## $ COUNTYFP <chr> "063", "003", "085", "047", "017", "115", "133", "145", "163"~
## $ COUNTYNS <chr> "01074044", "01074015", "01074055", "01074036", "01074021", "~
## $ GEOID    <chr> "39063", "39003", "39085", "39047", "39017", "39115", "39133"~
## $ GEOIDFQ  <chr> "0500000US39063", "0500000US39003", "0500000US39085", "050000~
## $ NAME     <chr> "Hancock", "Allen", "Lake", "Fayette", "Butler", "Morgan", "P~
## $ NAMELSAD <chr> "Hancock County", "Allen County", "Lake County", "Fayette Cou~
## $ LSAD     <chr> "06", "06", "06", "06", "06", "06", "06", "06", "06", "06", "~
## $ CLASSFP  <chr> "H1", "H1", "H1", "H1", "H1", "H1", "H1", "H1", "H1", "H1", "~
## $ MTFCC    <chr> "G4020", "G4020", "G4020", "G4020", "G4020", "G4020", "G4020"~
## $ CSAFP    <chr> "248", "338", "184", "198", "178", NA, "184", "170", NA, NA, ~
## $ CBSAFP   <chr> "22300", "30620", "17410", "47920", "17140", NA, "10420", "39~
## $ METDIVFP <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ FUNCSTAT <chr> "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "~
## $ ALAND    <dbl> 1376122055, 1042587391, 594129618, 1052469885, 1208270096, 10~
## $ AWATER   <dbl> 6024245, 11152061, 1942308103, 1694038, 9196537, 13868572, 43~
## $ INTPTLAT <chr> "+41.0002170", "+40.7716274", "+41.7781416", "+39.5552462", "~
## $ INTPTLON <chr> "-083.6659471", "-084.1061032", "-081.1973297", "-083.4618927~
## $ geom     <MULTIPOLYGON [°]> MULTIPOLYGON (((-83.61191 4..., MULTIPOLYGON (((~
```

So now we have all counties in Ohio. Cool. Let's do some simple calculations with the data

```
counties_areas <- oh_counties %>% sf::st_area()
```

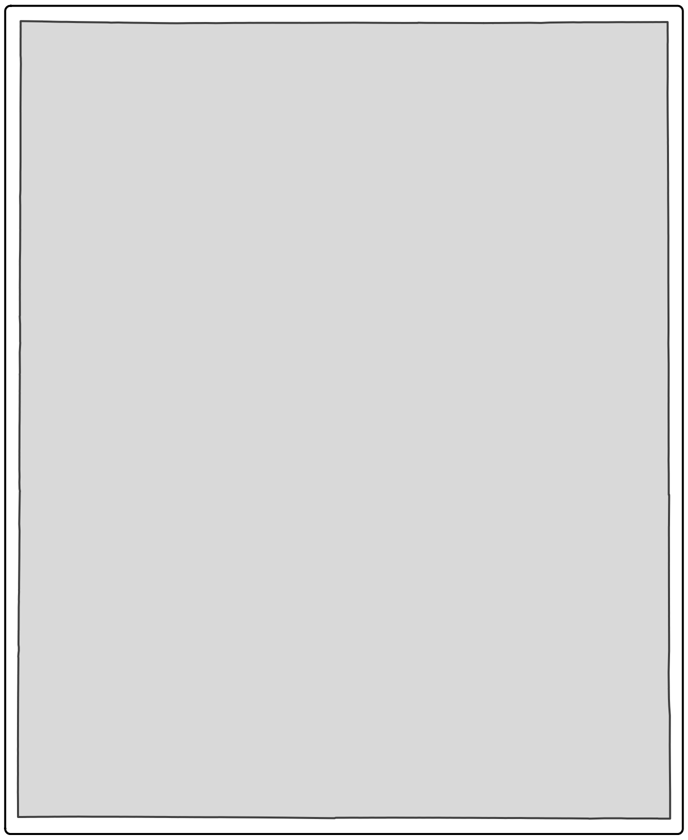
If you wanted to, how would you add the areas back to the sf data.frame?

Let's subset our data so that we're not working with ALL of Ohio. There are lots of ways to do this. How would we get ONLY Portage county?

```
portage <- oh_counties %>% dplyr::filter(., NAME == "Portage")
```

Check it/plot it

```
portage %>% tm_shape(.) + tm_polygons()
```



Yep, it's a rectangle.

Let's make a slightly larger study area to include Summit County as well. How could we do that? Let's just use an "or" within the filter command.

```
port.summit <- oh_counties %>% dplyr::filter(., NAME == "Portage" | NAME == "Summit")
```

That can be a bit clunky if we need to string together a bunch of "or" commands. So let's try a different notation that's also a bit more reuseable.

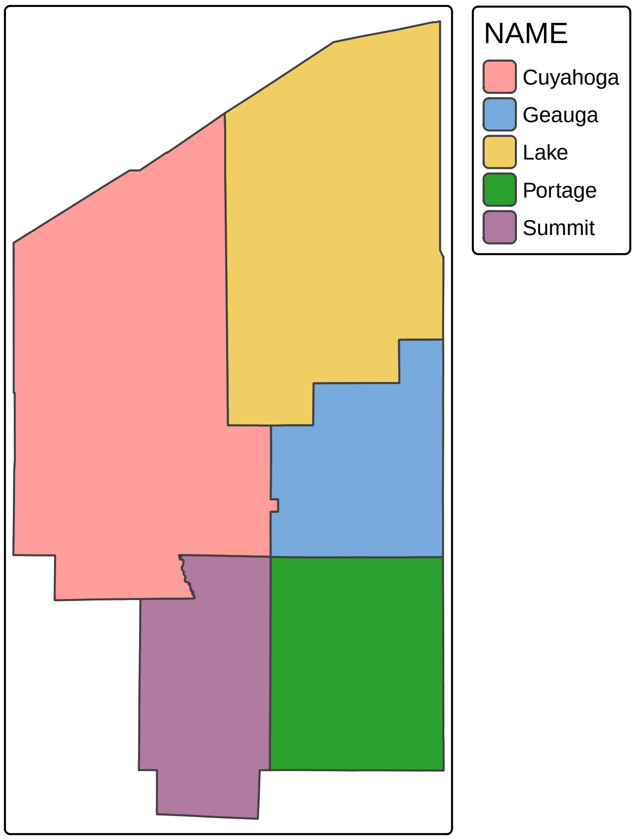
```
# what counties do I want?

# Make a simple vector
mycounties <- c("Portage", "Summit", "Lake", "Cuyahoga", "Geauga")

# then do the filter. Note the %in% notation. How do you think this works???
study.area <- oh_counties %>% dplyr::filter(., NAME %in% mycounties)
```

Plot it to check, add a fill based on a variable. It very handily adds a simple legend too!

```
study.area %>% tm_shape(.) + tm_polygons(fill = "NAME")
```



The streams dataset includes a variable for whether that stream segment is classified as impaired and on the "303d" list, which is list of impaired streams defined by section 303d of the Clean Water Act. Let's filter the line file such that we only have those streams

```
streams.303d <- oh_streams %>% dplyr::filter(., on303dlist == "Y")
# It would make more sense if they used a logical rather than Y/N, but I didn't create the data
```

Next, let's find only those 303d streams that are in our study area? What's the spatial operation again? Yes, an intersection

Oops, that didn't work. What was the problem?

Let's try again, this time dealing with the spatial reference/coordinate systems properly

```
st_crs(study.area)
```

```
## Coordinate Reference System:
##   User input: NAD83
##   wkt:
## GEOGCRS["NAD83",
##     DATUM["North American Datum 1983",
##         ELLIPSOID["GRS 1980",6378137,298.257222101,
##             LENGTHUNIT["metre",1]]],
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433]],
##     CS[ellipsoidal,2],
##         AXIS["geodetic latitude (Lat)",north,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["geodetic longitude (Lon)",east,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433]],
##     USAGE[
##         SCOPE["Geodesy."],
##         AREA["North America - onshore and offshore: Canada - Alberta; British Columbia; Manitoba; Ne
##         BBOX[14.92,167.65,86.45,-40.73]],
##     ID["EPSG",4269]]
```

```
st_crs(oh_streams)
```

```
## Coordinate Reference System:
##   User input: WGS 84 / Pseudo-Mercator
##   wkt:
## PROJCRS["WGS 84 / Pseudo-Mercator",
##     BASEGEOGCRS["WGS 84",
##         ENSEMBLE["World Geodetic System 1984 ensemble",
##             MEMBER["World Geodetic System 1984 (Transit)"],
##             MEMBER["World Geodetic System 1984 (G730)"],
##             MEMBER["World Geodetic System 1984 (G873)"],
##             MEMBER["World Geodetic System 1984 (G1150)"],
##             MEMBER["World Geodetic System 1984 (G1674)"],
##             MEMBER["World Geodetic System 1984 (G1762)"],
##             MEMBER["World Geodetic System 1984 (G2139)"],
##             ELLIPSOID["WGS 84",6378137,298.257223563,
##                 LENGTHUNIT["metre",1]],
##             ENSEMBLEACCURACY[2.0]],
##         PRIMEM["Greenwich",0,
##             ANGLEUNIT["degree",0.0174532925199433]],
##         ID["EPSG",4326]],
##     CONVERSION["Popular Visualisation Pseudo-Mercator",
##         METHOD["Popular Visualisation Pseudo Mercator",
##             ID["EPSG",1024]],
##         PARAMETER["Latitude of natural origin",0,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8801]],
##         PARAMETER["Longitude of natural origin",0,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8802]],
##         PARAMETER["False easting",0,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8806]],
##         PARAMETER["False northing",0,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8807]]],
##     CS[Cartesian,2],
##         AXIS["easting (X)",east,
##             ORDER[1],
##             LENGTHUNIT["metre",1]],
##         AXIS["northing (Y)",north,
##             ORDER[2],
##             LENGTHUNIT["metre",1]],
##     USAGE[
##         SCOPE["Web mapping and visualisation."],
##         AREA["World between 85.06°S and 85.06°N."],
##         BBOX[-85.06,-180,85.06,180]],
##     ID["EPSG",3857]]
```

```r
# they're not the same, so we need to reproject them into a common CRS

# The 6346 is an EPSG code (see: https://epsg.io) for a UTM 16N CRS

# let's reproject this one first
study.area_p <- sf::st_transform(study.area, 6346)

# we COULD (and maybe should) use a similar command to reproject the streams file too.
#But let's do something a bit different/crazy just to show what's possible


# Before you run this next line, break down what it does FIRST. It's definitely non-traditional

study.area_p %>% st_crs() %>% sf::st_transform(study.area, .) -> oh_streams_p

# Now, while the above line technically works, it's not very readable,
# and an example of "just because you can, doesn't mean you should"

# something like this is probably better

oh_streams_p <- study.area_p %>% st_crs() %>% sf::st_transform(oh_streams, .)
```

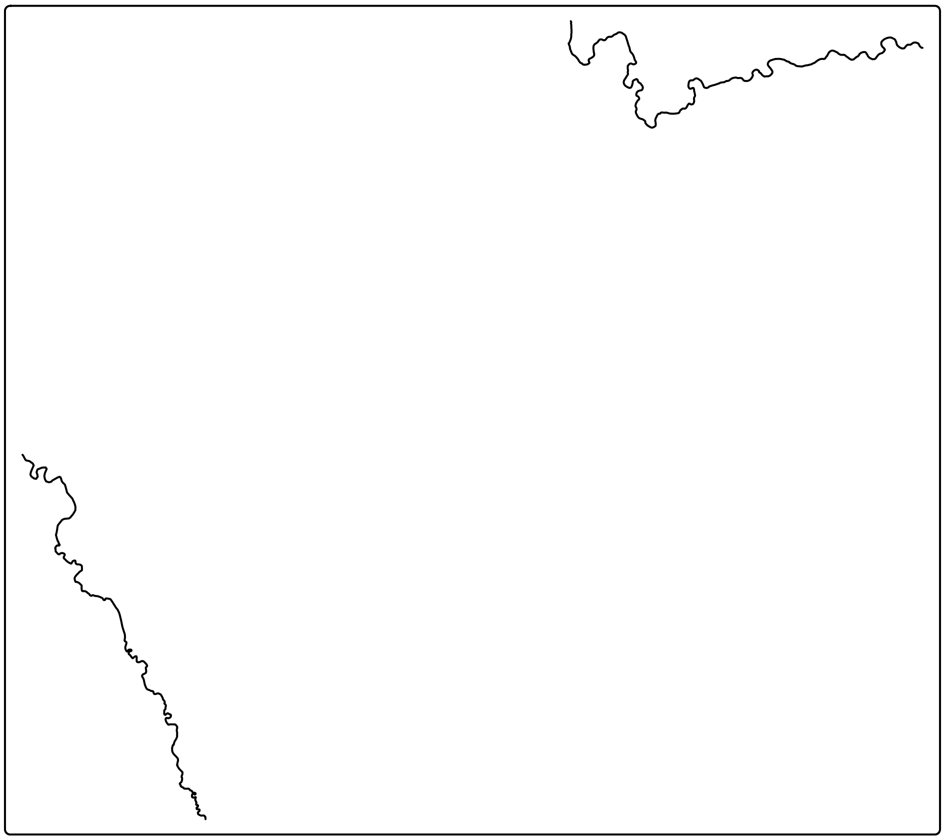
now let's try that intersect function again

```r
study.streams <- sf::st_intersection(oh_streams_p, study.area_p)
```

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

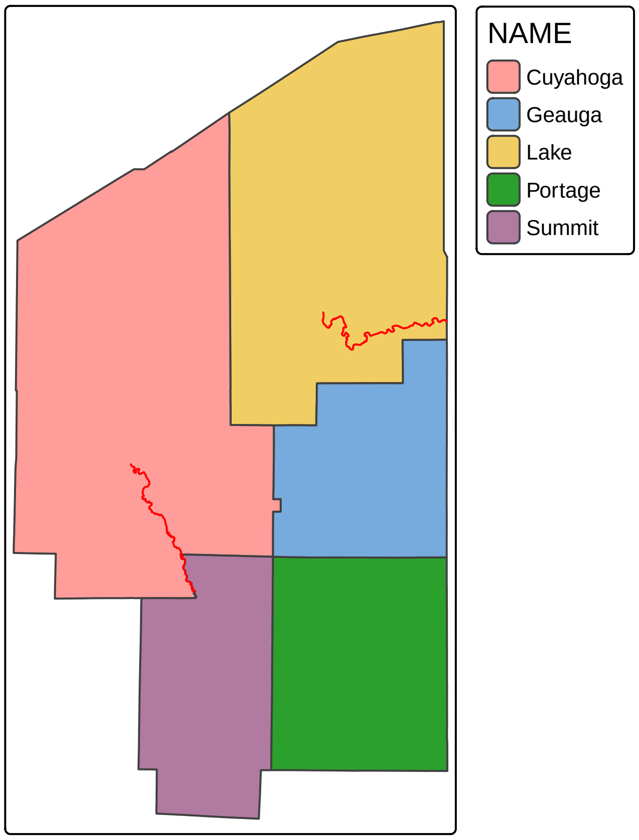
Plot it

```r
tm_shape(study.streams) + tm_lines()
```



Let's add both layers

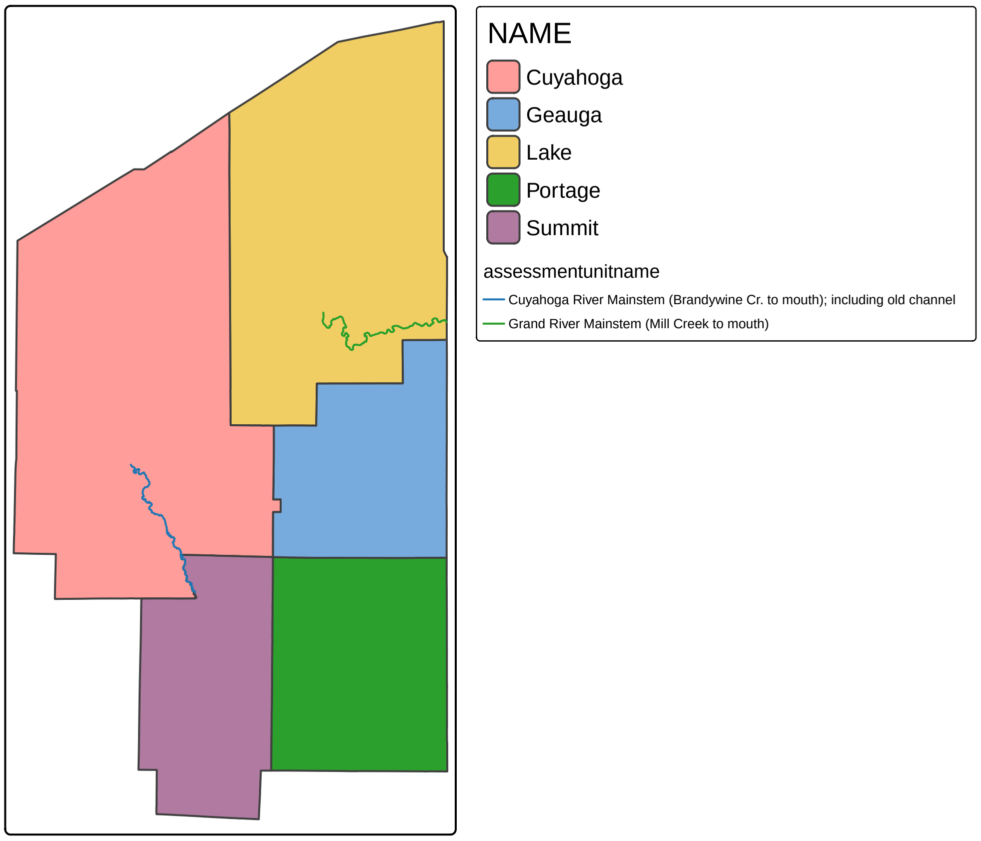
```r
tm_shape(study.area_p) + tm_polygons(fill = "NAME") +
  tm_shape(study.streams) + tm_lines(col = "red") # this colors the lines based on a color we gave it (
```



Another option

```r
tm_shape(study.area_p) + tm_polygons(fill = "NAME") +
  tm_shape(study.streams) + tm_lines(col = "assessmentunitname") # this colors the lines based on a var
```

```
## [plot mode] fit legend/component: Some legend items or map compoments do not
## fit well, and are therefore rescaled.
## i Set the tmap option `component.autoscale = FALSE` to disable rescaling.
```


NAME
Cuyahoga
Geauga
Lake
Portage
Summit
assessmentunitname
Cuyahoga River Mainstem (Brandywine Cr. to mouth); including old channel
Grand River Mainstem (Mill Creek to mouth)

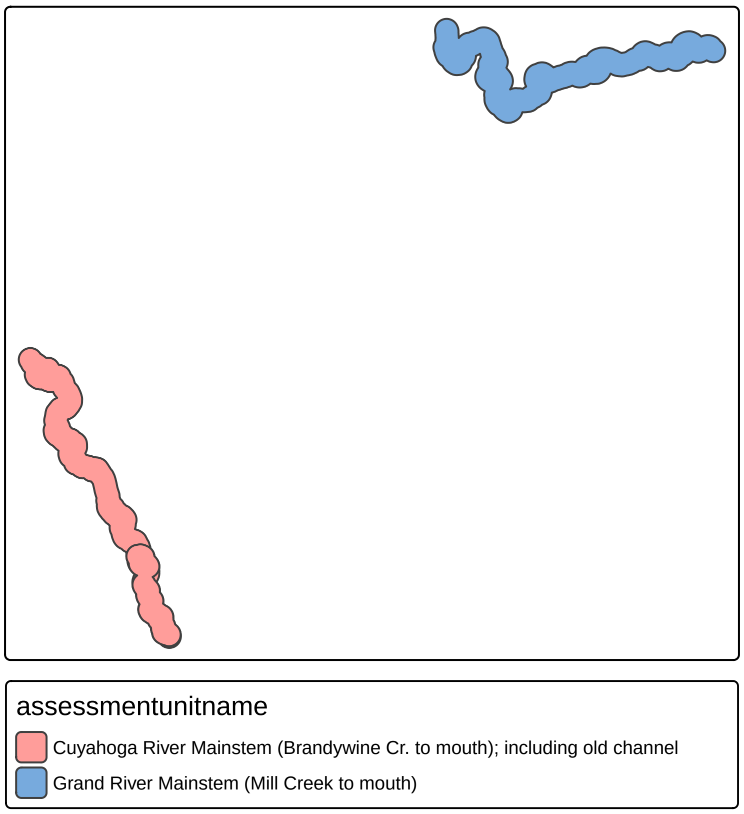
Buffers

What's a buffer?

Break down this code

```
buffs <- sf::st_buffer(study.streams, dist = 1000)

tm_shape(buffs) + tm_polygons(fill = "assessmentunitname")
```


assessmentunitname
Cuyahoga River Mainstem (Brandywine Cr. to mouth); including old channel
Grand River Mainstem (Mill Creek to mouth)

Let's add some parks. There are two parks files in the `/data/ohio/` directory. One is a shapefile, one is a geopackage. What's the difference?

```
oh_parks_shp <- read_sf("../data/ohio/ohio_parks.shp")
oh_parks <- read_sf("../data/ohio/oh_parks.gpkg")
```

They're VERY similar, but there are some cases where they might not be the same. Think about when/where, and let's have a class discussion if you're not sure

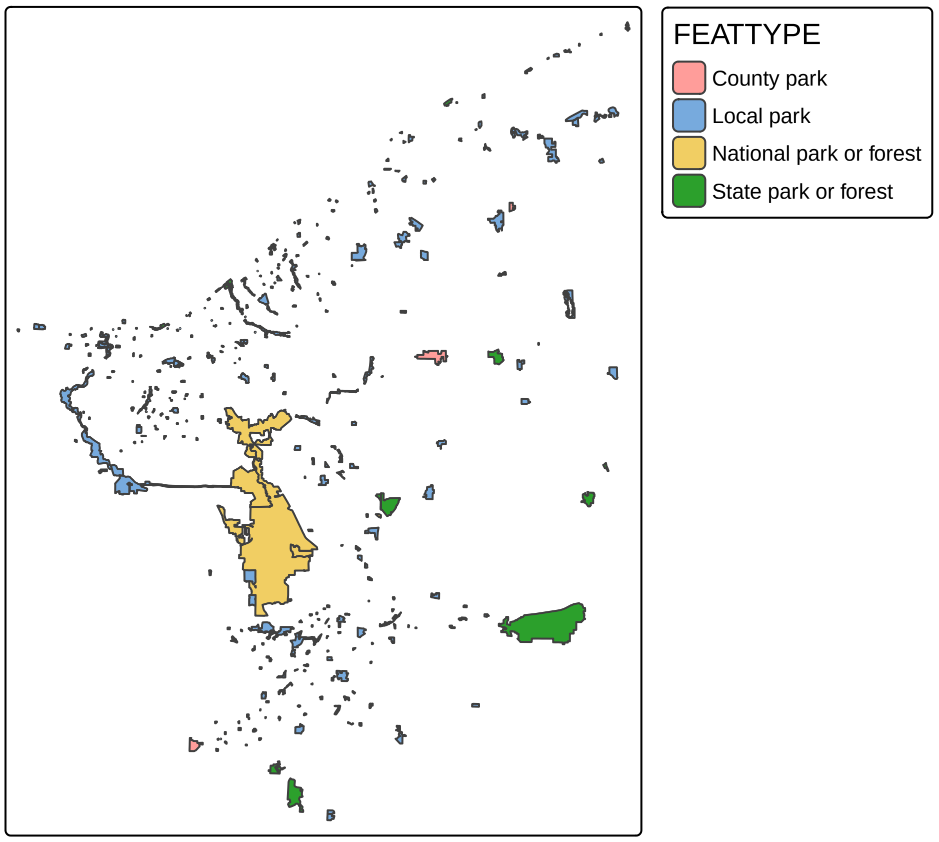
Let's subset the parks to our study area. Don't forget - we need to reproject first!

```
oh_parks_p <-  sf::st_transform(oh_parks, 6346)

oh_parks_p_studyarea <- sf::st_intersection(oh_parks_p, study.area_p)
```

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

```
tm_shape(oh_parks_p_studyarea) + tm_polygons(fill = "FEATTYPE")
```



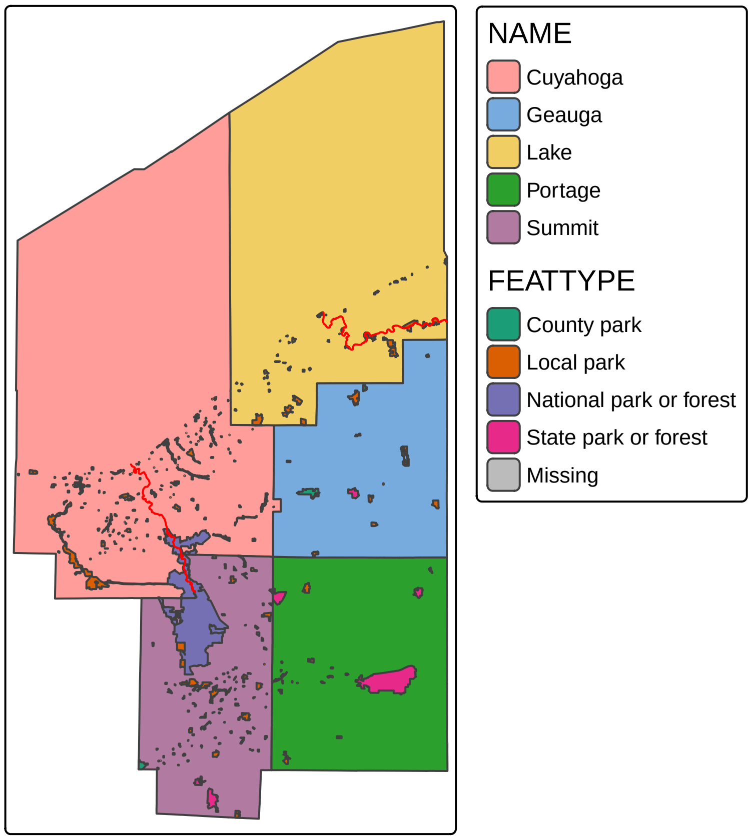
Let's do some more layering in a map - we can even change the palette we want to use!

```r
tm_shape(study.area_p) + tm_polygons(fill = "NAME") +
  tm_shape(oh_parks_p_studyarea) + tm_polygons(fill = "FEATTYPE", palette = "brewer.dark2") +
  tm_shape(study.streams) + tm_lines(col = "red")
```

```
##

## -- tmap v3 code detected -------------------------------------------------------

## [v3->v4] ‘tm_tm_polygons()‘: migrate the argument(s) related to the scale of
## the visual variable ‘fill‘ namely ’palette’ (rename to ’values’) to fill.scale
## = tm_scale(<HERE>).
```



```
## Perhaps a bit on the ugly side, but it gets the point across
```

YOUR TASK

I have given you all of the tools to complete the following items:

- find ALL of the Ohio parks within 1km of a 303d stream
- Make a map (using `tmap`) of just those parks (not all parks), overlaid on a Ohio county map
- Add a color to the parks based on the type of park (like we did)

Bonus work

- How many counties do NOT have a 303d stream in them?
- What county intersects the most parks? The most streams? The most 303d streams?