Lab 1: Intro to Attribute and Spatial Analysis in R

Bailey McNichol

9/17/2021

Load and examine the data

Select attributes that we specify

```
d.counties %>% dplyr::select(GEOID10, ALAND10) %>% head()
```

```
## Simple feature collection with 6 features and 2 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -79.38264 ymin: 37.69574 xmax: -76.95493 ymax: 40.72605
## Geodetic CRS:  WGS 84
## # A tibble: 6 x 3
##   GEOID10    ALAND10                                                  geometry
##   <chr>        <dbl>                                        <MULTIPOLYGON [°]>
## 1 51540     26517362 (((-78.47071 38.04872, -78.47033 38.04801, -78.47009 38.04~
## 2 51510     38919733 (((-77.06247 38.79497, -77.06268 38.79504, -77.06273 38.79~
## 3 51530     17362236 (((-79.36681 37.72723, -79.36687 37.72744, -79.36704 37.72~
## 4 51600     16159465 (((-77.31427 38.86701, -77.31414 38.867, -77.31398 38.8669~
## 5 42021   1782819861 (((-79.03392 40.3165, -79.03359 40.31674, -79.0322 40.3173~
## 6 42001   1343342705 (((-77.46596 39.85977, -77.46596 39.85983, -77.46599 39.86~
```

Remove unwanted attributes with "-"

```
d.counties %>% dplyr::select(-NAME10) %>% head()
```

```
## Simple feature collection with 6 features and 19 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -79.38264 ymin: 37.69574 xmax: -76.95493 ymax: 40.72605
## Geodetic CRS:  WGS 84
## # A tibble: 6 x 20
##   OBJECTID STATEFP10 COUNTYFP10 COUNTYNS10 GEOID10 NAMELSAD10   LSAD10 CLASSFP10
##      <int> <chr>     <chr>      <chr>      <chr>   <chr>        <chr>  <chr>
## 1        1 51        540        01789068   51540   Charlottesv~ 25     C7
## 2        2 51        510        01498415   51510   Alexandria ~ 25     C7
## 3        3 51        530        01498417   51530   Buena Vista~ 25     C7
## 4        4 51        600        01789070   51600   Fairfax city 25     C7
## 5        5 42        021        01213662   42021   Cambria Cou~ 06     H1
## 6        6 42        001        01213656   42001   Adams County 06     H1
```

```
## # ... with 12 more variables: MTFCC10 <chr>, CSAFP10 <chr>, CBSAFP10 <chr>,
## #   METDIVFP10 <chr>, FUNCSTAT10 <chr>, ALAND10 <dbl>, AWATER10 <dbl>,
## #   INTPTLAT10 <chr>, INTPTLON10 <chr>, Shape_Leng <dbl>, Shape_Area <dbl>,
## #   geometry <MULTIPOLYGON [°]>
```

Specify ranges that we want to keep (or not):

```
d.counties %>% dplyr::select(GEOID10:CLASSFP10) %>% head()
```

```
## Simple feature collection with 6 features and 5 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -79.38264 ymin: 37.69574 xmax: -76.95493 ymax: 40.72605
## Geodetic CRS:  WGS 84
## # A tibble: 6 x 6
##   GEOID10 NAME10          NAMELSAD10          LSAD10 CLASSFP10                  geometry
##   <chr>   <chr>           <chr>               <chr>  <chr>            <MULTIPOLYGON [°]>
## 1 51540   Charlottesville Charlottesville city 25     C7        (((-78.47071 38.04872, -~
## 2 51510   Alexandria      Alexandria city     25     C7        (((-77.06247 38.79497, -~
## 3 51530   Buena Vista     Buena Vista city    25     C7        (((-79.36681 37.72723, -~
## 4 51600   Fairfax         Fairfax city        25     C7        (((-77.31427 38.86701, -~
## 5 42021   Cambria         Cambria County      06     H1        (((-79.03392 40.3165, -7~
## 6 42001   Adams           Adams County        06     H1        (((-77.46596 39.85977, -~
```

```
d.counties %>% dplyr::select(-(GEOID10:CLASSFP10)) %>% head()
```

```
## Simple feature collection with 6 features and 15 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -79.38264 ymin: 37.69574 xmax: -76.95493 ymax: 40.72605
## Geodetic CRS:  WGS 84
## # A tibble: 6 x 16
##   OBJECTID STATEFP10 COUNTYFP10 COUNTYNS10 MTFCC10 CSAFP10 CBSAFP10 METDIVFP10
##      <int> <chr>     <chr>      <chr>      <chr>   <chr>   <chr>    <chr>
## 1        1 51        540        01789068   G4020   NA      16820    NA
## 2        2 51        510        01498415   G4020   548     47900    47894
## 3        3 51        530        01498417   G4020   NA      NA       NA
## 4        4 51        600        01789070   G4020   548     47900    47894
## 5        5 42        021        01213662   G4020   NA      27780    NA
## 6        6 42        001        01213656   G4020   564     23900    NA
## # ... with 8 more variables: FUNCSTAT10 <chr>, ALAND10 <dbl>, AWATER10 <dbl>,
## #   INTPTLAT10 <chr>, INTPTLON10 <chr>, Shape_Leng <dbl>, Shape_Area <dbl>,
## #   geometry <MULTIPOLYGON [°]>
```

```
d.counties %>% dplyr::select(starts_with("C"))
```

```
## Simple feature collection with 207 features and 5 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -81.01449 ymin: 36.55035 xmax: -74.16468 ymax: 44.09697
## Geodetic CRS:  WGS 84
```

```
## # A tibble: 207 x 6
##    COUNTYFP10 COUNTYNS10 CLASSFP10 CSAFP10 CBSAFP10                     geometry
##    <chr>      <chr>      <chr>     <chr>   <chr>              <MULTIPOLYGON [°]>
##  1 540        01789068   C7        NA      16820    (((-78.47071 38.04872, -78.~
##  2 510        01498415   C7        548     47900    (((-77.06247 38.79497, -77.~
##  3 530        01498417   C7        NA      NA       (((-79.36681 37.72723, -79.~
##  4 600        01789070   C7        548     47900    (((-77.31427 38.86701, -77.~
##  5 021        01213662   H1        NA      27780    (((-79.03392 40.3165, -79.0~
##  6 001        01213656   H1        564     23900    (((-77.46596 39.85977, -77.~
##  7 061        01213672   H1        NA      26500    (((-78.15023 40.17464, -78.~
##  8 035        01214721   H1        558     30820    (((-78.05358 41.27373, -78.~
##  9 093        01213681   H1        NA      14100    (((-76.55874 40.93904, -76.~
## 10 117        01209189   H1        NA      NA       (((-77.21965 41.99978, -77.~
## # ... with 197 more rows
```

Group data - create a new attribute that calculates the land area of all counties in each state.

```r
d.counties %>% group_by(STATEFP10) %>% mutate(stateLandArea = sum(ALAND10))
```

```
## Simple feature collection with 207 features and 21 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -81.01449 ymin: 36.55035 xmax: -74.16468 ymax: 44.09697
## Geodetic CRS:  WGS 84
## # A tibble: 207 x 22
## # Groups:   STATEFP10 [7]
##    OBJECTID STATEFP10 COUNTYFP10 COUNTYNS10 GEOID10 NAME10   NAMELSAD10   LSAD10
##  *    <int> <chr>     <chr>      <chr>      <chr>   <chr>    <chr>        <chr>
##  1        1 51        540        01789068   51540   Charlot~ Charlottesv~ 25
##  2        2 51        510        01498415   51510   Alexand~ Alexandria ~ 25
##  3        3 51        530        01498417   51530   Buena V~ Buena Vista~ 25
##  4        4 51        600        01789070   51600   Fairfax  Fairfax city 25
##  5        5 42        021        01213662   42021   Cambria  Cambria Cou~ 06
##  6        6 42        001        01213656   42001   Adams    Adams County 06
##  7        7 42        061        01213672   42061   Hunting~ Huntingdon ~ 06
##  8        8 42        035        01214721   42035   Clinton  Clinton Cou~ 06
##  9        9 42        093        01213681   42093   Montour  Montour Cou~ 06
## 10       10 42        117        01209189   42117   Tioga    Tioga County 06
## # ... with 197 more rows, and 14 more variables: CLASSFP10 <chr>,
## #   MTFCC10 <chr>, CSAFP10 <chr>, CBSAFP10 <chr>, METDIVFP10 <chr>,
## #   FUNCSTAT10 <chr>, ALAND10 <dbl>, AWATER10 <dbl>, INTPTLAT10 <chr>,
## #   INTPTLON10 <chr>, Shape_Leng <dbl>, Shape_Area <dbl>,
## #   geometry <MULTIPOLYGON [°]>, stateLandArea <dbl>
```

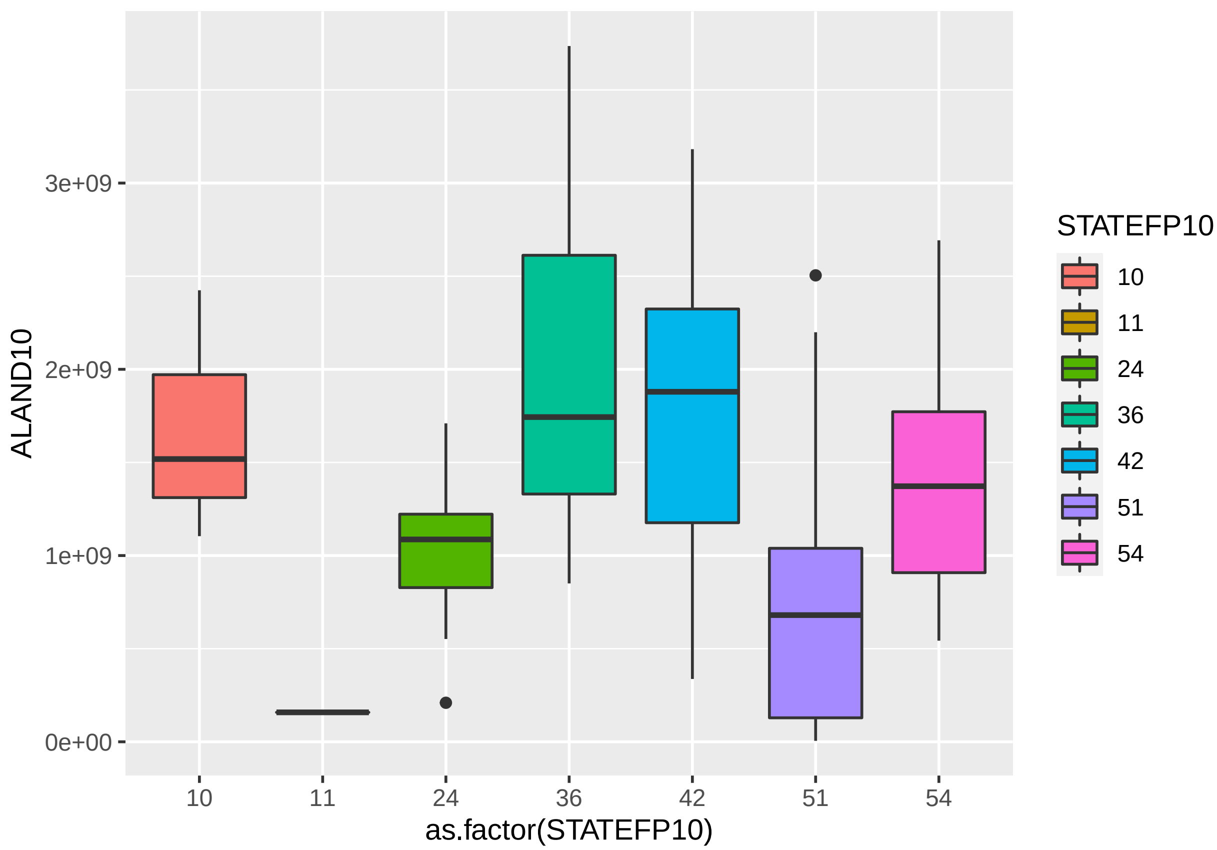
Convert sf data frame to a tibble, then remove the geometry before performing the group_by and summarise functions.

```r
d.counties %>%
  # this line converts the data because of wonky geometry
  as_tibble() %>% dplyr::select(-geometry) %>%
  group_by(STATEFP10) %>%
  summarise(stateLandArea = sum(ALAND10))
```

```
## # A tibble: 7 x 2
##   STATEFP10 stateLandArea
##   <chr>             <dbl>
## 1 10           5046703785
## 2 11            158114680
## 3 24          25141638381
## 4 36          40599407643
## 5 42          78174288199
## 6 51          69471293533
## 7 54          20781223859
```

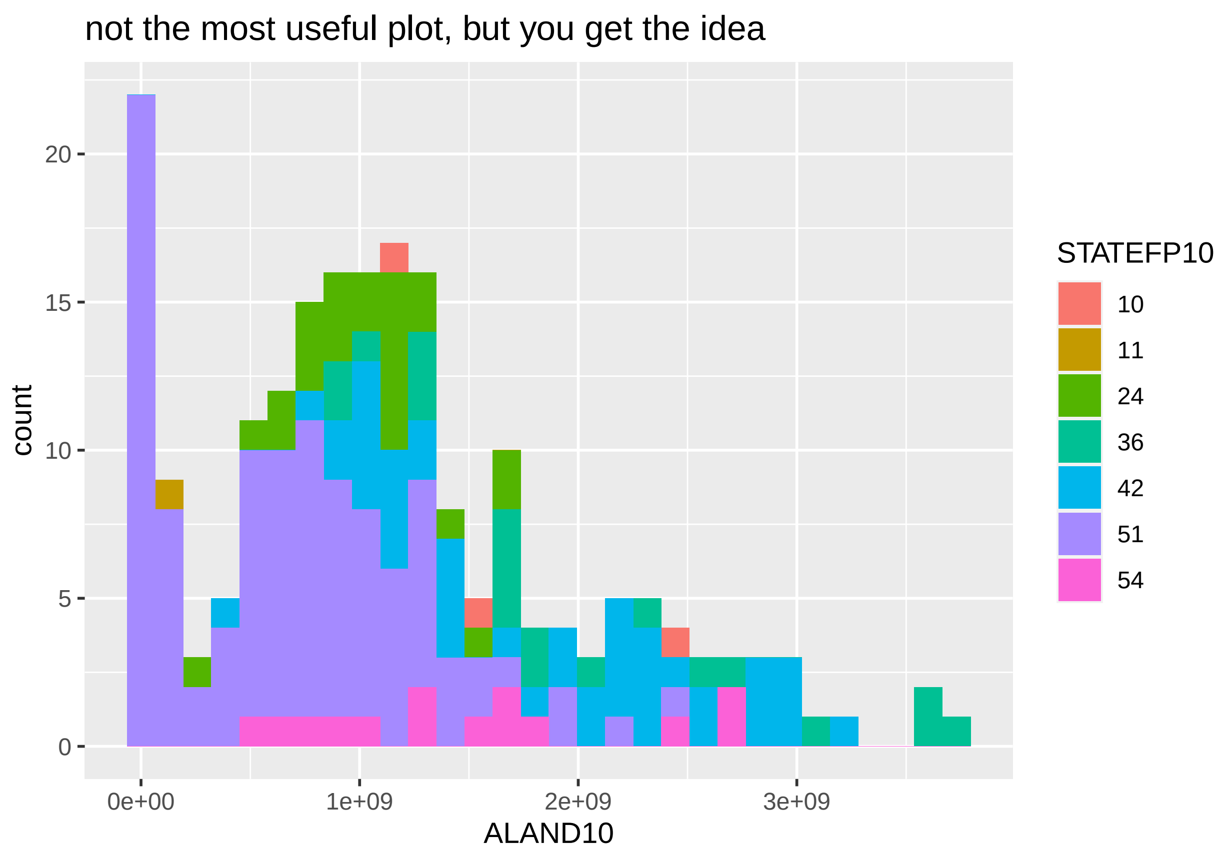
Use grouping in a plot

```
d.counties %>%
  ggplot(., aes(x = as.factor(STATEFP10), y = ALAND10)) +
  geom_boxplot(aes(fill = STATEFP10))
```




```
d.counties %>%
  ggplot(., aes(x = ALAND10)) +
  geom_histogram(aes(fill = STATEFP10)) +
  labs(title = "not the most useful plot, but you get the idea")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Examine coordinate reference system (CRS) for each file

```
d.counties %>% sf::st_crs()
```

```
## Coordinate Reference System:
##   User input: WGS 84
##   wkt:
## GEOGCRS["WGS 84",
##     DATUM["World Geodetic System 1984",
##         ELLIPSOID["WGS 84",6378137,298.257223563,
##             LENGTHUNIT["metre",1]]],
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433]],
##     CS[ellipsoidal,2],
##         AXIS["latitude",north,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["longitude",east,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433]],
##     ID["EPSG",4326]]
```

```
d.stations %>% sf::st_crs()
```

```
## Coordinate Reference System:
##   User input: WGS 84
##   wkt:
## GEOGCRS["WGS 84",
##     DATUM["World Geodetic System 1984",
##         ELLIPSOID["WGS 84",6378137,298.257223563,
##             LENGTHUNIT["metre",1]]],
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433]],
##     CS[ellipsoidal,2],
##         AXIS["latitude",north,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["longitude",east,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433]],
##     ID["EPSG",4326]]
```

Formally check CRS are the same

```r
d.counties %>% sf::st_crs() == d.stations %>% sf::st_crs()
```

```
## [1] TRUE
```

Subset data to only include counties in Delaware

```r
del.counties <- d.counties %>% dplyr::filter(STATEFP10 == 10)
```

Perform a *spatial intersection* to find all of the monitoring stations within our Delaware subset

```r
del.stations <- sf::st_intersection(d.stations, del.counties)
```

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

Examine data and plot

```r
glimpse(del.stations)
```

```
## Rows: 2
## Columns: 32
## $ OBJECTID   <int> 2, 1
## $ MAP_ID     <int> 2, 1
## $ USGS_STATI <int> 1488500, 1487000
## $ STATION_NA <chr> "MARSHYHOPE CREEK NEAR ADAMSVILLE, DE", "NANTICOKE RIVER NE~
## $ MAJOR_WATE <chr> "Eastern Shore", "Eastern Shore"
## $ Drainage_A <dbl> 46.79998, 75.39997
## $ START_DATE <int> 2005, 1998
```

```
## $ END_DATE   <int> 2018, 2018
## $ Lat        <dbl> 38.84969, 38.72833
## $ Long       <dbl> -75.67311, -75.56186
## $ STAID      <chr> "01488500", "01487000"
## $ OBJECTID.1 <int> 120, 122
## $ STATEFP10  <chr> "10", "10"
## $ COUNTYFP10 <chr> "001", "005"
## $ COUNTYNS10 <chr> "00217271", "00217269"
## $ GEOID10    <chr> "10001", "10005"
## $ NAME10     <chr> "Kent", "Sussex"
## $ NAMELSAD10 <chr> "Kent County", "Sussex County"
## $ LSAD10     <chr> "06", "06"
## $ CLASSFP10  <chr> "H1", "H1"
## $ MTFCC10    <chr> "G4020", "G4020"
## $ CSAFP10    <chr> NA, NA
## $ CBSAFP10   <chr> "20100", "42580"
## $ METDIVFP10 <chr> NA, NA
## $ FUNCSTAT10 <chr> "A", "A"
## $ ALAND10    <dbl> 1518196116, 2424432871
## $ AWATER10   <dbl> 549470508, 674204700
## $ INTPTLAT10 <chr> "+39.0970884", "+38.6775108"
## $ INTPTLON10 <chr> "-075.5029819", "-075.3354950"
## $ Shape_Leng <dbl> 269441.5, 302135.9
## $ Shape_Area <dbl> 3437654275, 5092675716
## $ geometry   <POINT [°]> POINT (-75.67311 38.8497), POINT (-75.56186 38.72834)
```

```
#plot(del.stations)
```

Calculation of the area of each county in Delaware using sf function

```
del.counties %>% st_area()
```

```
## Units: [m^2]
## [1] 2065913935 3096294979 1278231425
```

Lab 1 Tasks

Task 1: Basic data manipulation

1.1 For each county, calculate its land area as percentage of the total area (land + water) for that state.

I created a new tibble (d.counties.2) that groups data by state and then calculates the percent of state land area for each county (new variable = countyLandPercent). The column 'countyLandPercent' includes those values, and the last line is an easy check to make sure the percentages don't exceed 100% in each state.

```
d.counties.2 <- d.counties %>%
  # Group the data by state
        group_by(STATEFP10) %>%
  # Calculate the percent of state land area for each county (new variable = countyLandPercent)
        mutate(countyLandPercent = ((ALAND10+AWATER10)/sum(ALAND10+AWATER10))*100)
```

```r
# View values of the new variable - each county is a percent of its state
d.counties.2$countyLandPercent
```

```
##   [1] 3.425099e-02 5.164023e-02 2.266968e-02 2.095319e-02 2.272791e+00
##   [6] 1.709716e+00 2.913813e+00 2.938937e+00 4.336003e-01 3.725438e+00
##  [11] 3.332042e+00 1.327219e+00 3.224942e+00 2.984118e+00 1.820300e+00
##  [16] 3.781132e+00 1.606080e+00 4.093119e+00 2.836778e+00 6.673276e-03
##  [21] 3.317986e-02 5.075052e-02 6.689070e-02 1.709982e+00 3.332642e+00
##  [26] 5.182746e+00 2.626984e+00 2.727180e+00 2.565009e+00 1.435354e+00
##  [31] 2.487746e+00 1.482666e+00 3.543825e+00 4.075611e+00 3.804252e+00
##  [36] 1.727018e+00 1.269331e+00 1.358880e+00 1.041244e+00 1.187780e+00
##  [41] 1.523347e+00 1.828953e+00 1.564680e+00 2.969511e+00 1.803146e+00
##  [46] 3.541834e+00 1.305474e+00 2.532262e+00 1.289411e+00 3.646790e+00
##  [51] 3.223577e+00 2.459499e+00 2.726639e+00 2.733689e+00 4.373789e+00
##  [56] 9.614722e-01 8.445721e-01 5.834657e-02 3.465532e+00 7.419935e-01
##  [61] 5.379510e+00 5.601020e+00 4.428795e+00 2.542585e+00 9.024262e+00
##  [66] 4.853868e-02 1.660449e+00 9.345896e-01 3.843366e+00 5.285677e+00
##  [71] 3.368045e+00 4.919841e+00 3.767094e+00 4.020949e+00 5.495401e+00
##  [76] 4.986506e+00 3.042579e+00 4.100818e+00 5.562218e+00 9.083001e+00
##  [81] 3.962989e+00 3.236452e+00 2.118897e+00 6.287165e+00 6.402505e+00
##  [86] 5.882284e+00 1.169676e+01 5.964835e+00 7.258897e+00 2.628812e+00
##  [91] 2.851648e+00 3.226186e+00 3.648959e+00 4.086333e+00 4.116513e+00
##  [96] 7.923660e+00 4.245091e+00 6.160370e+00 3.104231e+00 5.345834e+00
## [101] 2.326012e+00 3.876758e+00 8.690805e+00 7.784469e+00 5.229245e+00
## [106] 4.088663e+00 8.005849e+00 1.291543e+01 3.993519e+00 1.272372e+01
## [111] 8.089803e+00 8.670534e+00 4.738058e+00 1.086279e+00 1.657545e-01
## [116] 1.498237e+00 1.117557e+00 5.238553e-01 9.536404e-01 3.207791e+01
## [121] 2.152106e+00 4.807246e+01 1.984963e+01 2.781503e+00 2.042525e+00
## [126] 1.356322e+00 1.000000e+02 2.787317e-02 1.036015e+00 1.693348e+00
## [131] 1.387241e+00 1.822618e+00 1.203328e+00 2.608243e-02 4.550583e-01
## [136] 1.181345e+00 9.678972e-01 9.535944e-01 9.409742e-01 1.048594e+00
## [141] 2.424269e+00 1.693799e+00 1.163443e+00 7.460988e-01 1.443563e+00
## [146] 9.687113e-01 1.705633e+00 1.739911e+00 1.458848e+00 6.268619e-01
## [151] 7.223464e-01 9.548623e-01 8.388300e-01 5.954935e-01 3.509154e-02
## [156] 1.583550e+00 8.462759e-03 3.216849e-01 1.893049e-02 1.277187e+00
## [161] 1.792432e+00 7.036647e-01 2.849077e+00 7.738371e-02 8.703496e-02
## [166] 8.758898e-01 2.085391e-01 3.988945e-01 2.618377e-01 5.984245e-01
## [171] 8.424541e-03 1.594286e+00 7.236264e-01 1.103815e+00 2.568288e+00
## [176] 1.432391e-01 3.241368e+00 1.560240e-01 1.211365e+00 2.304274e-02
## [181] 1.000770e+00 1.089362e+00 2.174228e+00 6.818976e-01 7.725615e-01
## [186] 1.383066e+00 7.176759e-01 3.089474e-02 1.388398e+00 1.948303e+00
## [191] 1.074634e+00 1.581883e+00 1.598722e+00 1.055185e+00 8.176652e-01
## [196] 1.196987e+00 8.915118e-01 1.171405e+00 3.606497e-02 1.432592e+00
## [201] 1.299628e+00 2.006449e+00 8.412194e-01 2.655076e+00 1.784796e+00
## [206] 3.066030e-02 1.146047e+00
```

```r
# Check on the operation to make sure we haven't made a mistake - new variable sums to 700% (100% for e
sum(d.counties.2$countyLandPercent)
```

```
## [1] 700
```

1.2 For each state, find the county that has the largest proportion of its land as water (water area / total area)

I first removed the weird geometry, then grouped data by county, calculated the proportion of water per county, then grouped by state to find the county with the largest proportion within each state. The final output lists county in each state with the largest proportion of land as water (proportions listed in 'countyWaterProp').

```
       d.counties %>%
# First, remove weird geometry from the tibble
        as_tibble() %>% dplyr::select(-geometry) %>%
# Group data by county
        group_by(NAME10) %>%
# Then, calculate the proportion of area occupied by water for each county
        mutate(totalArea = ALAND10 + AWATER10,
          countyWaterProp = AWATER10/totalArea)  %>%
# Then, group data by state
        group_by(STATEFP10) %>%
# Filter to only include the max county per state
        slice(which.max(countyWaterProp)) %>%
# Select to only include some columns in output
        dplyr::select(c(STATEFP10,NAME10,AWATER10,totalArea,countyWaterProp))
```

```
## # A tibble: 7 x 5
## # Groups:   STATEFP10 [7]
##   STATEFP10 NAME10                 AWATER10  totalArea countyWaterProp
##   <chr>     <chr>                     <dbl>      <dbl>           <dbl>
## 1 10        Kent                  549470508 2067666624         0.266
## 2 11        District of Columbia   18884970  176999650         0.107
## 3 24        St. Mary's           1054309969 1979402010         0.533
## 4 36        Cayuga                445707605 2236897583         0.199
## 5 42        Dauphin                85809611 1445674118         0.0594
## 6 51        Poquoson              163451700  203121542         0.805
## 7 54        Jefferson               5274680  548225759         0.00962
```

1.3 Count the number of counties in each state

I first removed the weird geometry, then grouped the data by state, and then counted the number of rows per state with n() to get the number of counties in each state. The output lists these counts.

```
# Each row represents a county within a state, so we can remove the weird geometry, group the data by s
d.counties %>%
        as_tibble() %>% dplyr::select(-geometry) %>%
        group_by(STATEFP10) %>%
        summarise(NumberCounties = n())
```

```
## # A tibble: 7 x 2
##   STATEFP10 NumberCounties
##   <chr>              <int>
## 1 10                     3
## 2 11                     1
## 3 24                    24
## 4 36                    20
## 5 42                    43
## 6 51                   102
## 7 54                    14
```

1.4 Which station has the shortest name (STATION_NA) in the study area?

I examined the lengths of character strings to determine which station name was shortest. I found that the shortest name, ABRAM CREEK AT OAKMONT, WV, had 26 characters.

```
# Determine the minimum character string length (name) for Station Name in the d.stations dataset - 26
min(str_length(d.stations$STATION_NA))
```

```
## [1] 26
```

```
# Determine which position has that minimum (which index) - 105
which.min(str_length(d.stations$STATION_NA))
```

```
## [1] 105
```

```
# Index the data to determine which station is in the 105th position
d.stations$STATION_NA[105]
```

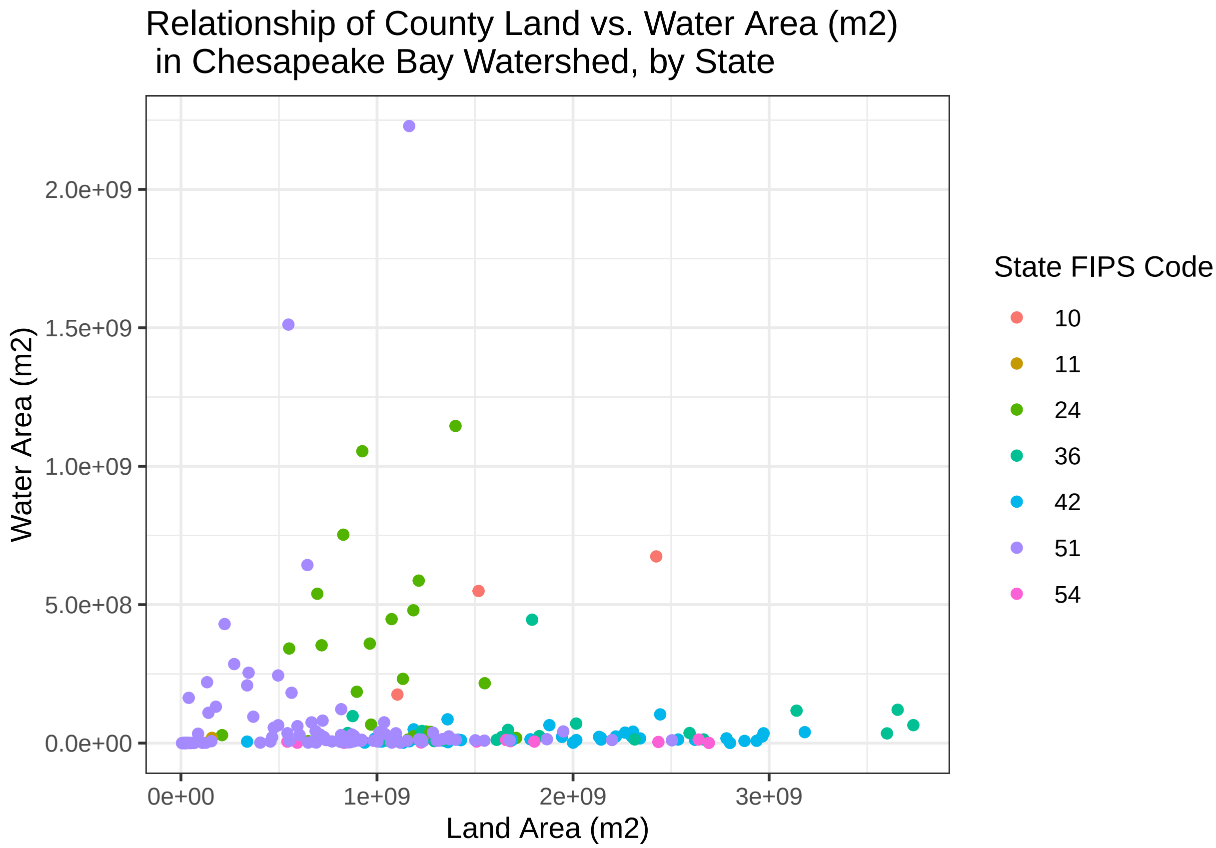
```
## [1] "ABRAM CREEK AT OAKMONT, WV"
```

Task 2: Plotting attribute data

Label your axes properly and give each plot a title

2.1 Make a scatterplot showing the relationship between land area and water area for each county. Color each point using the state variable

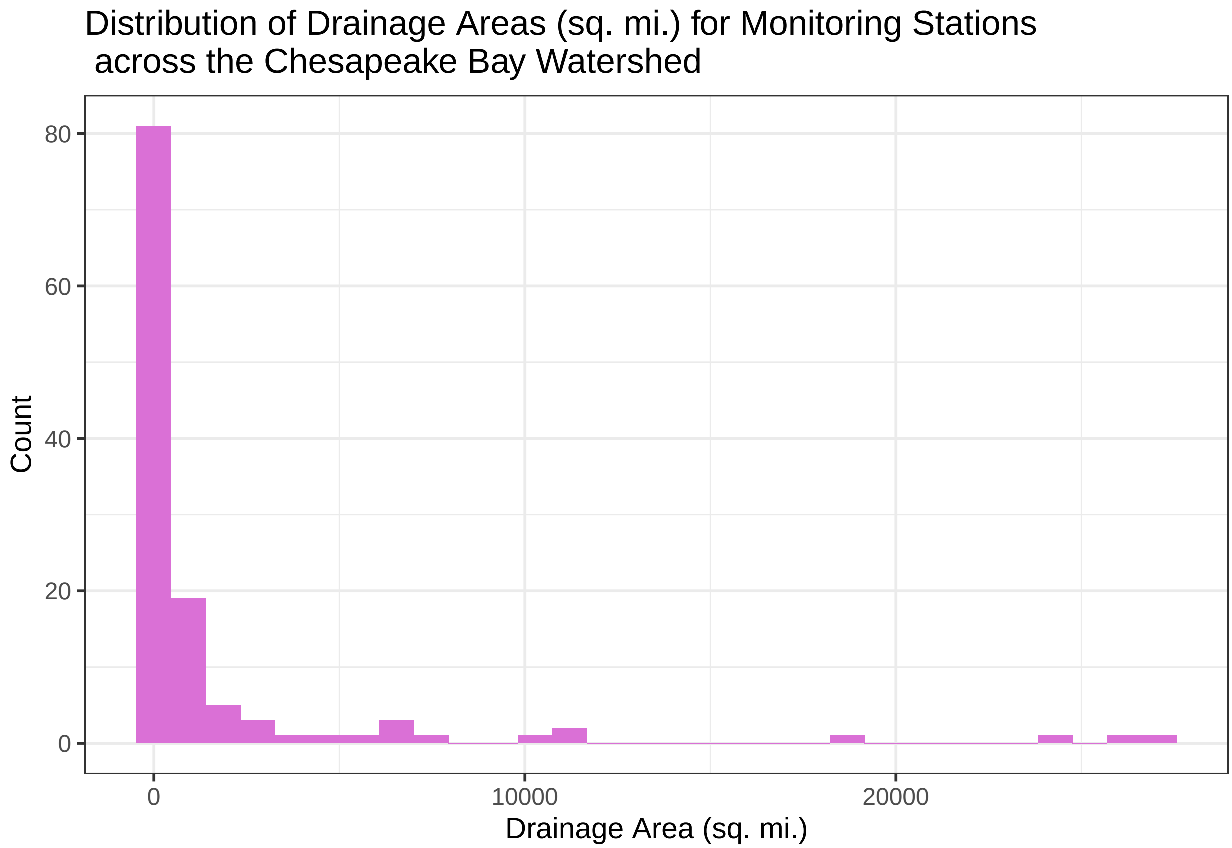
```
ggplot(data = d.counties) +
    geom_point(mapping = aes(x = ALAND10, y =  AWATER10,
    color = STATEFP10)) +
    theme_bw() +
    labs(x = "Land Area (m2)", y = "Water Area (m2)",
    title = "Relationship of County Land vs. Water Area (m2) \n in Chesapeake Bay Watershed, by State",
```



2.2 Make a histogram of drainage area (Drainage_A) for all monitoring stations

```
ggplot(data = d.stations) +
    geom_histogram(mapping = aes(x = Drainage_A), fill = "orchid") +
    theme_bw() +
    labs(x = "Drainage Area (sq. mi.)", y = "Count",
    title = "Distribution of Drainage Areas (sq. mi.) for Monitoring Stations \n across the Chesapeake B
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

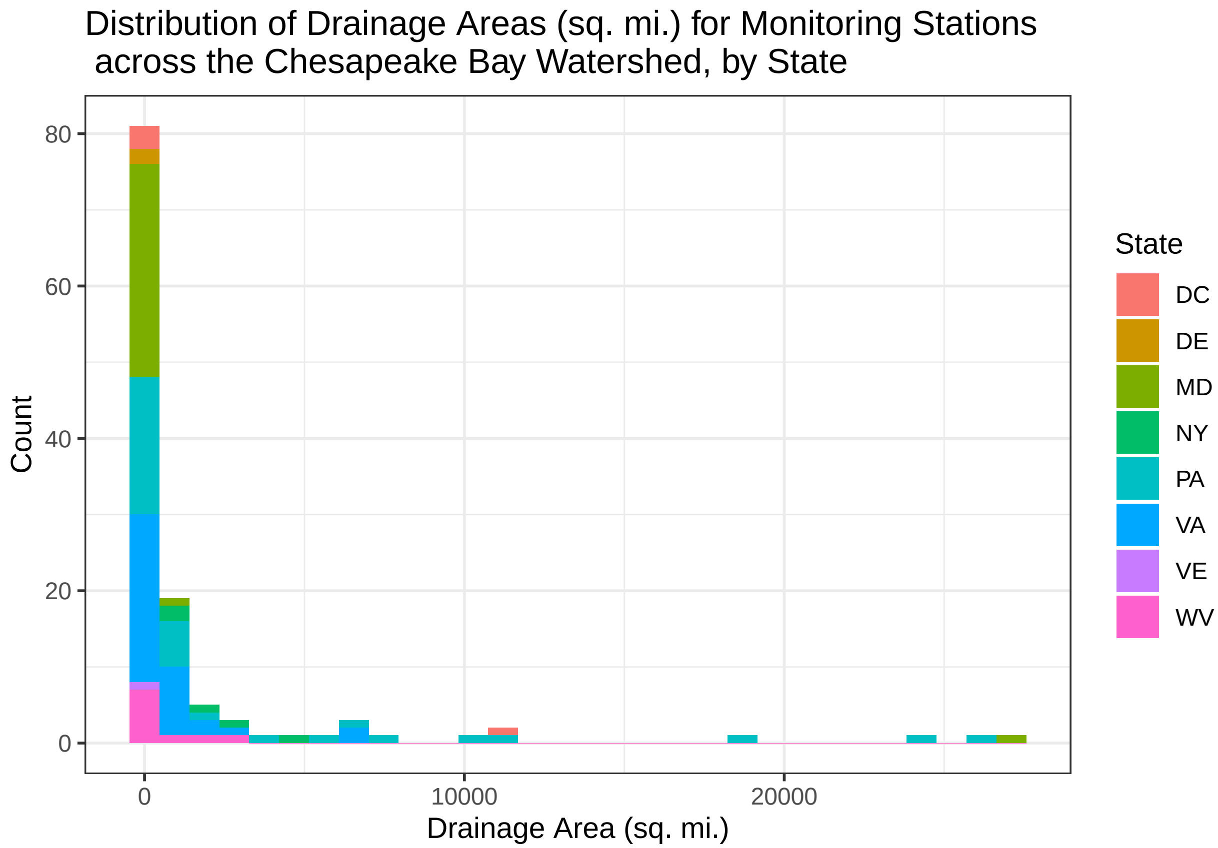


2.3 Make a similar histogram, this time of drainage area (Drainage_A) for all monitoring stations. Color each point using the state variable.

To accomplish this, I created a state variable (STATE) within d.stations by extracting the two-letter state abbreviation from the Station Names. I then incorporated this into the histogram as a fill variable to color code by state.

```
# First, we need to make a new variable for state - we can do this using the last two characters of the
n_last <- 2
d.stations$STATE <- substr(d.stations$STATION_NA,
            nchar(d.stations$STATION_NA) - n_last + 1, nchar(d.stations$STATION_NA))
# Now, we can make the histogram, coloring stations by state
ggplot(data = d.stations) +
    geom_histogram(mapping = aes(x = Drainage_A, fill = STATE)) +
    theme_bw() +
    labs(x = "Drainage Area (sq. mi.)", y = "Count", fill = "State",
    title = "Distribution of Drainage Areas (sq. mi.) for Monitoring Stations \n across the Chesapeake I
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Task 3: Write a function

3.1 Write a function that does the following:

A. Accepts a vector of arbitrary numbers, calculates the mean, median, maximum, and minimum of the vector

B. Sorts the vector

C. Returns a list of those values from A and the sorted vector from B

D. The function should only work with numeric values and print an error message if any other data type are found

I have written annotations throughout my function explaining what the various components are accomplishing.

```
my.fxn <- function(x) {
  # Command to abort running function if vector includes non-numeric values
  if(is.numeric(x) == "FALSE") {
        print("Value(s) in the supplied vector are not numeric")
          } else {
    # If data is numeric, then calculate the mean, median, max, and min
```

```
      mean <- mean(x)
      median <- median(x)
      maximum <- max(x)
      minimum <- min(x)
    # Also, sort the vector in ascending order
      sorted_vector <- sort(x)
    # Finally, give me an output that returns these quantities
      output = list(mean = mean,
                    median = median,
                    maximum = maximum,
                    minimum = minimum,
                    sorted_vector = sorted_vector)
      return(output)
      }
  }
```

Test it with the following vectors

`c(1, 0, -1), c(10, 100, 1000), c(.1, .001, 1e8), c("a", "b", "c")`

See the results of running the test vectors through my function below.

```
# Label the vectors to pass through my function
v1 <- c(1, 0, -1)
v2 <- c(10, 100, 1000)
v3 <- c(.1, .001, 1e8)
v4 <- c("a", "b", "c")

# Test the function
my.fxn(v1)
```

```
## $mean
## [1] 0
##
## $median
## [1] 0
##
## $maximum
## [1] 1
##
## $minimum
## [1] -1
##
## $sorted_vector
## [1] -1  0  1
```

```
my.fxn(v2)
```

```
## $mean
## [1] 370
##
```

```
## $median
## [1] 100
##
## $maximum
## [1] 1000
##
## $minimum
## [1] 10
##
## $sorted_vector
## [1]   10  100 1000
```

```
my.fxn(v3)
```

```
## $mean
## [1] 33333333
##
## $median
## [1] 0.1
##
## $maximum
## [1] 1e+08
##
## $minimum
## [1] 0.001
##
## $sorted_vector
## [1] 1e-03 1e-01 1e+08
```

```
my.fxn(v4)
```

```
## [1] "Value(s) in the supplied vector are not numeric"
```

Task 4: (slightly) more complex spatial analysis.

…Note, you may need to find supplementary data to help you with these tasks

4.1 Calculate the number of monitoring stations in each state

To accomplish this, I removed the weird geometry, then grouped by the STATE variable that I made above (when creating the second histogram with groups), and then counted the number of stations (each is a row) with summarize(). The output shows the state and these counts.

```
d.stations %>%
        # Remove geometry to avoid weird errors
        as_tibble() %>% dplyr::select(-geometry) %>%
        # Group by the State variable in d.stations defined above when making histogram
        group_by(STATE) %>%
        # Count the number of stations per state
        summarise(NumberStations = n())
```

```
## # A tibble: 8 x 2
##   STATE NumberStations
##   <chr>          <int>
## 1 DC                 4
## 2 DE                 2
## 3 MD                30
## 4 NY                 5
## 5 PA                34
## 6 VA                36
## 7 VE                 1
## 8 WV                10
```

4.2 Calculate the average size of counties in New York (that are also in this study area)

First, I removed the weird geometry, then filtered the data to create a subset only including counties in New York, and then I calculated the mean county area for New York counties using summarize(), which is displayed in the output below.

```
      d.counties %>%
# Remove weird geometry from data
            as_tibble() %>% dplyr::select(-geometry) %>%
# Filter to only include counties in New York using FIPS code
            dplyr::filter(STATEFP10 == 36) %>%
# calculate the average county area in m^2 (need to include both land and water area) with summarize(
            summarize(AverageSize = mean(ALAND10 + AWATER10))
```

```
## # A tibble: 1 x 1
##   AverageSize
##         <dbl>
## 1 2092187568.
```

4.3 Calculate which state has monitoring stations with the greatest average drainage area (Drainage_A)

I removed the weird geometry from d.stations, grouped by the state variable (STATE) that I previously defined, calculated the mean drainage area by state with summarize, and then used the which.max() function inside slice() to determine which state has monitoring stations with the greatest mean drainage area. We see in the output that this state is Pennsylvania with a mean area of 3549.196 square miles.

```
d.stations %>%
# Remove geometry to avoid weird errors
      as_tibble() %>% dplyr::select(-geometry) %>%
# Group by the State variable in d.stations defined above when making histogram
      group_by(STATE) %>%
# Calculate the average drainage area across monitoring stations within each state
      summarise(MeanDrainageArea = mean(Drainage_A)) %>%
# Use slice() and which.max() to determine which state has the greatest average drainage area (in squ
      slice(which.max(MeanDrainageArea))
```

```
## # A tibble: 1 x 2
##   STATE MeanDrainageArea
##   <chr>            <dbl>
## 1 PA               3549.
```

Questions

1. In using the intersection functions, are the following two statements equivalent? If not, explain how. Be sure to think about BOTH the spatial data structures AND the attribute data. Would your answer be different if we were using different types of data?

I have provided some description below of the differences between these two intersection statements. Overall, the answer that each statement gives you (i.e., the data that is merged) is the same, but the order in which the information is compiled is reversed. The first statement gives station information merged with the relevant county data, whereas the second statement gives county information merged with the relevant station data. The attribute data produced by both statements with the intersection function is analagous, as these variables are assumed to be constant spatially across geometries (point geometry with same number of features and fields). If the types of data were not consistent, this intersection would produce different results depending on the order of the elements in the statement (thus affecting the resulting geometry and structure).

```r
# This statement finds all of the monitoring stations in counties in Delaware, showing station informat
sf::st_intersection(d.stations, del.counties)
```

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

```
## Simple feature collection with 2 features and 32 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: -75.67311 ymin: 38.72834 xmax: -75.56186 ymax: 38.8497
## Geodetic CRS:  WGS 84
## # A tibble: 2 x 33
##   OBJECTID MAP_ID USGS_STATI STATION_NA         MAJOR_WATE Drainage_A START_DATE
## *    <int>  <int>      <int> <chr>              <chr>           <dbl>      <int>
## 1        2      2    1488500 MARSHYHOPE CREEK ~ Eastern S~       46.8       2005
## 2        1      1    1487000 NANTICOKE RIVER N~ Eastern S~       75.4       1998
## # ... with 26 more variables: END_DATE <int>, Lat <dbl>, Long <dbl>,
## #   STAID <chr>, STATE <chr>, OBJECTID.1 <int>, STATEFP10 <chr>,
## #   COUNTYFP10 <chr>, COUNTYNS10 <chr>, GEOID10 <chr>, NAME10 <chr>,
## #   NAMELSAD10 <chr>, LSAD10 <chr>, CLASSFP10 <chr>, MTFCC10 <chr>,
## #   CSAFP10 <chr>, CBSAFP10 <chr>, METDIVFP10 <chr>, FUNCSTAT10 <chr>,
## #   ALAND10 <dbl>, AWATER10 <dbl>, INTPTLAT10 <chr>, INTPTLON10 <chr>,
## #   Shape_Leng <dbl>, Shape_Area <dbl>, geometry <POINT [°]>
```

```r
# This statement also finds all of the monitoring stations in Delaware counties, but sorts the informat
sf::st_intersection(del.counties, d.stations)
```

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

```
## Simple feature collection with 2 features and 32 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: -75.67311 ymin: 38.72834 xmax: -75.56186 ymax: 38.8497
## Geodetic CRS:  WGS 84
## # A tibble: 2 x 33
##   OBJECTID STATEFP10 COUNTYFP10 COUNTYNS10 GEOID10 NAME10 NAMELSAD10    LSAD10
```

```
## *    <int> <chr>     <chr>      <chr>      <chr>   <chr>  <chr>         <chr>
## 1      122 10        005        00217269   10005   Sussex Sussex County 06
## 2      120 10        001        00217271   10001   Kent   Kent County   06
## # ... with 25 more variables: CLASSFP10 <chr>, MTFCC10 <chr>, CSAFP10 <chr>,
## #   CBSAFP10 <chr>, METDIVFP10 <chr>, FUNCSTAT10 <chr>, ALAND10 <dbl>,
## #   AWATER10 <dbl>, INTPTLAT10 <chr>, INTPTLON10 <chr>, Shape_Leng <dbl>,
## #   Shape_Area <dbl>, OBJECTID.1 <int>, MAP_ID <int>, USGS_STATI <int>,
## #   STATION_NA <chr>, MAJOR_WATE <chr>, Drainage_A <dbl>, START_DATE <int>,
## #   END_DATE <int>, Lat <dbl>, Long <dbl>, STAID <chr>, STATE <chr>,
## #   geometry <POINT [°]>
```

2. What did you find challenging in this lab? What was new?

Because I am not used to coding in the tidyverse(), I had a bit of a learning curve with the syntax and pipes, although it really is not that hard. But I really enjoyed it and am kicking myself for not just committing to using tidyr() more extensively earlier! It also took me longer than it should have to get the error message to print in the if else statement in my function… I also need to keep practicing my function writing! The main challenge for me is starting to visualize and interpret the geometries and overall structure of spatial data. I think I'll be able to continue connecting the dots as we begin more visualizations of these features.

3. What types of activities would you like to see in labs this semester?

I would like to learn the best ways to merge large datasets efficiently to perform spatial analyses, e.g., merging coordinate or other spatial reference data with the rest of your data (in my case, plant variables). I would also like to keep having opportunities to practice writing useful, but more complicated functions to iterate through tasks. I know you said we're going to be pretty minimal on using for loops, but they are often useful in my work and they take me *way* too long to code every time… Just more practice and opportunities to add a spatial component to other types of models analyses (e.g., accounting for location/spatial autocorrelation in a model predicting diversity across habitats).