

*Lab 01: An introduction to attribute and spatial analysis in R***Read the instructions COMPLETELY before starting the lab**

This lab builds on many of the discussions and exercises from class. This lab also builds on Chapters 1-3 from your textbook, as well as R for Data Science by Hadley Wickham and Garrett Grolemund (<https://r4ds.had.co.nz>)

Formatting your submission

This lab must be placed into a public repository on GitHub (www.github.com). Before the due date, submit **on Canvas** a link to the repository. I will then download your repositories and run your code. The code must be contained in either a .R script or a .Rmd markdown document. As I need to run your code, any data you use in the lab must be referenced using **relative path names**. Finally, answers to questions I pose in this document must also be in the repository at the time you submit your link to Canvas. They can be in a separate text file, or if you decide to use an RMarkdown document, you can answer them directly in the doc.

Exploratory data analysis

This lab uses two files from the /data/CBW directory of this course's main repository: 1. County_Boundaries.shp: A polygon file containing the boundaries for all counties in the Chesapeake Bay Watershed 2. Non-Tidal_Water_Quality_Monitoring_Stations_in_the_Cheseapeake_Bay.shp: point locations of non-tidal monitoring stations in the Chesapeake Bay Watershed

Step 1, load your packages and data

```
library(tidyverse)

## -- Attaching packages ----- tidyverse 1.3.1 --

## v ggplot2 3.3.5      v purrr   0.3.4
## v tibble  3.1.2      v dplyr   1.0.7
## v tidyr   1.1.3      v stringr 1.4.0
## v readr   1.4.0      v forcats 0.5.1

## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

library(ggplot2) #technically included in tidyverse
library(sf)

## Linking to GEOS 3.8.1, GDAL 3.2.1, PROJ 7.2.1
```

```
library(sp) #just in case
```

Next, load your data:

```
## note the ".." as opposed to "." <- need to go back one additional level from where this file is
p.counties <- "../data/CBW/County_Boundaries.shp"
p.stations <- "../data/CBW/Non-Tidal_Water_Quality_Monitoring_Stations_in_the_Chesapeake_Bay.shp"
```

```
d.counties <- sf::read_sf(p.counties)
d.stations <- sf::read_sf(p.stations)
```

```
glimpse(d.counties)
```

```
## Rows: 207
## Columns: 21
## $ OBJECTID    <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, ~
## $ STATEFP10   <chr> "51", "51", "51", "51", "42", "42", "42", "42", "42", "42", ~
## $ COUNTYFP10  <chr> "540", "510", "530", "600", "021", "001", "061", "035", "09~
## $ COUNTYNS10  <chr> "01789068", "01498415", "01498417", "01789070", "01213662", ~
## $ GEOID10     <chr> "51540", "51510", "51530", "51600", "42021", "42001", "4206~
## $ NAME10      <chr> "Charlottesville", "Alexandria", "Buena Vista", "Fairfax", ~
## $ NAMELSAD10  <chr> "Charlottesville city", "Alexandria city", "Buena Vista cit~
## $ LSAD10      <chr> "25", "25", "25", "25", "06", "06", "06", "06", "06", "06", ~
## $ CLASSFP10   <chr> "C7", "C7", "C7", "C7", "H1", "H1", "H1", "H1", "H1", "H1", ~
## $ MTFCC10     <chr> "G4020", "G4020", "G4020", "G4020", "G4020", "G4020", "G402~
## $ CSAFP10     <chr> NA, "548", NA, "548", NA, "564", NA, "558", NA, NA, NA, NA, ~
## $ CBSAFP10    <chr> "16820", "47900", NA, "47900", "27780", "23900", "26500", "~
## $ METDIVFP10  <chr> NA, "47894", NA, "47894", NA, NA, NA, NA, NA, NA, NA, N~
## $ FUNCSTAT10  <chr> "F", "F", "F", "F", "A", "A", "A", "A", "A", "A", "A", "A", ~
## $ ALAND10     <dbl> 26517362, 38919733, 17362236, 16159465, 1782819861, 1343342~
## $ AWATER10    <dbl> 52974, 1140371, 223855, 95054, 13680552, 8081576, 37883358, ~
## $ INTPTLAT10  <chr> "+38.0376579", "+38.8183429", "+37.7316634", "+38.8531833", ~
## $ INTPTLON10  <chr> "-078.4853806", "-077.0820263", "-079.3563746", "-077.29902~
## $ Shape_Leng  <dbl> 47968.96, 43943.77, 34310.52, 29395.95, 260532.87, 195653.0~
## $ Shape_Area  <dbl> 42902561, 66086698, 28163001, 26840867, 3109865228, 2297092~
## $ geometry    <MULTIPOLYGON [°]> MULTIPOLYGON (((-78.47071 3..., MULTIPOLYGON (~
```

```
glimpse(d.stations)
```

```
## Rows: 122
## Columns: 12
## $ OBJECTID    <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, ~
## $ MAP_ID      <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, ~
## $ USGS_STATI  <int> 1487000, 1488500, 1491000, 1491500, 1495000, 1502500, 15030~
## $ STATION_NA  <chr> "NANTICOKE RIVER NEAR BRIDGEVILLE, DE", "MARSHYHOPE CREEK N~
## $ MAJOR_WATE  <chr> "Eastern Shore", "Eastern Shore", "Eastern Shore", "Eastern~
## $ Drainage_A  <dbl> 75.39997, 46.79998, 112.99995, 85.19996, 51.59998, 519.9997~
## $ START_DATE  <int> 1998, 2005, 1985, 2005, 2005, 2005, 2006, 2005, 2006, 2005, ~
## $ END_DATE    <int> 2018, 2018, 2018, 2018, 2018, 2018, 2018, 2018, 2018, 2018, ~
## $ Lat         <dbl> 38.72833, 38.84969, 38.99719, 38.96681, 39.66758, 42.37778, ~
## $ Long        <dbl> -75.56186, -75.67311, -75.78581, -75.94306, -75.82558, -75.~
```

```
## $ STAID      <chr> "01487000", "01488500", "01491000", "01491500", "01495000", ~
## $ geometry   <POINT [°]> POINT (-75.56186 38.72834), POINT (-75.67311 38.8497)~
```

```
# check for validity
d.stations %>% sf::st_is_valid()
```

```
## [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [16] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [31] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [46] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [61] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [76] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [91] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [106] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [121] TRUE TRUE
```

```
d.counties %>% sf::st_is_valid() # returns false for one feature, so we need to fix
```

```
## [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [13] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [25] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [37] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [49] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [61] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [73] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [85] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [97] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [109] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [121] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [133] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [145] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [157] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [169] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [181] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [193] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [205] TRUE FALSE TRUE
```

```
# fix it "in place"
d.counties <- d.counties %>% sf::st_make_valid()
```

In class, we discussed how to use dplyr verbs such as *filter*, *select*, and *mutate*. There are some useful cheatsheets on the RStudio website to help with *dplyr*, *ggplot*, and other functions here: <https://www.rstudio.com/resources/cheatsheets/>

Let's start with the *select* function, which SELECTS 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      2.65e7 (((-78.47082 38.04893, -78.47086 38.04893, -78.47096 38.048~
## 2 51510      3.89e7 (((-77.06129 38.79457, -77.0612 38.79454, -77.06092 38.7944~
## 3 51530      1.74e7 (((-79.36668 37.7267, -79.36655 37.72627, -79.36653 37.7261~
## 4 51600      1.62e7 (((-77.31476 38.86701, -77.31534 38.86702, -77.31537 38.867~
## 5 42021      1.78e9 (((-79.03546 40.31539, -79.03611 40.31477, -79.0363 40.3145~
## 6 42001      1.34e9 (((-77.46594 39.85958, -77.46589 39.85924, -77.46586 39.859~
```

```
# head truncates the data.frame to the first n rows
```

Note that because we're using a spatial data frame in the `sf` package, the geometry is preserved, even though we didn't specify it. We can also get rid of attributes we DON'T WANT (but not the geometry attribute) using a `-` flag. For example:

```
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 NAME10 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 [°]>
```

We can also 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 NAME10 LSAD10 CLASSFP10 geometry
##   <chr> <chr> <chr> <chr> <chr> <MULTIPOLYGON [°]>
## 1 51540 Charlot~ Charlottes~ 25 C7 (((-78.47082 38.04893, -78.4708~
## 2 51510 Alexand~ Alexandria~ 25 C7 (((-77.06129 38.79457, -77.0612~
```

```
## 3 51530 Buena V~ Buena Vist~ 25 C7 (((-79.36668 37.7267, -79.36655~
## 4 51600 Fairfax Fairfax ci~ 25 C7 (((-77.31476 38.86701, -77.3153~
## 5 42021 Cambria Cambria Co~ 06 H1 (((-79.03546 40.31539, -79.0361~
## 6 42001 Adams Adams Coun~ 06 H1 (((-77.46594 39.85958, -77.4658~
```

```
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 (((-78.47082 38.04893, -78.~
## 2 510 01498415 C7 548 47900 (((-77.06129 38.79457, -77.~
## 3 530 01498417 C7 <NA> <NA> (((-79.36668 37.7267, -79.3~
## 4 600 01789070 C7 548 47900 (((-77.31476 38.86701, -77.~
## 5 021 01213662 H1 <NA> 27780 (((-79.03546 40.31539, -79.~
## 6 001 01213656 H1 564 23900 (((-77.46594 39.85958, -77.~
## 7 061 01213672 H1 <NA> 26500 (((-78.14963 40.1743, -78.1~
## 8 035 01214721 H1 558 30820 (((-78.05375 41.27349, -78.~
## 9 093 01213681 H1 <NA> 14100 (((-76.55799 40.93887, -76.~
## 10 117 01209189 H1 <NA> <NA> (((-77.21985 41.99978, -77.~
## # ... with 197 more rows
```

Grouping data

We can also “group” our data according to categorical data in our data.frames. This is useful if you want to create a function that works across the entire group. For example, we’ll create a new attribute the calculates the land area of all counties in each state.

```
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>
```

The above function is useful if you want to make calculations “in place” and use them in further row-by-row functions. However, we can further summarize our data such that we don’t see all the extra data not relevant to our query. Note that sometimes buggy geometry can affect normal dplyr functions, so the code below converts the sf data frame to a tibble, then removes the geometry before performing the `group_by` and `summarise` functions. This is an unnecessary step when using validated geometry, but can also speed up computation.

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

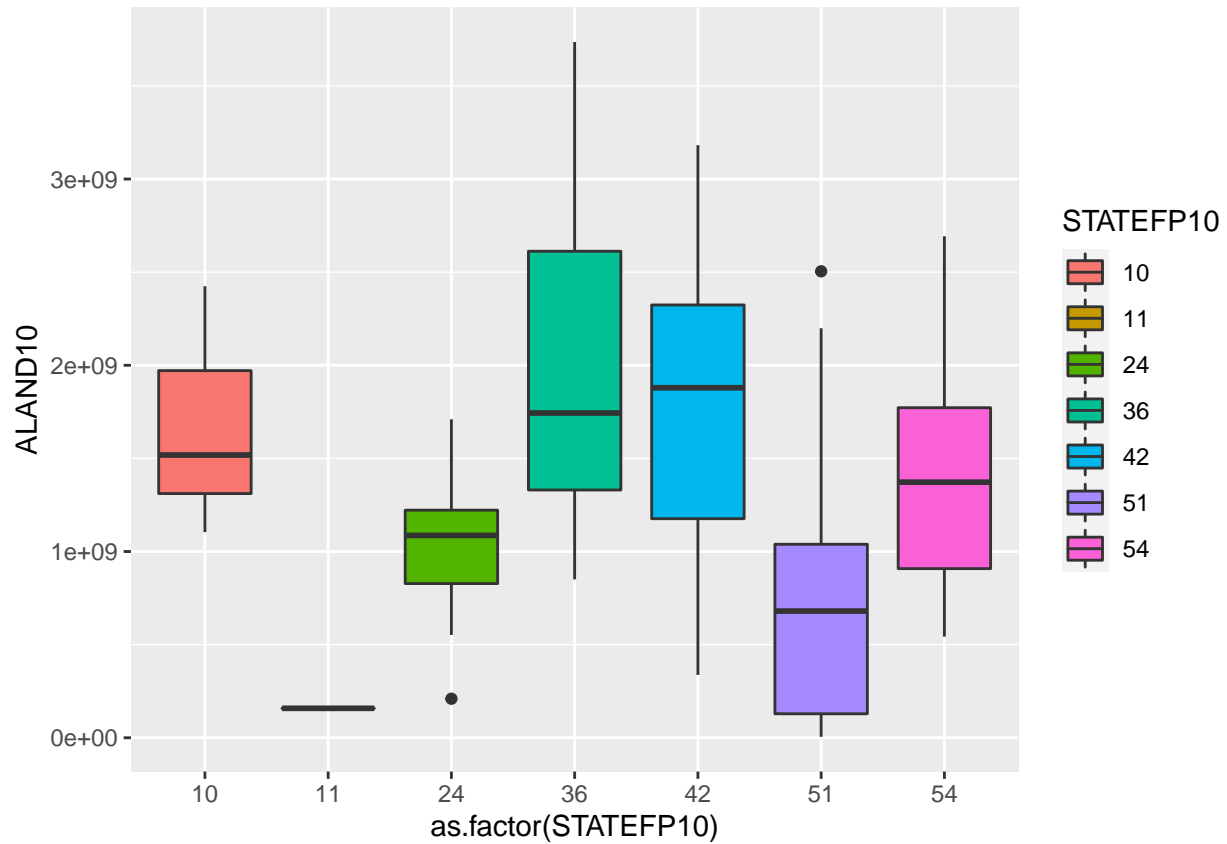
```
## # A tibble: 7 x 2
## STATEFP10 stateLandArea
## <chr> <dbl>
## 1 10 5046703785
## 2 11 158114680
## 3 24 25141638381
## 4 36 40599407643
## 5 42 78174288199
## 6 51 69471293533
## 7 54 20781223859
```

...and we’re left with a sum of all the land area in each state (by FIPS code)

A diversion into plots

We can also use grouping functions in our visualization. For example:

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

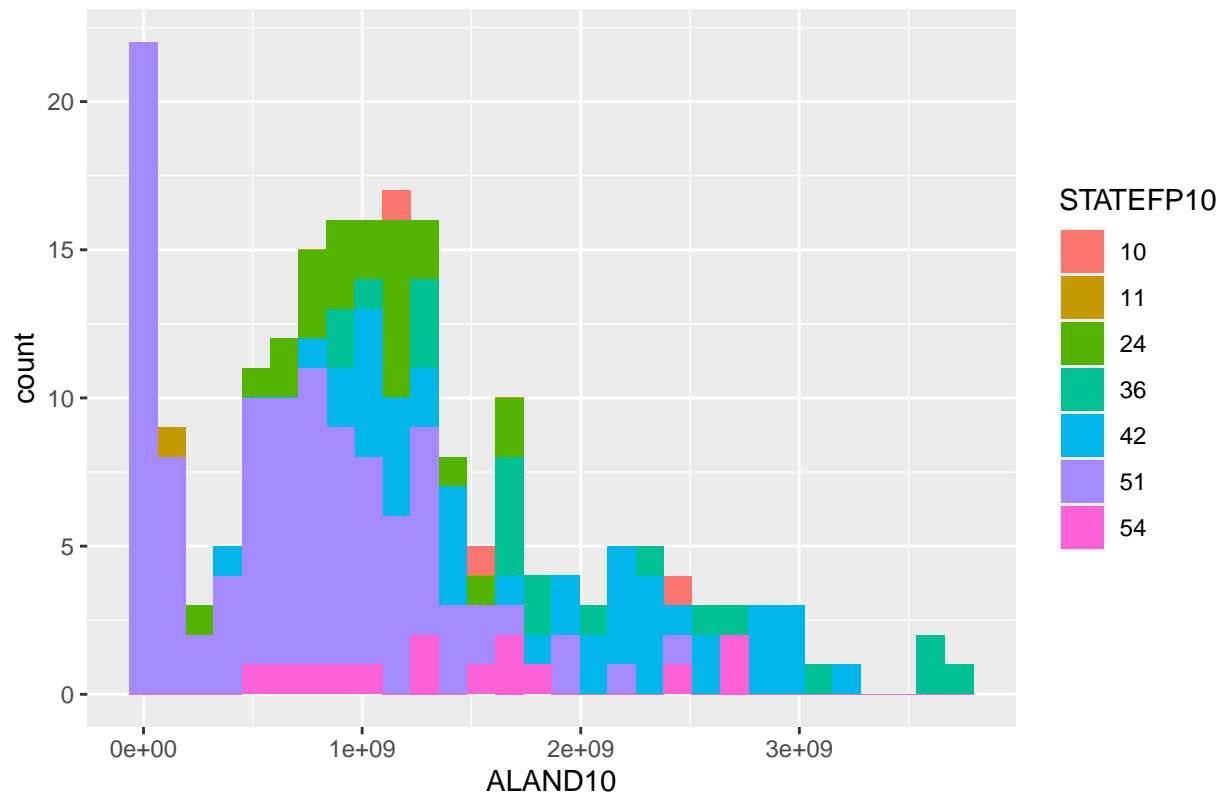


Or:

```
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'.

not the most useful plot, but you get the idea



Spatial operations

Since we have spatial data, we can perform some basic spatial operations with it. First, let's take a look at the 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]]
```

They're the same, but we can formally check

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

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

We need to make sure the files have the same CRS before we do our spatial operations using the both of them. But to make the problem more tractable, let's first pare down our data such that we only have the counties in the state of Delaware:

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

then, we can perform a *spatial intersection* to find all of the monitoring stations within our Delaware subset

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

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

Plotting this small number of points will be ok, so let's look at the data first, then check the plot:

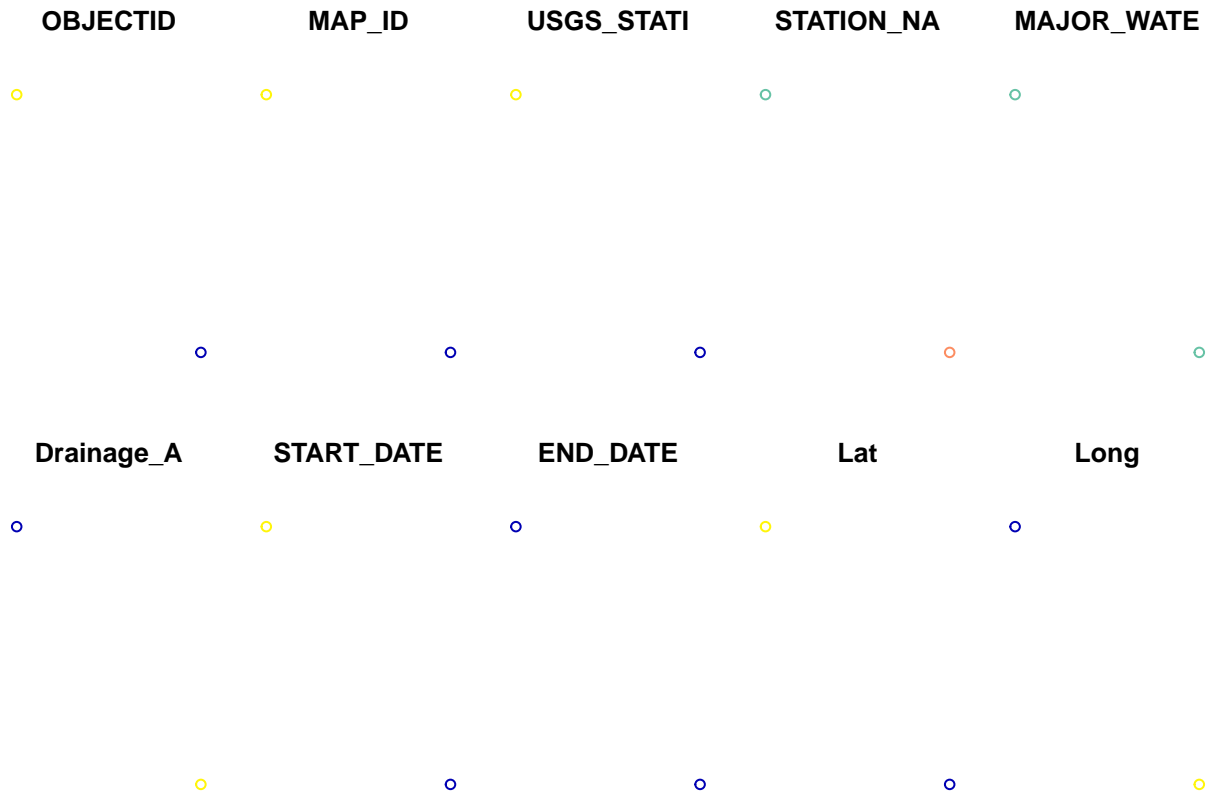
```
glimpse(del.stations)
```

```
## 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)
```

```
## Warning: plotting the first 10 out of 31 attributes; use max.plot = 31 to plot
## all
```



There are only 2 points, and the plot isn't super helpful without any other sort of spatial reference, but you've successfully completed your first spatial operation in R!

`sf` has a number of other useful functions built-in that you can try. For example, a quick calculation of the area of each county in Delaware:

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

```
## Units: [m^2]
## [1] 2065913885 3096294967 1278231146
```

Note that `sf` gives you the units of the calculation, but also that the data are in the form of a vector

Your tasks

This lab requires you to put together many of the tasks demonstrated above, in class, help documentation (don't forget the `?` command!), and in your readings. I don't expect you'll know them all immediately, so you'll need to reference those resources, your classmates, and possibly web resources as well. This process is representative of real-world problem solving in this domain. There are a very large number of packages and functions available to you in R, and no one person knows how to use them all. So be inventive, be clever, and be persistent!

Complete each task COMPLETELY USING R CODE. YOU MUST SHOW YOUR WORK FOR EACH ANSWER. Label your variables sensibly and use comments such that I can find your answers and your work.

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.
- 1.2 For each state, find the county that has the largest proportion of its land as water (water area / total area)
- 1.3 Count the number of counties in each state
- 1.4 Which station has the shortest name (STATION_NAME) in the study area?

Task 2: Plotting attribute data

...for each plot, label your axes properly and give your 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
- 2.2 Make a histogram of drainage area (Drainage_A) for all monitoring stations
- 2.3 Make a similar histogram of drainage area (Drainage_A) for all monitoring stations. This time, color each point using the state variable

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

Test it with the following vectors

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

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
- 4.2 Calculate the average size of counties in New York (that are also in this study area)
- 4.3 Calculate which state has monitoring stations with the greatest average drainage area (Drainage_A)

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?

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

2. What did you find challenging in this lab? What was new?
3. What types of activities would you like to see in labs this semester?