Assignment 9: Spatial Analysis in R

Benjamin Culberson

OVERVIEW

This exercise accompanies the lessons in Environmental Data Analytics (ENV872L) on spatial analysis.

Directions

1. Change “Student Name” on line 3 (above) with your name.
2. Use the lesson as a guide. It contains code that can be modified to complete the assignment.
3. Work through the steps, **creating code and output** that fulfill each instruction.
4. Be sure to **answer the questions** in this assignment document. Space for your answers is provided in this document and is indicated by the “>” character. If you need a second paragraph be sure to start the first line with “>”. You should notice that the answer is highlighted in green by RStudio.
5. When you have completed the assignment, **Knit** the text and code into a single HTML file.
6. After Knitting, please submit the completed exercise (PDF file) in Sakai. Please add your last name into the file name (e.g., “Fay_A10_SpatialAnalysis.pdf”) prior to submission.

DATA WRANGLING

Set up your session

1. Check your working directory
2. Import libraries: tidyverse, sf, leaflet, and mapview

```
#1.
getwd()
```

```
## [1] "/Users/benculberson/Documents/Duke /Spring 2022/Environmental Data Analytics/Environmental_Data_Analytics
_2022/Assignments"
```

```
#2.
library(tidyverse, quietly = TRUE)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.0 ──
```

```
## ✓ ggplot2 3.3.3     ✓ purrr   0.3.4
## ✓ tibble  3.0.5     ✓ dplyr   1.0.3
## ✓ tidyr   1.1.2     ✓ stringr 1.4.0
## ✓ readr   1.4.0     ✓ forcats 0.5.0
```

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

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(sf)
```

```
## Warning: package 'sf' was built under R version 4.0.5
```

```
## Linking to GEOS 3.9.1, GDAL 3.4.0, PROJ 8.1.1; sf_use_s2() is TRUE
```

```
library(leaflet)
```

```
## Warning: package 'leaflet' was built under R version 4.0.5
```

```
library(mapview)
```

```
## Warning: replacing previous import 'terra::extend' by 'raster::extend' when
## loading 'satellite'
```

```
## Warning: replacing previous import 'terra::crop' by 'raster::crop' when loading
## 'satellite'
```

```
## Warning: multiple methods tables found for 'crop'
```

```
## Warning: multiple methods tables found for 'extend'
```

Read (and filter) county features into an sf dataframe and plot

In this exercise, we will be exploring stream gage height data in Nebraska corresponding to floods occurring there in 2019. First, we will import from the US Counties shapefile we’ve used in lab lessons, filtering it this time for just Nebraska counties. Nebraska’s state FIPS code is `31` (as North Carolina’s was `37`).

3. Read the `cb_2018_us_county_20m.shp` shapefile into an sf dataframe, filtering records for Nebraska counties (State FIPS = 31)
4. Reveal the dataset’s coordinate reference system
5. Plot the records as a map (using `mapview` or `ggplot`)

```
#3. Read in Counties shapefile into an sf dataframe, filtering for just NE counties
Nebraska_counties_sf<- st_read('../Data/Spatial/cb_2018_us_county_20m.shp') %>%
  filter(STATEFP == 31) #Filter for just Nebraska Counties
```

```
## Reading layer `cb_2018_us_county_20m' from data source
##   `/Users/benculberson/Documents/Duke /Spring 2022/Environmental Data Analytics/Environmental_Data_Analytics_2
022/Data/Spatial/cb_2018_us_county_20m.shp'
##   using driver `ESRI Shapefile'
## Simple feature collection with 3220 features and 9 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -179.1743 ymin: 17.91377 xmax: 179.7739 ymax: 71.35256
## Geodetic CRS:  NAD83
```

```
head(Nebraska_counties_sf)
```

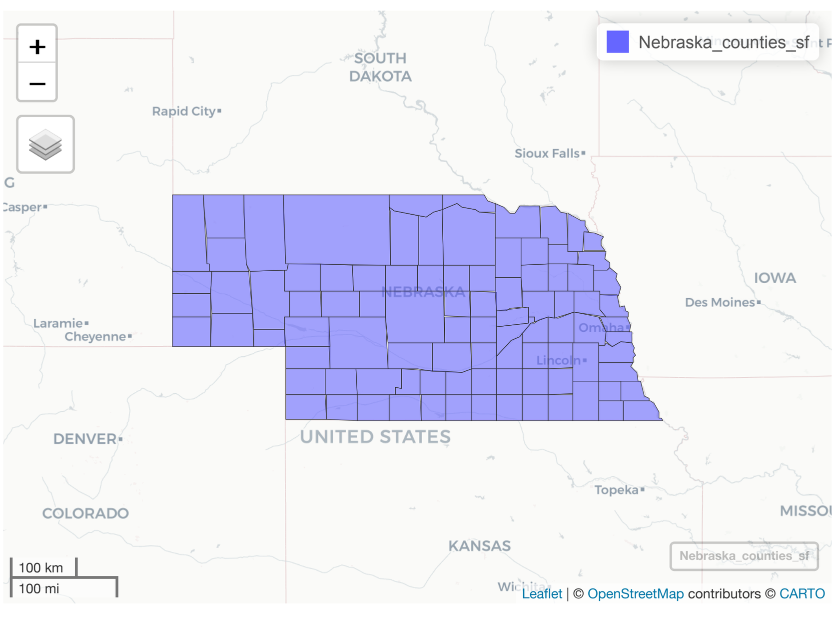
	STATEFP <chr>	COUNTYFP <chr>	COUNTYNS <chr>	AFFGEOID <chr>	GEOID <chr>	NAME <chr>	LSAD <chr>	ALAND <dbl>	AWATER <dbl>
1	31	133	00835888	0500000US31133	31133	Pawnee	06	1116478182	4852361
2	31	135	00835889	0500000US31135	31135	Perkins	06	2287828025	2840176
3	31	165	00835904	0500000US31165	31165	Sioux	06	5352724893	1681023
4	31	151	00835897	0500000US31151	31151	Saline	06	1486812299	5821039
5	31	137	00835890	0500000US31137	31137	Phelps	06	1398048574	1646533
6	31	143	00835893	0500000US31143	31143	Polk	06	1135309851	5947393

6 rows | 1-10 of 11 columns

```
#4. Reveal the CRS of the counties features
st_crs(Nebraska_counties_sf)
```

```
## 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["latitude",north,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["longitude",east,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433]],
##     ID["EPSG",4269]]
```

```
#5. Plot the data
mapView(Nebraska_counties_sf)
```



6. What is the EPSG code of the Counties dataset? Is this a geographic or a projected coordinate reference system? (Or, does this CRS use angular or planar coordinate units?) To what datum is this CRS associated? (Tip: look the EPSG code on https://spatialreference.org)

> ANSWER: The EPSG code of the Counties data set is 4269, most commonly used by US federal agencies. This is a geographic coordinate system that uses angular coordinate units. This CRS is associated with the North American Datum 1983 (NAD83).

Read in gage locations csv as a dataframe, then display the column names it contains

Next we’ll read in some USGS/NWIS gage location data added to the `Data/Raw` folder. These are in the `NWIS_SiteInfo_NE_RAW.csv` file.(See `NWIS_SiteInfo_NE_RAW.README.txt` for more info on this dataset.)

7. Read the NWIS_SiteInfo_NE_RAW.csv file into a standard dataframe.
8. Display the column names of this dataset.

```
#7. Read in gage locations csv as a dataframe
gage_locations <- read_csv('../Data/Raw/NWIS_SiteInfo_NE_RAW.csv')
```

```
##
## ── Column specification ────────────────────────────────────────────────────────
## cols(
##   site_no = col_character(),
##   station_nm = col_character(),
##   site_tp_cd = col_character(),
##   dec_lat_va = col_double(),
##   dec_long_va = col_double(),
##   coord_acy_cd = col_character(),
##   dec_coord_datum_cd = col_character()
## )
```

```
#8. Reveal the names of the columns
colnames(gage_locations)
```

```
## [1] "site_no"            "station_nm"         "site_tp_cd"
## [4] "dec_lat_va"         "dec_long_va"        "coord_acy_cd"
## [7] "dec_coord_datum_cd"
```

9. What columns in the dataset contain the x and y coordinate values, respectively?
> ANSWER: > x coordinates are in the dec_long_va column, y coordinates are in the dec_lat_va column

Convert the dataframe to a spatial features (“sf”) dataframe

10. Convert the dataframe to an sf dataframe.

- Note: These data use the same coordinate reference system as the counties dataset

11. Display the column names of the resulting sf dataframe

```
#10. Convert to an sf object
gage_locations_sf <- gage_locations %>%
  st_as_sf(coords = c('dec_long_va','dec_lat_va'), crs=4269)

#11. Re-examine the column names
colnames(gage_locations_sf)
```

```
## [1] "site_no"            "station_nm"         "site_tp_cd"
## [4] "coord_acy_cd"       "dec_coord_datum_cd" "geometry"
```

```
#It looks like the longitude and the latitude columns are now a geometry column
```

12. What new field(s) appear in the sf dataframe created? What field(s), if any, disappeared?

> ANSWER: We now have a geometry field that looks like we can plot on top of the country data. The longitude and latitude fields disappeared.

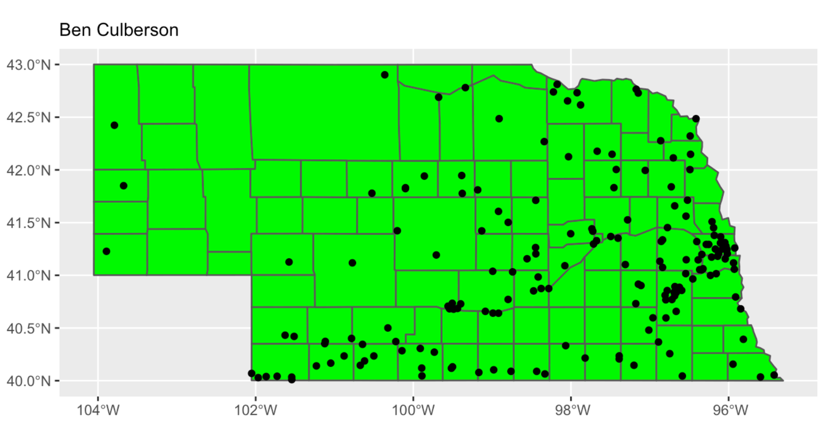
Plot the gage locations on top of the counties

13. Use `ggplot` to plot the county and gage location datasets.

- Be sure the datasets are displayed in different colors
- Title your plot “NWIS Gage Locations in Nebraska”
- Subtitle your plot with your name

```
#13. Plot the gage locations atop the county features
NWIS_Gage_Locations_in_Nebraska <- ggplot() +
  geom_sf(data=Nebraska_counties_sf,fill='green') +
  geom_sf(data=gage_locations_sf) +
  labs(subtitle = "Ben Culberson")

NWIS_Gage_Locations_in_Nebraska
```



Read in the gage height data and join the site location data to it.

Lastly, we want to attach some gage height data to our site locations. I’ve constructed a csv file listing many of the Nebraska gage sites, by station name and site number along with stream gage heights (in meters) recorded during the recent flood event. This file is titled `NWIS_SiteFlowData_NE_RAW.csv` and is found in the Data/Raw folder.

14. Read the `NWIS_SiteFlowData_NE_RAW.csv` dataset in as a dataframe.
15. Show the column names .
16. Join our site information (already imported above) to these gage height data.

- The `site_no` and `station_nm` can both/either serve as joining attributes.
- Construct this join so that the result only includes spatial features where both tables have data.

17. Show the column names in this resulting spatial features object
18. Show the dimensions of the resulting joined dataframe

```
#14. Read the site flow data into a data frame
NWIS_SiteFlowData_NE_RAW <- read_csv("../Data/Raw/NWIS_SiteFlowData_NE_RAW.csv")
```

```
##
## ── Column specification ────────────────────────────────────────────────────────
## cols(
##   site_no = col_character(),
##   station_nm = col_character(),
##   date = col_datetime(format = ""),
##   gage_ht = col_double()
## )
```

```
#15. Show the column names
colnames(NWIS_SiteFlowData_NE_RAW)
```

```
## [1] "site_no"    "station_nm" "date"       "gage_ht"
```

```
#16. Join location data to it
gage_height_location_joined <- merge(x = gage_locations_sf,
                     y = NWIS_SiteFlowData_NE_RAW,
                     by.x = 'site_no',
                     by.y = 'site_no')

#17. Show the column names of the joined dataset
colnames(gage_height_location_joined)
```

```
## [1] "site_no"            "station_nm.x"       "site_tp_cd"
## [4] "coord_acy_cd"       "dec_coord_datum_cd" "station_nm.y"
## [7] "date"               "gage_ht"            "geometry"
```

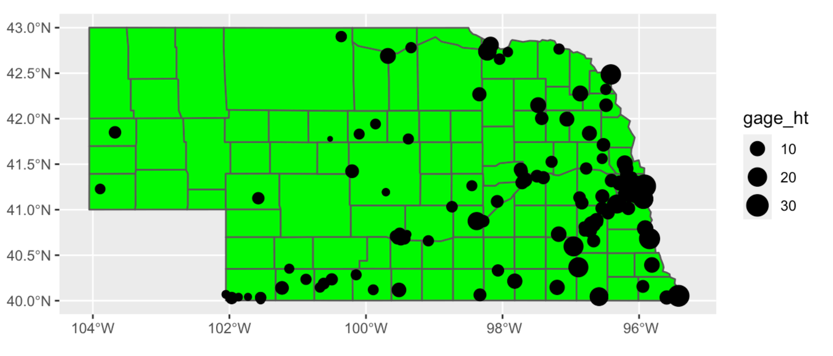
```
#18. Show the dimensions of this joined dataset
dim(gage_height_location_joined)
```

```
## [1] 136   9
```

Map the pattern of gage height data

Now we can examine where the flooding appears most acute by visualizing gage heights spatially. 19. Plot the gage sites on top of counties (using `mapview`, `ggplot`, or `leaflet`) * Show the magnitude of gage height by color, shape, other visualization technique.

```
#Map the points, sized by gage height
ggplot() +
  geom_sf(data=Nebraska_counties_sf,fill='green') +
  geom_sf(data=gage_height_location_joined,aes(size=gage_ht))
```



SPATIAL ANALYSIS

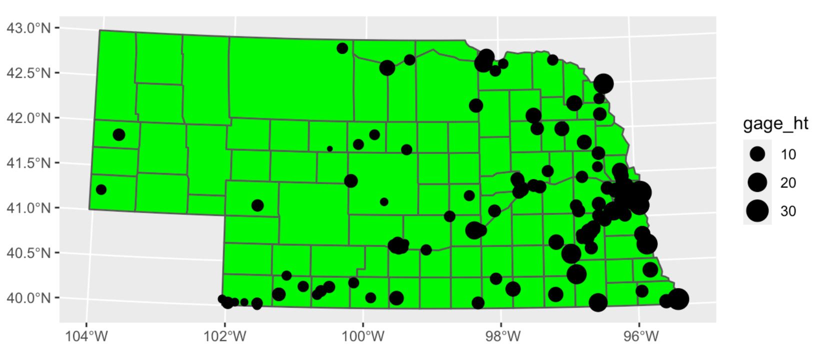
Up next we will do some spatial analysis with our data. To prepare for this, we should transform our data into a projected coordinate system. We’ll choose UTM Zone 14N (EPGS = 32614).

Transform the counties and gage site datasets to UTM Zone 14N

20. Transform the counties and gage sf datasets to UTM Zone 14N (EPGS = 32614).
21. Using `mapview` or `ggplot`, plot the data so that each can be seen as different colors

```
#20 Transform the counties and gage location datasets to UTM Zone 14
Nebraska_counties_sf_utm <- st_transform(Nebraska_counties_sf, crs = 32614)
gage_height_location_joined_utm <- st_transform(gage_height_location_joined, crs = 32614)

#21 Plot the data
ggplot() +
  geom_sf(data=Nebraska_counties_sf_utm,fill='green') +
  geom_sf(data=gage_height_location_joined_utm,aes(size=gage_ht))
```



Select the gages falling within a given county

Now let’s zoom into a particular county and examine the gages located there. 22. Select Lancaster county from your county sf dataframe 23. Select the gage sites falling `within` that county * Use either matrix subsetting or tidy filtering 24. Create a plot showing: * all Nebraska counties, * the selected county, * and the gage sites in that county

```
#22 Select the county
Lancaster_sf <- Nebraska_counties_sf_utm %>%
  filter(NAME == "Lancaster")

#23 Select gages within the selected county
Lacaster_intersect <- gage_height_location_joined_utm[Lancaster_sf,]

#24 Plot
myMap = mapview(Lancaster_sf,
                col.regions = 'green',
                alpha.regions = 0.2,
                map.types = "CartoDB.Positron",
                legend = FALSE)

myMap + Lacaster_intersect + mapview(Nebraska_counties_sf_utm, alpha.regions = 0)
```