# Spatial Analysis in R using sf

Matthew L. Sisk

December, 2024

## Spatial Analysis Examples

This is a quick overview of various spatial analysis techniques. As students ask me questions, I will add them here.

### Matching corrdinate systems.

All of the data I have given you so far is in lat/long using the WGS84 coordinate system, but there may be cases where the strings used to define them do not match perfectly. Fortunately, it is easy to just assign the string from one layer to the next. Please note, this will not work if they are actually two different systems being used.

```r
library(sf)


districts <- st_read("..\\Week03\\Demonstrations\\SampleData\\City_Council_Districts.shp")
```

```
## Reading layer 'City_Council_Districts' from data source
##   'C:\Users\msisk1\Documents\GIT\teaching\Data-Viz-2024-Fall\Week03\Demonstrations\SampleData\City_C
##   using driver 'ESRI Shapefile'
## Simple feature collection with 6 features and 14 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -86.36085 ymin: 41.59745 xmax: -86.19133 ymax: 41.76027
## Geodetic CRS:  WGS 84
```

```r
facilities.points <- read.csv("..\\Week03\\Demonstrations\\SampleData\\Public_Facilities.csv")
```

If you run st_crs() on a layer with spatial information (like the districts shapefile below), it will return the coordinate system used to define it.

```r
st_crs(districts)
```

```
## 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]]
```

Now you can use this string as the coordinate system for another layer when you create a new spatial layer from a table

```
facilities.spatial <- facilities.points %>%
    st_as_sf(coords = c("Lon","Lat")) %>%
    st_set_crs(st_crs(districts))
```

This only works if the coordinate systems are the same. If they are totally different coordinate systems (like lat/long for one and google's Web Mercator for another), you would use the st_transform() function to reproject, or convert, the coordinates from one to the other. Just ask if you run into this problem (I think everything should be fine with the data I have provided), but here is an example

```
google.crs <- 3857 # the EPSG code for the google maps / web mercator CRS

districs.google <- districts %>% st_transform(crs = google.crs)
st_crs(districs.google)
```

```
## Coordinate Reference System:
##   User input: EPSG:3857
##   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]]
```

## Counting the points in Polygons

To count the number of points within each polygon we can use the st_intersects() function. This creates a list of the polygon that each point falls within (blank). Note that the only remaining information from the points is the row.name, which can be used to merge or cbind the info back onto the original spatial data frame. In this case, a lot of the points are not within a polygon (hence the NAs)

```
# districts <- st_make_valid(districts)

ov <- st_intersects(districts, facilities.spatial)
ov
```

```
## Sparse geometry binary predicate list of length 6, where the predicate
## was 'intersects'
##  1: 9, 11, 23, 38, 42, 45
##  2: 7, 14, 41
##  3: 17, 20, 32, 51
##  4: 47
##  5: 4, 19
##  6: 13, 18
```

However, this returns a complicated list of IDs for each point. Often, the easiest thing is just to figure out what polygon each point is in and then summarize based on that.

## Joining polygons onto points

Here is an example of doing this with the st_join function. This would then make it very easy to filter the points based on their district in shiny.The select is just removing all of the extra columns from the districts table.

```r
ov <- st_join(x = facilities.spatial, y = districts %>% select(Dist))
ov %>%
  st_set_geometry(NULL)%>% #just removing the geometry
  group_by(Dist)%>%
  summarize(facilites = n())
```

```
## # A tibble: 7 x 2
##   Dist  facilites
##   <chr>     <int>
## 1 1301          6
## 2 1302          3
## 3 1303          4
## 4 1304          1
## 5 1305          2
## 6 1306          2
## 7 <NA>         33
```

## Calculating centroids for polygons

Centroids are the center point of polygons. They can be useful for quicker spatial analysis or to find out generally what a polygon in within. By creating centroids we can essentially convert a polygons layer into a points layer.

```r
districts.center <- st_centroid(districts)
```

```
## Warning: st_centroid assumes attributes are constant over geometries
```